MANUFACTURING COMPLIANCE ANALYSIS FOR ARCHITECTURAL DESIGN:

A KNOWLEDGE-AIDED FEATURE-BASED MODELING FRAMEWORK

A Dissertation
Presented to
The Academic Faculty

by

Francisco Valdes

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
College of Architecture

Georgia Institute of Technology
May 2016

MANUFACTURING COMPLIANCE ANALYSIS FOR ARCHITECTURAL DESIGN: A KNOWLEDGE-AIDED FEATURE-BASED MODELING FRAMEWORK

Approved by:

Dr. Russell Gentry, Advisor
College of Architecture
Georgia Institute of Technology

Dr. John Haymaker
Director of Research
Perkins & Will

Dr. Charles Eastman, Co-Advisor
College of Architecture
Georgia Institute of Technology

Mr. Kevin Caravati
Senior Research Scientist
Georgia Tech Research Institute

Dr. Jason Brown
College of Architecture
Georgia Institute of Technology

Date Approved: March 16th, 2016

To my parents and brothers for their love and unconditional support.

ACKNOWLEDGEMENTS

After this life-changing academic journey, firstly, I'd like to thank my advisor, Dr. Russell Gentry and my co-advisor Chuck Eastman for all their support, kindness, and inspiration during all these years. The outcome of this dissertation has been mostly possible because of their exceptional knowledge of the field and their encouraging attitude. In addition, I'd like to thank the rest of my committee, Dr. John Haymaker, Dr. Jason Brown, and Mr. Kevin Caravati for they important involvement during my dissertation proposal and final defense. Furthermore, I'd like to thank the Georgia Tech Research Institute and Mr. Gary McMurray, Food Processing Technology Division Chief of the Aerospace, Transportation, and Advanced Systems laboratory for all the understanding and financial support that allowed me to achieve this important academic milestone. Also, I would like to thank Fulbright, the Comision Nacional de Ciencia y Tecnologia of Chile, and Universidad Tecnica Federico Santa Maria for their essential financial and organizational support during my Ph.D.

Besides my committee and financial supporters, this dissertation would not have been possible without the help of many other significant people. I would like to thank Mr. Stephen Forrest, from Maplesoft, for all his transcendental assistance during the implementation stage of this project; Dr. Ann Carpenter for her endless support, insightful comments, and dedicated reading of my dissertation; and Mr. Andres Cavieres for our daily discussions about the future of our field. Furthermore, I'd like to thank my friends of the Section 313 at the NASA Jet Propulsion Laboratory, Dr. Sebastian Herzig, Mr. Chris Delp, and Dr. Nicholas Rouquette, who helped me to get started in the Model Based Systems Engineering domain that later became a critical aspect of my dissertation.

Also, I would like thank my friends of the College or Architecture Mrs. Paula Gomez, Mr. Marcelo Bernal, and Mr. Pedro Soza for all their moral support and help during this academic process. Finally, I would like thank Robin Tucker from the College of Architecture and Christy Rackness from the Office of International Education for helping me to efficiently navigate all administrative and institutional matters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF EQUATIONS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

ANSI.	American National Standards Institute
AITC.	American Institute of Timber Construction
AEC.	Architecture, Engineering, and Construction
ACI.	American Concrete Institute
API.	Application Programing Interface
AREMA.	American Railway Engineering and Maintenance of way Association
ASCC.	American Society of Concrete Contractors
ASME.	American Society of Mechanical Engineers
AISC.	American Institute of the Steel Construction
ASTM.	American Society for Testing and Materials
AWS.	American Welding Society
BIM.	Building Information Modeling
BOS-X.	Balance of Systems in photovoltaics
BDD.	Block Definition Diagram
CAD.	Computer Aided Design
CAA.	Computer Aided Analysis
CASE.	Council of American Structural Engineers
CAM.	Computer Aided Manufacturing
CFD.	Computational Fluid Dynamics
COMPASS.	Comprehensive Modeling for Advanced Systems of Systems
CSG.	Constructive Solid Geometry
CWM.	Common Warehouse

DAG.	Directed Acyclic Graph
DSL.	Domain Specific Language
FEA.	Finite Element Analysis
GD&T.	Geometric Dimensioning and Tolerancing
GUM.	Guide to the Expression of Uncertainty and Measurement
GTRI.	Georgia Tech Research Institute
HCT.	Heterogeneous Construction Tolerances
IBD.	Internal Block Diagram
IFC.	Industry Foundation Classes
IRDS.	Information Resource Dictionary Systems
ISO.	International Organization for Standardization
IT.	Information Technology
KAOS.	Knowledge Acquisition in Automated Specification
KCMA.	Kitchen Cabinet Manufacturers
KBE.	Knowledge Based Engineering
LL.	Lower Limit tolerances
MBMA.	Metal Building Manufacturers Association
MBSE.	Model Based System Engineering
MDD.	Model Driven Development
MOF.	Meta-Object Facility
NC.	Numeric Control
NX.	Siemens NX
OMG.	Object Management Group

PAR. Parametric Diagram

PV. Photovoltaics

PVC. Polyvinyl Chloride

QUPER. Quality PERformance

RCSC. Research Council on Structural Connections

RTM. Requirements Traceability Matrix

RFI. A written request for information or clarification generated during the construction phase of the project

ROI. Return Of Investment

RSS. Root Sum Square

SCRAM. Scenario Requirements Analysis Method

SDCT. Single Domain Construction Tolerances

SE. Systems Engineering

SEMP. Systems Engineering Management Plan

SJI. Steel Joist Institute

SM. Sheet Metal

SoS. Systems of Systems

STEP. Standard for The Exchange of Product model data

STREAM-A. Strategy for Transition between Requirements and Architectural Models for Adaptive Systems

SysML. System Modeling Language

UML. Unified Modeling Language

UL. Upper Limit tolerances

WC. Worst Case scenario

WDMA. Windows and Door Manufacturing Association

SUMMARY

Given that achieving nominal (all dimensions are theoretically perfect) geometry is challenging during building construction, understanding and anticipating sources of geometric variation through tolerances modeling and allocation is critical. However, existing building modeling environments lack the ability to support coordinated, incremental and systematic specification of manufacturing and construction requirements. This issue becomes evident when adding multi-material systems produced off site by different vendors during building erection. Current practices to improve this situation include costly and time-consuming operations that challenge the relationship among the stakeholders of a project. As one means to overcome this issue, this research proposes the development of a knowledge-aided modeling framework that integrates a parametric CAD tool with a system modeling application to assess variability in building construction. The CAD tool provides robust geometric modeling capabilities, while System Modeling allows for the specification of feature-based manufacturing requirements aligned with construction standards and construction processes know-how. The system facilitates the identification of conflicting interactions between tolerances and manufacturing specifications of building material systems. The expected contributions of this project are the representation of manufacturing knowledge and tolerances interaction across off-site building subsystems to identify conflicting manufacturing requirements and minimize costly construction errors. The proposed approach will store and allocate manufacturing knowledge as Model-Based Systems Engineering (MBSE) design specifications for both single and multiple material systems. Also, as new techniques in

building design and construction are beginning to overlap with engineering methods and standards (e.g. in-factory prefabrication), this project seeks to create collaborative scenarios between MBSE and Building Information Modeling (BIM) based on parametric, simultaneous, software integration to reduce human-to-data translation errors, improving model consistency among domains.

Important sub-stages of this project include the comprehensive review of modeling and allocation of tolerances and geometric deviations in design, construction and engineering; an approach for model integration among System Engineering models, mathematical engines and BIM (CAD) models; and finally, a demonstration computational implementation of a System-level tolerances modeling and allocation approach.

CHAPTER 1: Introduction

The following extract from ACI 117R-90, "Commentary on Standard Specifications for Tolerances for Concrete Construction and Materials" condenses the need for construction tolerances, and some principles that should be applied in selecting proper tolerances [1]:

"No structure is exactly level, plumb, straight, and true. Fortunately, such perfection is not necessary. Tolerances are a means to establish permissible variations in dimensions and location, giving both the designer and the contractor parameters within which the work is to be performed. They are the means by which the designer conveys to the contractor the performance expectations upon which the design is based or the use of the project requires. Such specified tolerances should reflect design assumptions and project needs, being neither overly restrictive nor lenient. Necessity rather than desirability should be the basis of selecting tolerances." [2]

In building construction, it is common that after the execution of a project, certain stakeholders will not be pleased with the manufacturing accuracy or overall quality of the final product [3]. For Instance design and construction failures in the Frank Gehry's MIT $300 million Stata Center resulted in pervasive leaks, cracks and drainage problems that have required costly repairs [4]. In some cases the contractor finds that some components will not come together during building erection [5] which requires repairs to the assemblies, or it could be that the architect or the owner have concerns that the walls are not straight or the slabs are not flat enough [5]. This often occurs because the construction process was not precise enough, specifications were not properly communicated [6], or, as is often the case, because the suggested requirements of

tolerances were unachievable or unrealistic. One important issue that continues to spread in construction projects and contributes to cost and schedule growth is design changes and errors [7] [8] [9]. The genesis of these situations are hard to trace during the building lifecycle, but in the end they require extra project time and costly repairs. Chapter 1 of this dissertation starts by describing this problem, introducing important concepts such as nominal geometry, tolerances, and geometric variability; and concludes by proposing a high-level modeling framework to overcome the weaknesses of the current approach.

During the design stages of a building, it may be assumed to be that the geometric CAD models are dimensionally perfect – that is to say, that the features and parts in the model contain exactly the geometry that is desired in the final building[1]. This representation is known as nominal geometry. However, nominal geometry is not achievable during the stages of construction. On the contrary, there are a number of factors leading to a resulting building that differs geometrically from the nominal model:

- Complex building assemblies made by human labor;
- Unpredicted deviations from manufacturing processes;
- Incomplete manufacturing documentation and knowledge;
- Improper assumptions about materials and processes during design stages;

---

[1] This dissertation assumes that CAD models are able to completely and accurately describe the geometry of the parts and assemblies. This dissertation does not address modeling mistakes, which are inaccuracies made in the nominal geometry which lead to geometrically or functionally inadmissible models.

- Addition of different material systems with different levels of variability;
- Reaction of materials to forces and temperature changes, and building behavior;

These factors produce significant geometric deviations that must be considered, accommodated, and mitigated as part of the construction requirements and specifications development. The main formal modeling element used to prescribe these kinds of geometric deviations is known as a tolerance. Tolerance has many different meanings based in the field that it applies. For this dissertation a tolerance is defined as the permissible limit or limits of variation in a physical dimension [10]. Although the concept of tolerance is broadly understood, applicability of construction specifications and tolerances allocation have not been adequately established due to the lack of knowledge integration during design stages, and the lack of multidisciplinary coordination among different stakeholders of a building project. Furthermore, many of the construction requirements or specifications cannot be assured from the beginning because they evolve and transform during the course of a project. Early decisions about tolerances and clearances are usually made based on improper assumptions, or without an understanding of the "big picture" with respect to system implications. Decisions made late in a design or construction stage are often taken without knowledge or consideration of earlier decisions, or without understanding of the effects that these changes will produce in other material systems. In any of these cases, as-built geometric deviations obtained in construction are much larger than commonly expected [11]. While early multidisciplinary integration and constant coordination efforts under a BIM-augmented workflow are certainly important means to reduce geometric variability problems [12] [13], they are

not sufficient. Current tools and methodologies lack the ability to support coordinated, incremental, and systematic specification of tolerances requirements and the set of interactions that emerge across them during building lifecycle. Also, for the most common design-bid-build project delivery system, the team includes design professionals, a construction manager or general contractor, and many subcontractors [14]. In the early part of the project, the design team is primary – but in the later stages the general contractor assumes primacy. And so in this case, the responsibility for addressing tolerance incompatibility issues is often not clearly defined. In managed contractual systems in which the construction manager does not self-perform the work, field personnel may not be familiar with the manufacturing specifications of the project, and they are also less likely to anticipate tolerance requirements and incompatibility problems [1].

During the development of a building project the state of knowledge about construction tolerances is diffuse, and no stakeholder has access to the entire knowledge base about material-specific manufacturing, or what dimensional tolerances are realistic to prescribe. A common example of a geometric variability, which is the deviation range of the nominal geometry of a part or assembly, occurs when designers make late changes to reduce construction costs associated with some building component (e.g. to replace welding in steel connections of a roof structure with bolted connections). While a modification may satisfy the specific construction requirement goal (e.g., reduce installation time), the systems-level implications and long-term side effects are usually not well understood (e.g. bolted connections may allow more movement at the joints, increasing deflection, leading to poor rain drainage, leakage, corrosion, and air

infiltrations) and even if the problem is identified qualitatively, there exists no modeling framework in which to assess the implications of the problem quantitatively. The following list of typical tolerances compatibility problems in modeling and among different material systems [1] has been presented by an inter-industry working group. These problems have been subdivided into four main areas: (1) Tolerance Modeling and Simulation, (2) Building Behavioral Modeling, (3) Manufacturing Knowledge Documentation and Coordination, and (4) Process Standards; and was hosted by the American Society of Concrete Contractors (ASCC) and co-sponsored by several other important construction organizations such as the American Concrete Institute, American Institute of Steel Construction, American Society of Civil Engineers Construction Institute, among others.

Tolerances Modeling and Simulation:

- A modeling approach to determining conformance with stated tolerances is needed.
- Steel connections may require three-dimensional adjustability when steel and concrete dimensions are at their tolerance extremes.
- Anchor bolts embedded in concrete for steel connections may be incorrectly positioned—laterally or vertically—or may be bent after being correctly placed. Field solutions are often available, but increase cost significantly (Figure 1).
- Tolerances at the interface between precast cladding panels and the structural frame are critical. Cladding must be capable of field adjustability.
- Out-of-square or out-of-plane racking interferes with operation of windows and doors, mars appearance, and decreases resistance to water and air infiltration.

- Installation costs for windows, doors, and curtain walls increase when openings are too large or too small, embeds are improperly located, or floor edges or columns are not properly aligned.

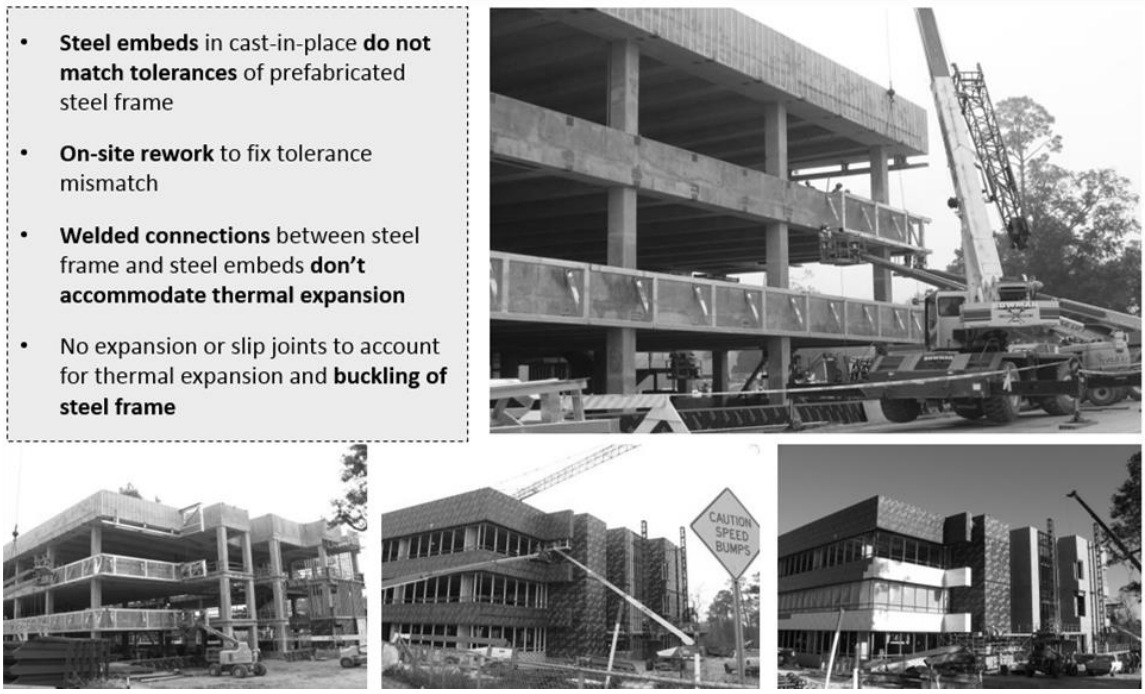


Figure 1: Example of tolerances incompatibility of cast-in-place concrete with prefabricated steel frame.

Building Behavioral Modeling:

- Windows, doors, and curtain walls in concrete openings must be designed to accommodate construction tolerances and building movement after construction.
- Three-way adjustment is needed to allow for alignment changes. Field fixes to accommodate out-of-tolerance openings may not be structurally sound or allow the needed movement after construction.
- Doors and windows that open and close require especially tight tolerances to operate properly.

Manufacturing Knowledge Documentation and Coordination:

- Manufacturing requirements are not always clearly written and are thus subject to differing interpretations by members of the construction team.
- Geometry does not comply with material-specific manufacturing rules.
- When tolerances have been allocated, every material system complies independently with its own manufacturing rules, without considering any heterogeneous materials assembly or mating conditions. This situation leads to tolerances incompatibility.
- Multiple tolerances allocated by different contractors for the same building component create disputes about which tolerance should be used.
- Project documents should clearly indicate how tolerance measurements will be made, who will make them, what corrective actions are needed when tolerances are exceeded, and who is responsible for taking the corrective actions.
- For precast cladding operations, the structural engineer does the slab drawings and the architect does the cladding details. The concrete contractor who builds to the structural drawings often does not see the cladding details. But if the detail allows little or no tolerance, and the slab is built to common ACI 117 tolerances, panels may not fit and the concrete contractor is blamed.

Processes Standards:

- Because there are no measurement protocols for many tolerances, disputes about conformance with tolerances sometimes result.

From all the groups of tolerances compatibility problems presented above, the main focus of this dissertation is the Manufacturing Knowledge Documentation and Coordination category.

1.1 Impact of Geometric Variability in the Building Industry

Failure to predict geometric variability during design stages and failure in the appropriate application of construction tolerances may contribute to the following issues: cracks in walls, cladding, and tiles; buckling; building condensation; leaky facades; structural collapse; poor visual results; poor energy performance; window and door defects; curtain wall defects; mechanical equipment installation defects; and unexpected clashes, among others. These problems result in redundant work on the construction site, demolition of defective work, lost time, failure to meet construction specifications, disputes among stakeholders, and, potentially, a financial burden on the occupier or owner [15].

With regard to cost impact, the average cost of design-attributable errors is about 14 percent of contract value [16], which is approximately the total budget dedicated to design fees. Also, as can be seen in Figure 3, design errors and omissions (D1) have the maximum impact and maximum likelihood of all the different risk categories of construction [17], followed by construction cost overruns (C1), which are also commonly related to geometric variability and re-work problems [18]. In order to reduce these issues, BIM tools need to be able to represent a building at a whole-system level, capturing the functional and behavioral relationships that span across different domains, material systems, and lifecycle stages. It is in modeling these relationships that the identification of conflicts among tolerances requirements and manufacturing

specifications can be facilitated.

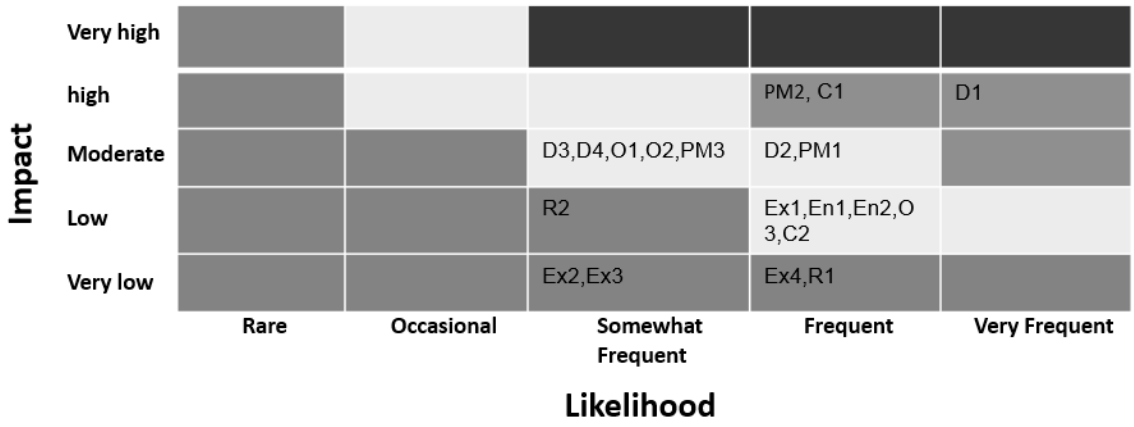


Figure 2: Construction Risk Matrix: adapted from: http://dx.doi.org/10.5772/51460 (Intech). Construction Risk Categories: Design Risk (D): External Risk (E): Environmental Risk (En): Organizational Risk (O): Project Management Risk (PM): Right of Way Risk(R): Construction Risk (C).

1.2. General Comparison between Aerospace and Construction Modeling Methods

During the past decades, aerospace engineering have improved their approach to managing geometric variability and manufacturing knowledge [19]. The aerospace industry has taken advantage of modern computer-aided manufacturing technology to integrate CAD tools with manufacturing processes. For example, the Active Workspace tool was created at Siemens and used for the development of the Curiosity Mars Rover at the Jet Propulsion Laboratory [20]. This tools supports a systems engineering driven product development process. It is systems engineering that allows linking together all the disparate elements of a product design into an intelligent product model, which can be continuously validated over its lifecycle. It is the key to enabling true model-based development [20]. Hence, the aerospace realm has dramatically reduced the need for human translation or interpretation of project data. However, an aerospace approach

cannot be directly applied to construction due to critical differences among current practices in these two domains.

In building design and construction, drawings are mostly interpreted by human labor and, there is a high risk of ambiguity and the chance for accumulated measurement error. For example, in the way most construction measurements are expressed, the dimension or size may or may not indicate the accuracy of the measurement. These tolerances may be specified in a national standard which is included as a project requirement by reference, but it is likely that neither the designer nor the general contractor is aware of the implications of the tolerance requirements. The tolerance information is, at least potentially, available, but it is not part of the modeling or fabrication process. It is an unmet requirement. Also, the level of accuracy while translating measurements from drawings to real parts and assemblies usually varies from worker to worker or even from measuring system to measuring system [21]. As a result, construction processes executed by human labor are highly stochastic in their outcomes. In contrast, in the aerospace engineering domain, automated methods of manufacturing promote tight levels of accuracy in their measurements that result in high quality products. Similarly, because most of the aerospace manufacturing processes are mostly repetitive, production of mechanical parts relies on high cost tooling and dies instead of the one-at-the-time approaches often used in building construction.

Despite all these differences, building construction and aerospace engineering share numerous guidelines concerning geometric deviations taxonomy, manufacturing requirements, and process standardization [22]. Furthermore, based on the normalization

of cross-field CAD platforms[2], such as BIM, together with the development of highly engineered building products, the construction industry is undertaking an exponential modernization [23]. As an illustration, modern building construction processes are shifting from on-site centered to in-factory centered. By taking advantage of better supply chain, specialized factories, and controlled production environment, modern building products companies are ensuring higher quality control and better working environment while reducing overall time to market of projects.

Another important engineering advancement, which constitutes a critical focus of this dissertation, has occurred in the intersection of information technology and industrial engineering. The expansion of Systems Engineering[3] (SE) has enabled the development of model-centric architectures, and the ability to integrate numerous domain-specific tools in a single computer application and modeling language. This dissertation demonstrates how the tools promulgated by SE can generate new collaborative environments that allow geographically and functionally distributed groups of stakeholders to facilitate the process of tolerances and knowledge allocation in

---

[2] This dissertation uses the term CAD for the generic 3D model, and the terms BIM and solid modeling when some of the specific features of these modeling paradigms are referenced.

[3] Systems Engineering is an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the development cycle, documenting requirements, then proceeding with design synthesis and system validation while considering the complete problem

construction models. The following section introduces a general description of how SE could support a knowledge-aided modeling environment for construction.

1.3. Towards an Integrated Modeling Approach for Building Design

In the SE field, the development of a mature Model Based System Engineering (MBSE) approach allows the management of multiple domains and applications in a progressively complex Information Technology (IT) environment [24] [25] [26]. MBSE is defined as a practice of applying modeling and simulation for implementing the processes and practices of SE [27]. The main characteristic of a MBSE methodology is to link different modeling requirements and views, from different domains, in a central model that allows interoperability and consistency between domains. Use of MBSE has led to the development of a general-purpose system-level architecture that allows multi-disciplinary modeling with proper levels of abstraction. One of these knowledge-modeling environment is the System Modeling Language (SysML). SysML is a general-purpose modeling language for systems engineering applications. It supports the specification, analysis, design, verification and validation of a broad range of systems and systems-of-systems [24]. As Delligatti states "MBSE and its associated language SysML promise increased modeling quality and affordability for one simple reason: The cheapest defect to fix is the one you prevented. And at the heart of this approach is this new kind of engineering artifact called the system model" [27]. Based on these characteristics and considering that current BIM tools cannot fully model tolerances requirements among different material systems, this research proposes the development of a tolerances modeling framework that integrates a parametric CAD tool with a MBSE modeling application. The CAD tool provides robust geometric modeling capabilities, while MBSE

allows the modeling of tolerances requirements from a system-level standpoint. Thus, the identification of system interactions between manufacturing requirements and specifications of building material systems is based on this CAD-MBSE integration. This framework provides high-level descriptions of manufacturing specifications on the MBSE (SysML) side, which becomes a low level description of feature-based (geometric) tolerances allocation on the CAD side. Tolerances calculations are performed by a mathematical engine, and tolerances are allocated in the CAD model.

With the aim of describing and implementing this approach, this document identifies several parallel tracks:

- Review of tolerances and geometric deviations in construction and engineering,
- Study of the likelihood of using a MBSE approach to model and store reusable manufacturing knowledge and design specifications for construction,
- Proposal of a model integration and model consistency approach among system engineering models, mathematical engines and BIM (CAD) models, and
- Development and computational implementation of a system-level tolerances modeling and allocation based on a MBSE approach.

The expected general contributions of this dissertation are the representations of manufacturing knowledge and tolerances interaction across building sub-systems to identify conflicting manufacturing requirements and minimize costly construction errors. The proposed approach stores and allocates manufacturing knowledge as MBSE design specifications of single and multiple material systems. In addition, as new techniques in building construction are beginning to overlap with mechanical engineering methods and standards (e.g. in-factory pre-fabrication), this dissertation provides examples of

integration scenarios between MBSE and BIM to reduce human data translation errors, improving model consistency among domains. In this regard, other specific expected contributions for the work presented in this dissertation are:

- Model-to-Model Transformation: Development of a structural, feature-based decomposition method of parametric CAD models into System Models;
- Model Integration Approach: Development of a parametric, simultaneous, seamless software integration for knowledge allocation, analysis, and verification to reduce human data translation;
- One Truth, multiple Model Views: Foundation of a model-centric architecture to manage manufacturing knowledge, project requirements, geometry, and design specifications in an interoperable modeling environment;
- Domain Expert Advice: Development of an automated allocation of material-specific knowledge for components and assemblies based of geometric features and material systems;
- Machine Readable/ Executable: Development of a programmatic integration of CAD geometry with manufacturing know-how through knowledge-based mathematical and logical constraints; and
- Model Consistency Approach: On-demand model-to-model and tool-to-tool consistency assessment and model data update.

1.4. Boundaries of the approach

Due to the heterogeneity of the domain knowledge and tools that have been incorporated for the development of this dissertation, it is important to establish a set of limitations and assumptions for the created framework.

Assumptions about the field of study: The field of study for this dissertation is architecture and construction. Although, this dissertation may have also contributed to other areas of engineering such as MBSE and computer science, it should be evaluated only by its contribution in its main areas of study.

Assumptions about interoperability and model integration: This dissertation does not deal with interoperability in the sense of creating standard neutral files to exchange model information among proprietary CAD or BIM tools. This dissertation proposes a model integration approach that does not require an exchange file. Rather, all commands performed in application A can be simultaneously executed in application B.

Assumption about modeling mistakes: Inaccuracies made in the nominal geometry which lead to geometrically and functionally inadmissible models, or documented design that does not reflect the designer's intent will not be considered as relevant for this dissertation. For example a bolt will not fit through a hole because the two parts do not line up due to a modeling mistake in applying mating conditions to an assembly. Rather, it is assumed that all models are nominally perfect, and compliant with the rules of solid modeling.

Assumptions about design errors: It is the focus of this dissertation to deal with errors discovered as part of the model integration methodology proposed for the present computational implementation. Design errors emerge when design, as documented, does reflect the designer's intent, but that intent is flawed [28]. By applying design and construction specifications, along with material-specific knowledge, to the nominal geometry, the application will determine if the proposed design will either result as intended or not from the nominal geometry model.

Assumptions about mathematical models: It is not the focus of this dissertation to propose, assess, or improve any of the mathematical or statistical models used to describe geometric variability in construction. For this dissertation such models and equations will be assumed as valid and incorporated as they are described in the literature into constraints modeling elements of the implementation.

Assumptions about material-specific knowledge: It will be assumed that all the material-specific knowledge, from standards and other sources of know-how has been properly validated in each material system field. This dissertation does not focus on creation of new material-specific manufacturing knowledge. Rather, this dissertation focuses on the development of a modeling framework that allows the seamless integration of knowledge and geometry to perform simultaneous analysis for manufacturing compliance.

1.5. Organization of this Dissertation

The remainder of this dissertation is organized as follows:

Chapter 2 Hypothesis, introduces the research questions and hypothesis and then presents the expected impact and possible generalizations of this dissertation;

Chapter 3 Background, presents the overall background of this dissertation that has been divided in Geometric Dimensioning and Tolerancing; geometric variation and GD&T evolution; Model Based Systems Engineering and its associated language SysML. This section also introduces a discussion about the differences between document-centric approach and a model-centric approach for engineering. Furthermore, this section also presents the diagrams of the System Modeling Language (SysML), a section that deals

with Requirements Management in Systems Engineering; and finally a review of SysML modeling integrations and consistency management in MBSE.

Chapter 4 Tolerances in building construction, presents the challenges of modeling construction tolerances; introduces a tolerances taxonomy created for the implementation; and finally presents the main outcomes of this taxonomy: Single Domain Construction Tolerances (SDCT), and Heterogeneous Construction Tolerances (HCT).

Chapter 5 Knowledge and Tolerances Representation in Construction, discusses drawings and specifications for construction; the representation of construction tolerances, a mathematical approach to represent construction tolerances; Statistical tolerances analysis through Monte Carlo methods; and model simplification and allocation of manufacturing knowledge and tolerances on solid models.

Chapter 6 Methodology, Goes from domain issues to functionalities proposed for the modeling framework; presents a general SysML-CAD integration approach; presents an approach to SysML-CAD semantic integration through Domain Specific Languages (DSL); introduces the representation of CAD data structures in SysML; offers a general description of the present project and explain the modeling framework though a first case study : Cylindrical fit.

Chapter 7 System Evaluation, presents a second and third case study in a SDCT and a HCT domains: A multi-feature, 4 components, single-material (sheet metal) critical assembly of an architectural PV racking structure, QuadPod; and a light gauge wall assembly with eleven components and four concurrent material systems (Cast-in-place concrete, precast concrete, light gauge framing, and PVC windows);

Chapter 8 System Validation, restates the case studies developed during Chapter 7, presents 4 complementary evaluation methods for the implementation; delivers the positive aspects of the implementation, presents the found and resolved issues faced during the implementation, and finally suggests the items of the present dissertation that require further improvement.

Chapter 9 Conclusions, summarizes the motivations and approach for this dissertation, answers the research questions and assess the hypothesis, presents and develop a list of contributions of the present dissertation, and finally delivers some concluding remarks.

CHAPTER 2: Research Questions and Hypothesis

2.1. Research Questions

Based on the problem statement and motivations offered in the introduction section and further supported by a comprehensive background review (see Chapter 3), the research questions of the present dissertation are:

1. *Is it possible to represent and store machine-readable manufacturing knowledge to parametrically assess manufacturability and tolerances of CAD geometry in the early stages of building design?*
2. *Is it possible to develop a computationally-integrated modeling framework among Model Based Systems Engineering models, mathematical engines, and CAD models?*
3. *Given that questions 1 and 2 above can be answered affirmatively, can use the systems as postulated to predict conflicting tolerances interactions among different material systems from different vendors before creating building assemblies on the site?*

These research questions are integrated below in the dissertation's hypothesis which is further decomposed in detailed explanations of its core concepts. Key elements of the hypothesis are numbered (a) through (e) and are discussed in detail in the following sections.

2.2. Hypothesis

The seamless integration of parametric CAD geometry with a system-level modeling environment (a) allows the feature-based allocation of manufacturing

specifications (b), based on material-specific knowledge and processes constraints (c), and also identifies complex conflicting interactions of tolerances (d) across multi-material building assemblies(e).

The details of each aspect of the hypothesis are explained in what follows and also describes the first intent to enumerate the contributions of the present dissertation:

(a) Seamless integration of parametric CAD geometry with a system-level modeling environment, SysML

- A seamless CAD-SysML integration fills the gap between geometry-focused CAD and analysis and simulation-focused SysML through an simultaneous modeling tool. The proposed approach programmatically integrates two different data structures by recreating the meta-model[4] of the CAD application through a graph-based representation in SysML (see Section 6.7). The set of elements and rules to perform such a transformation will be called a SysML Profile or Domain Specific Language (DSL). Thus, this profile or DSL defines the elements, languages and processes from which to form a model, and will be based on the assembly> part> feature> parameter> value paradigm to describe geometry as used in most solid modeling applications.

---

[4] A meta-model is a detailed classification of the constructs and rules required for creating semantic models, which means the implementation of specific independent descriptions of the underlying algorithmic ideas [117].

- Integration of specific features of geometric data with a system modeling tool will allow rule-based design and solve operations that otherwise require manual data translation, which is error prone and time consuming. One of the main difficulties of tolerances allocation in the construction industry is that rules and values of tolerances specifications are not based in a geometric-specific context. For example, when applying tolerances to a specific building component, designers usually follow tables and standards that do not consider mating conditions between components that belong to different material systems. Also, tolerance specifications based on tables [29] are usually described in ranges instead of instance-based approaches, which reduces tolerances accuracy. This implementation proposes tolerances allocation as a factor of the critical dimension to be specified (case-based tolerances allocation).
- A CAD-SysML integration will provide geometric data to numerous domain-specific tools. For example, it will populate tolerance model equations by linking CAD critical dimensions with construction knowledge and standards, which are instantiated from SysML profiles. Although the implementation presented in this dissertation implements the integration of a single CAD application with a single system-level tool, one the of the contributions of the approach is to demonstrate that such integration could be achieved with any design or engineering tool in the building lifecycle that has an API and a data structure that can be represented as a SysML profile. In addition, this implementation is integrated with a mathematical solver that can perform calculations transferring metrics from any of the integrated tools.
- An automated CAD-SysML integration will ensure data consistency among models. As it has been explained previously, one of the main sources of geometric deviations

during construction is the lack of numeric consistency among different model views and tools. Reasons for this lack of consistency range from simple isolated data transcription mistakes to consistency issues that arise from the document-oriented nature of construction. An encoded consistency approach is one of the bases of the present implementation, and it will promote a truly model-based approach for construction.

(b) Feature-based automated allocation of manufacturing specifications

- This dissertation focuses on the integration of manufacturing specifications and geometry, as tolerances analysis and allocation processes that require geometry handling are intrinsically interconnected and codependent. This implementation, through the creation of material specific profiles in SysML, produces reusable blocks of manufacturing knowledge to assess geometric variability and tolerances allocation. Each block of manufacturing knowledge, described as a <<Design Specification>> or <<Manufacturing Specification>> in the proposed SysML profile, contains the rationale of a specific tolerances or manufacturing rule and is automatically enforced via connection to specific CAD features through mathematical expressions as <<Constraints>> (see Section 6.8). As Bernal and Haymaker [30] suggested, constraint-based methods capture design knowledge in the form of constraints and requirements that must be satisfied by the design.
- An automated integration between manufacturing knowledge and geometry is required due to the highly heterogeneous environment of domain-specific applications and languages that affect tolerances modeling and allocation. Parametric geometry tools like that used for this implementation have modest domain-specific

knowledge capabilities in a few well-understood domains. For example, sheet metal bends or flanges can be easily created because placeholders for all necessary domain-specific parameters are included in the user interface. Yet, this modeling environment does not contain proper tools to calculate and allocate a correct value for each of those domain-specific parameters. For instance, a bending radius is automatically applied when a flange is created. However, if the material thickness changes, the bending radius, which is highly dependent on material thickness, will not be updated. The required information to update these parameters is mainly contained in manufacturing specifications, managed by different stakeholders, is largely human readable, and stored in different documents. Therefore, one of the important contributions of this integration is the knowledge-based allocation of metrics for CAD feature parameters.

- A systematic approach for tolerances specification starts from high-level descriptions of manufacturing specifications on the SysML side and progress into low-level descriptions of feature-carried geometric tolerances on the CAD side. A tolerances lifecycle[5] must be embedded as part of the entire project lifecycle. The lifecycle includes building requirements that inform design specifications, and these are then

[5] For this dissertation, tolerances lifecycle represents the different stages of tolerances modeling and allocation. It starts with requirements modeling, then tolerances are converted in SDCT specifications applied to individual components; and then the SDCT evolve to HCT where multiple-material assemblies are analyzed to provide case-based tolerances and clearances allowances.

instantiated as geometric features to fulfill those original requirements. However, challenges arise due to different semantic nature and non-interoperable modeling environments of the building industry. An important aspect of this implementation is to ensure data continuity by allowing text-based requirements to be automatically traced from a geometric feature and vice versa.

- Two or more engineering views[6] can read from and write to a shared attribute of the geometric design. For this reason, the associated manufacturing knowledge, model elements and their possible parallel changes and updates have to be consistent. For instance, two component-specific geometric features that belong to two different material systems could share a mating relationship. This mating relationship will create an HCT assembly specification (e.g. clearance). However, the two same features will most likely also have manufacturing specifications that only apply within their material system (SDCT). In this case, the automated allocation of manufacturing knowledge must consider an appropriate process hierarchy to ensure that both parts of the process are complementary and not conflicting. Consequently, SysML rules correct both for internal material logics (SDCT) as well as external

[6] A view is a representation of a whole system or subsystem from the specific well-defined perspective. A viewpoint is a specification of the conventions and rules for constructing and using a view for the purpose of addressing a set of stakeholder concerns [24].

tolerance logics (HCT) , and it is important that these corrections take place in the proper sequence.

(c) Material-specific knowledge and processes constraints

- Most of the tolerances parameters of a building product depend on non-geometric rules and process modeling. Although most solid modeling applications have robust parametric capabilities that allow the creation of associations among parameters (e.g. "the length is the double of the width" or x= 2y), the rationale behind such an expression is not present in those models. Thus, there is a need for an integrated functionality that keeps a text-based specification or requirement tied to a mathematical expression to enforce its applicability.
- An automated system for tolerances allocation should identify and verify critical dimensions against current construction specifications and/or user-defined, domain-specific knowledge. This is because not all of the knowledge necessary to design building assemblies with proper understanding of its geometric variability is captured in the specification. It is important that the design methodology is extended to permit the instantiation of rules from experts. This functionality will be guaranteed by creating an encoded relation among four different model elements in SysML an element of the meta-class <<NXFeature>> that contains metrics linked from the CAD geometry and typed as <<NXValueProperty>>; an element typed as <<Design Specification>> or <<Manufacturing Specification>> that contains the rationale of a manufacturing or assembly rule; and an element of the meta-class <<Knowledge-Based Constraint>> or <<Critical Dimension>> that contains the mathematical representation of the manufacturing or assembly rule. This last element, by using

binding connections, will verify that all metrics are compliant with the rationale expressed in the <<Design Specification>>.

- Modeling material processes as special arrangements of metrics and constraints[7] will allow the formal specification of tolerances that are behavior-dependent (for instance, geometric deviations due to kinematics or temperature changes). This reusable system will be based on the element <<Analysis Context>>. The <<Analysis Context>> stereotypes are specializations of SysML blocks that are used to create system boundaries defining where to execute a domain-specific evaluation, in similar fashion of a scenario. As Gane and Haymaker [31] state, an scenario is a specific group of constraints, which restricts the context of design decisions

(d) Complex conflicting interactions of tolerances

- Representing the building as a whole system will capture the functional and behavioral interactions that occur across different domains and material systems. This will be achieved by integrating geometry, processes, and design specifications in a single modeling platform that enables the calculation of tolerances and clearances of combined tools and multiple material systems. This capability will replace the current industry approach that specifies tolerances allocation as a separate task for each material system and vendor.

---

[7] For this dissertation, metrics are numerical values assigned to model parameters, and constraints are domain-specific mathematical expressions that condition those numerical values.

- Reinforcing a system-level semantic layer on the CAD environment will facilitate the representation of geometric and non-geometric interactions of a building project.

(e) Multi-material building assemblies

- The multi-party nature of the building construction lifecycle and the uncertain outcomes from construction processes are some of the main causes of geometric variability in construction [1]. The generality of the implementation, which allows modeling new knowledge through SysML profiles and meta-classes, enables the creation of model elements that represent material-system boundaries, for example joints and clearances, and other relationships within heterogeneous assemblies.
- System-based modeling of multi-material building assemblies will result in not just tolerances attributable to fabrication accuracy but also behavioral considerations that affect their variation. Examples include the addition of materials with different mechanical properties and the addition of components fabricated by different subcontractors with dissimilar processes and standards.

2.3. Impact of Proposed Dissertation and Possible Generalizations

As it was explained in Chapter 1, design errors arise when design, as documented, reflects the designer's intent, but that intent is flawed due to a lack of information or due to wrong design assumptions [28]. A survey research by Lopez and Love [32] estimated design error costs obtained from 139 building construction projects. The mean direct and indirect costs for design errors were estimated to be 6.85 percent and 7.36 percent of contract value, respectively. This totals more than 14 percent of the project contract in design attributable errors alone. Also, as Li [16] presents, "The proportion of money and time spent on rework in the design phase is usually higher than that of the construction

phase, as design is an iterative process during which engineers try to solve coupled problems with complex relationships." In the same regard, as it can be seen in the MacLeamy Curve (Figure 3), costs and time associated with errors or conflict correction increase substantially if the error is identified after construction documentation is complete. Although the promise of completely eliminating design errors seems impractical, this project will help reduce the time and cost associated with tolerances-related design issues including on-site re-work, demolition of defective work, and disputes among stakeholders.

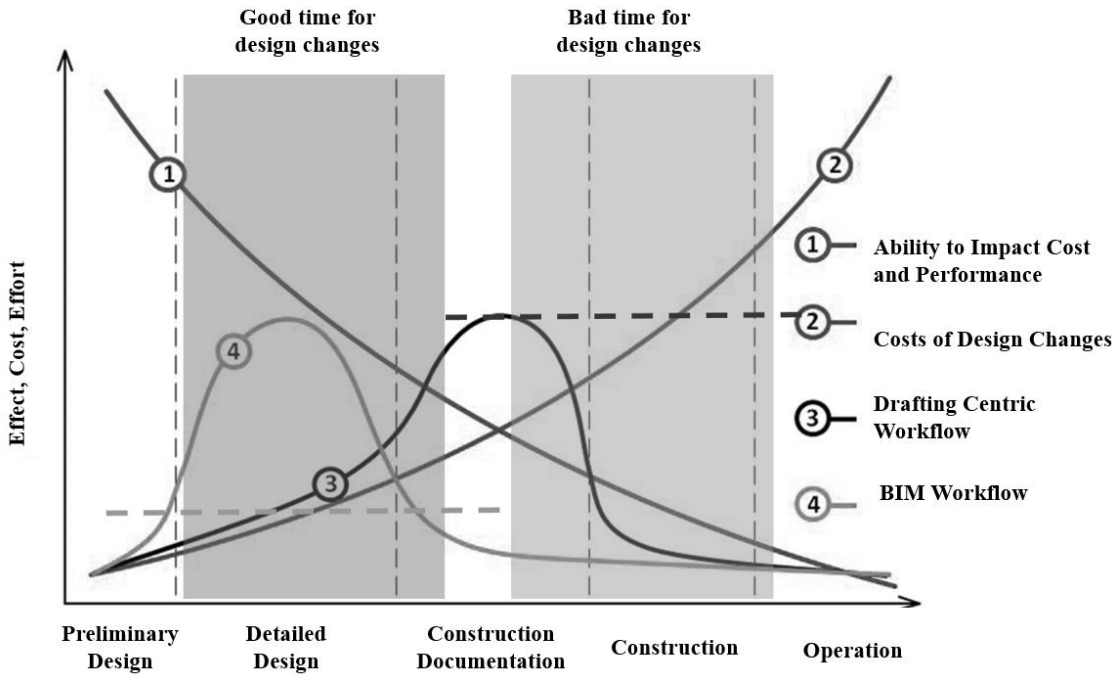


Figure 3: Impact of design changes during building lifecycle (adapted from Patrick MacLeamy)

This project supports the early identification of conflicting manufacturing and performance requirements and minimize costly construction errors by representing tolerances interaction across different building sub-systems. This objective is achieved by integrating a BIM tool (Siemens NX) with a system engineering tool (SysML), and a

mathematical simulation engine for analysis calculations (Maple 17-18). This integration is intended to support the collaborative modeling of a building project as a "system-of-systems," and to provide the computational infrastructure and knowledge necessary to fix conflicts when they are detected.

Another important contribution of this dissertation, which can be generalized in different aspects of building design, is to formulate and execute more consistent analysis and simulations by incorporating parametric CAD data into a system modeling environment. In a highly specialized and heterogeneous modeling environment as described by Haymaker [33], a parametric software integration will reduce human data translation errors, improving model consistency among domains. Furthermore, an integrated multi-system view of structure and behavior will enable the comparison of alternatives based on trade-offs and risks. Also, this implementation can integrate different modeling requirements tools, from different engineering fields, in a central model that enhances interoperability and consistency among domains.

CHAPTER 3: Background Review

The background section will be divided in several critical areas related to this project: Geometric Dimensioning and Tolerancing (GD&T), geometric variation and GD&T Evolution, Model Based System Engineering (MBSE), System Modeling Language (SysML) and BIM, document-centric versus model-centric approaches, diagrams of the SysML language, requirements management, SysML modeling integration, and consistency management in MBSE.

3.1. Geometric Dimensioning and Tolerancing (GD&T)

Geometric Dimensioning and Tolerancing (GD&T) is an engineering approach to describe a nominal – or theoretically perfect – geometry of parts and assemblies and to subsequently describe the allowable difference in form and size of individual features and the allowable variation between features from this theoretically perfect geometry [15]. Tolerances specifications are a set of rules that are applied to different types of relations among geometric features. Figure 4 presents a geometric tolerances taxonomy common to most of the engineering domains. This taxonomy, although not specific to building construction, refers to the general geometric representations of deviations of parts and processes. Accordingly, the level of abstraction of this taxonomy is appropriate to represent tolerances of the construction industry.

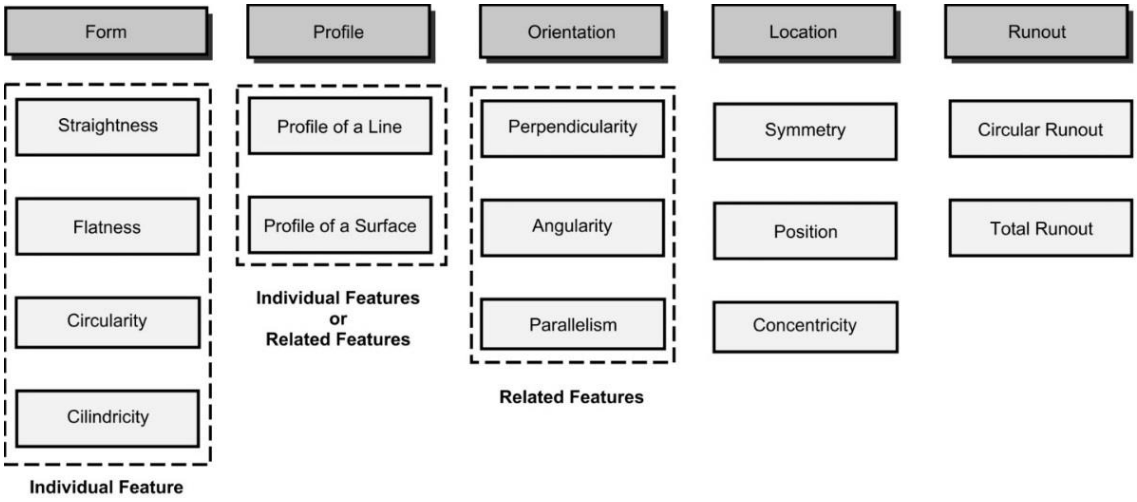


Figure 4: Geometric Tolerances Taxonomy (adapted from [30])

According to Juster [35], Kandikjan, Shah, and Davidson [36], the manufacturing industry employs two types of Geometric Dimensioning and Tolerancing (GD&T) approaches that are supported by the current standards: conventional tolerancing and geometric tolerancing (Figure 5). Conventional tolerancing represents the long-established practice of using plus-minus tolerances. In conventional dimensioning and tolerancing, tolerances applied to dimensions depict the allowed deviation of the shape in the direction of a given dimension. In contrast, geometric tolerancing provides a complete set of controls for every specific characteristic of the geometry (form, orientation, location, etc.) to the degree required to satisfy the function or interchangeability requirements of the mechanical part. For example, for specifications of geometric tolerances it is significant to provide material and theoretical dimensions. Theoretical dimensions exist between theoretical entities [37]. A theoretical entity is a datum or a resolved object of a feature. For example, the center of a circle is a theoretical entity because it is a virtual element that is not really there, that is, it only exists as a concept. However, the edge of a face is real as it represents and object of the physical environment. Theoretical dimensions can become the basis for the specification of

geometric tolerances [37]. Furthermore, because of its ability to manage a large amount of inter-related feature-based geometric variation, geometric tolerancing emerges as more suitable for representing the complexity of the construction domain [15]. As information technology becomes more powerful to manipulate large parametric models, the potential grows to build increasingly sophisticated functional systems for designing, modeling and fabricating buildings [38]. This dissertation focuses specifically on how the formal description of design requirements, manufacturing specifications, and a subsequent feature-based integration with CAD geometry can describe a more accurate tolerance specification. Geometric tolerancing is more closely related to the conceptual framework used in feature-based solid modelers. In this context, explicit interactions between the entities that control the geometry of a part (parametric model) can be joined to the geometric tolerancing specification through the SysML profile. This is in contrast to the conventional tolerancing (plus/minus) where the tolerance rule is applied locally but is not really relatable to the overall behavior of the part (from the material logic perspective) or the tolerance stack.

Since 1970, research in GD&T has been widely developed from several points of view such as geometry representation, variation of geometry representation, tolerances allocation, and manufacturing processes. One of the first steps to incorporate GD&T into CAD models was defining description languages for parts and assemblies [39]. Additionally, the development of Constructive Solid Geometry (CSG) of Requicha and Tilove [40], was also relevant to create geometric representations that are able to carry information of variation and tolerances. Later, Virtual Boundary Requirements (VBRs) and offsetting operations in solid models were successfully implemented by Srinivasan

and Jayaraman [41]. Furthermore, tolerance allocation for manufacturing processes has been also proposed For example, Zhang and Wang [42] have investigated the tolerances and variation that come from machine selection, which extends the scope for geometric variation to identify conflicting manufacturing interactions. In addition, Sodeberg [43] has focused on the association between critical dimensions and product life cycle to identify conflicting correlations among these categories. This design approach is demonstrated through a case study that considers the tolerance specifications for an automotive body panel, where specified tolerances influence a critical dimension that affects the product's assembled functionality. The potential for loss of functionality and the impact of selecting alternative manufacturing sequences on tolerances has been also covered by Fathi, Mittal, and Cline [44]. In the same regard, from the manufacturing prospective, Fraticelli, Lehtihet, and Cavalier [45] investigated the alternative processes definition. They described how tool wear influences the geometric variation of manufactured parts.

GD&T has also been studied from the optimization point of view through experimental design [46], and by means of Monte Carlo simulations [47]. GD&T research has also covered issues regarding quality loss under the restraints of process capability limits, functionality of design, and production quality requirements. These issues have been analyzed by using tolerance chart optimization procedures [48]. Tolerance allocation is another important topic covered by GD&T research. In this regard, a feature-based tolerance charting methodology was developed by Tseng and Terng [49], and Tseng and Kung [50]. This important work proposed a feature-based

tolerance charting methodology to automatically allocate the working dimensions and tolerances for 3D prismatic parts represented in boundary representation data.

Although various GD&T conceptual models are described in the literature, among scientists, classifying them is still a matter of discussion. For this research, GD&T models are classified using the Kandikjan, Shah, and Davidson [36] schema. This classification includes: documentation-oriented models, analysis-oriented models, production-oriented models, and control-oriented models. The documentation-oriented models aimed to include dimension and tolerancing information in CAD models and their documentation. In practice, these tolerances were introduced in the 2D drawings as notes only – and are not machine readable and thus rely on human transcription and interpretation The second GD&T model is the analysis-oriented model. This model is based on the concept of variational geometry and represents the tolerances through the variation of the position of some specific control points within the Euclidian space. The system-oriented tolerances model centers on creating the boundaries of the tolerance zone and conformance to tolerance. The production-oriented tolerances model employs graph-based tolerance representation for fabrication as well as machining setup and texture planning. Control-oriented models, based on graph representations, allows the specification of tolerances according to manufacturing standards.

Since the creation of the production-oriented modeling, most of the CAD/CAM systems that include tolerances follow the widely accepted ISO 10303 Standard for The Exchange of Product model data (STEP) to encapsulate tolerance data. STEP describes geometric tolerances information via EXPRESS language. [51]. This STEP implementation is important to assure quality throughout the process of transferring data

among different actors in the manufacturing industry. Today many manufacturing (but few construction) industries require process planners to generate manufacturing specifications based on functional requirements of the ISO standard [52].

The majority of the efforts presented above have been introduced by the mechanical engineering domain and implemented in the aerospace and automotive industries. However, they are rarely deployed in construction. Therefore, an important goal of this research is to adapt these accomplishments made in the mechanical world into a model for the construction domain.

3.2. Geometric Variation and GD&T Evolution

Since the 1980s, the representation of GD&T in aerospace and mechanical engineering have vastly improved. New devices to capture and assess geometric deviations, such as electronic scanners, and new GD&T computational approaches have allowed a robust interoperability among different CAD systems. During the early days of CAD implementation, texts and symbols were written into exchange files. A receiving system could display them on the screen or print them, but only a human could interpret them (Figure 5 left). This approach is called conventional tolerancing. The conventional tolerancing method is still used in many construction activities. Then, with the purpose of improving the readability of the tolerances information, variational data was introduced by means callouts referring to specific features of the model, for instance, a datum feature callout and a datum reference frame. This advanced approach was later known as GD&T. As it can be seen in Figure 5 (right), the advancement of human-readable tolerances representation included several new fields of information, which mostly refer to the feature-based context of the variational data. Besides numeric values and the variational

limits of the dimension, these new descriptions depicted tolerances types and datum frames to further define the expected variation. Also, the development of user-driven GD&T representation specified which element of the geometry of a product model has GD&T capabilities. For example, a system supporting GD&T representation may display the GD&T information in a tree or other dialog that allows the user to directly select and highlight specific features of the product in 2D and 3D. With the purpose of having better interoperability within a GD&T representation systems, the next level of evolution incorporated all the previous capabilities in an exchange file, for instance a STEP exchange. More specifically, a receiving system that allows a user to select a GD&T callout and view the corresponding feature highlighted on the shape of the product [53].

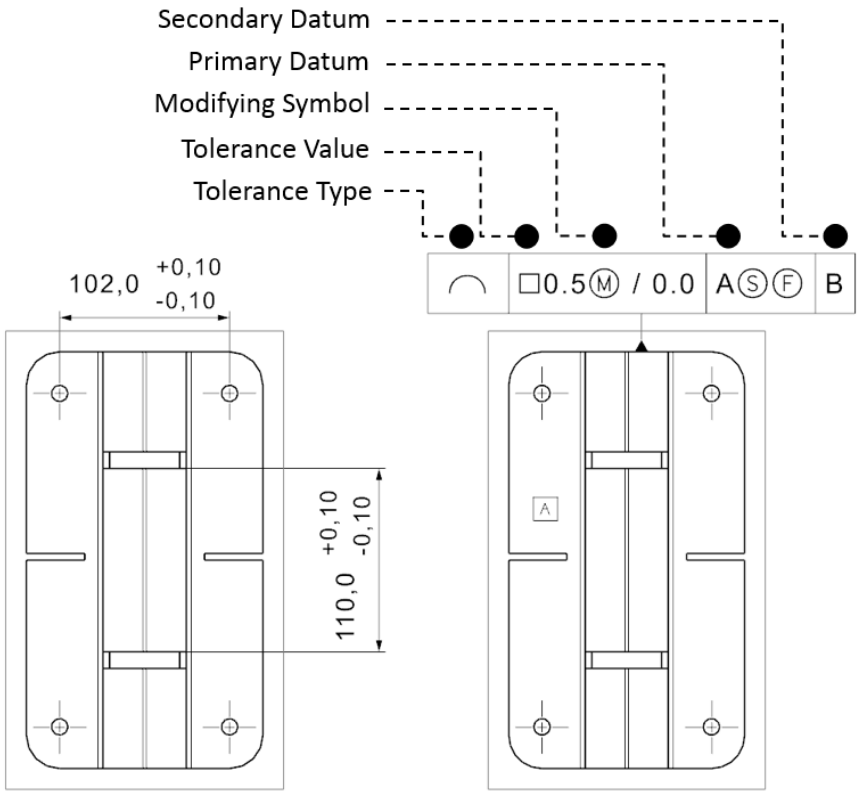


Figure 5. Conventional +/- Tolerancing and GD&T specification

Another critical improvement of the GD&T method was the development of a formal language to describe its functionality. This language, supported by the ISO organization, has built-in rules and restrictions for proper GD&T usage. This capability led to another important advancement in the representation of tolerances: the GD&T validation approach. Using the variational data as well as the GD&T representation and a supportive geometric format (e.g. boundary representation), it is possible to validate the completeness and consistency of the GD&T information. For instance, the newer approach classifies syntactic errors in a GD&T specification by converting ASME standards into grammar rules to check for ambiguities in datum referencing for a CAD model. Also, the ASME standard Y14.5 defines a set rule for GD&T to specify permissible variation in manufacturing [54].

Further development of the approach proposed in this dissertation will use geometric variation data at a system level to improve complex manufacturing and assembly processes, energy simulations, realistic visualization, and geometric assurance within the construction domain. In order to achieve these goals, the semantic layer created by the MBSE platform needs to be integrated with the current CAD environment to allow the representation of system-level manufacturing specifications interaction. The next section will present a review of the state of the art in MBSE, its integration with other modeling and simulation tools, and will further explain the importance of model consistency within MBSE to ensure successful integration in the construction realm.

3.3. Model Based System Engineering (MBSE), System Modeling Language (SysML), and Building Information Modeling (BIM)

MBSE is the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases [27]. MBSE tools and the associated visual architectural[8] modeling language, which has been established by the Object Management Group (OMG) based on the Unified Modeling Language (UML) is the System Modeling Language (SysML). It both extends UML to the domain of physical objects (UML is focused primarily on software and data) and limits UML by identifying the subset of UML that is useful for modeling objects and processes in the physical world. SysML is a general purpose modeling language for systems engineering applications and its scope goes through a wide range of systems, or systems of systems, including hardware, software, processes, and facilities [27]. Some of the general issues of using SysML for MBSE have been identified in [55] is that while SysML creators indicates that it is a "smaller, simpler" language for systems engineers, SysML suffers from language bloat because it adds two new diagrams (Requirements and Parametrics) and substantially increases the number of stereotypes with imprecise

[8] Here, the word “architectural” is used to describe system architecture, that is, how parts and assemblies relate to one another in terms of geometry, requirements, production process, supplier, etc. A synonym used for architectural used in this context might be organizational.

semantics. Also, another issue identified that has close relation with this dissertation is that Instance Specifications are ambiguously defined and poorly integrated with the rest of SysML [55]. An <<Instance Specification>> defines an occurrence (real-world examples) of a <<Block>> element. For this dissertation, instead of using <<Instance Specification>> elements to capture numeric values from CAD geometry, the approach uses simple <<Block>> elements (classes) as they are better integrated with the rest of the SysML language. This section of the document will introduce the motivations for the development of systems engineering; it will explain how systems engineering transitions from a document-centric approach to a model-centric methodology; and it will explain the development of System Modeling Language with its motivations and main components. A brief background about the state of the art in SysML model integration, requirements engineering, and model consistency will complete this chapter.

In current practices of architectural design, building engineering and construction, products and systems are expected to perform at predicted levels. As Friedenthal et al. [56] states: "Competitive pressures demand that these systems leverage technological advances to provide continuously increasing capability at reduced costs and within shorter delivery cycles." In the building industry, this increasing capability usually refers to a highly detailed set of functional requirements that challenge current modes of design, delivery, and operation of buildings. In order to successfully produce better buildings, the design and construction industry has integrated computational tools to shift away from the traditional approach of independent development of material systems and stakeholders requirements towards Building Information Modeling (BIM). BIM can be defined as a centralized modeling environment that allows connectivity of multiple

vectors, including project information, assembly specifications, building operation, and building users [57]. However, the development of BIM, although crucial at the geometry level, has not been equally successful in developing well-defined transactional construction process models to eliminate data interoperability issues [58].

A building, as any other complex system, is not a static entity. Rather, it changes over time as sub-systems or other building components are incorporated or detached during the building lifecycle. These changes result in requirements and behaviors of constituent systems that may not have been anticipated when the system was developed [56]. Furthermore, in building design, multi-functional components are highly common. For example, a building roof covers and encloses the space of a building; it protects the inner space from weather events such as rain and snow. it adds thermal protection to the interior; it enables the installation of other systems such as windows or solar panels. Any of these functions has to comply with a very precise set of functional, structural, aesthetic, and economical constraints during the building lifecycle. If no proper knowledge and project data integration platform is implemented, presumably any change of the roof design, meant to improve one aspect, will result in the detriment or at least some change of some other functionality. As one proposed solution to this larger problem, the systems engineering approach, through its modeling language SysML, has been extensively recognized in the aerospace and mechanical engineering industry to provide system solutions to technologically challenging and mission-critical problems [56] [27] [59]. The next section of the dissertation will explain how systems engineering is applied to develop a model-centric approach in engineering, and how this approach can be used as a platform for dealing with the data heterogeneity in the building domain.

3.3.1. From a Document-centric approach to a Model-centric approach:

One of the important contributions of MBSE has been the development of model-based architectures that have enhanced the ability to share and exchange project data. This approach, although significant, requires improved knowledge and skills of users to facilitate the adoption of model-based practices. This need has led to the increasing significance of the system architect as a managing entity for the integrated platform. In the following section, we will contrast the distinctions of the document-based approach and the model-based approach for systems engineering applications. In the AEC world, the skill of modelers has been challenged by the implementation of BIM, which is inherently 3D and requires a higher level of modeling skill. Most BIM authoring tools require that modelers assert the relationships between building objects as part of building BIM models, which is an additional challenge, but which makes the building model richer and more useful. The system model includes everything in BIM and adds sub-models for requirements and processes. Thus the modeling complexity is increased even further, leading to the identification of the "systems architect" as a managing entity.

Even with the development of BIM and system engineering, the current practice of architectural design and construction still relies on the conventional document-centric approach to deliver and manage building lifecycle data. This method usually emphasizes the generation of individual design documents, in hard copy or electronic file format with restrictive interoperable capabilities, which are exchanged among the project stakeholders.

If systems models are deployed using a document-based approach, the following modeling objects are generated to assert the relationship between documents [27]:

concept of operations (ConOps), requirements specifications, requirement traceability and verification matrices (RTVMs), interface definition documents (IDDs), N2 charts (also known as N-squared charts—matrices of structural interfaces), architecture description documents (ADDs), system design specifications, test case specifications, and specialty engineering analyses (e.g., analyses of reliability, availability, schedulability, throughput, response time).

Considering that in this increasingly complex IT environment [38] a building project creates endless amounts of project data from different people and tools, the document-centric approach requires a significant amount of time to ensure that documentation is valid, complete and consistent. The classic document-centric approach specifications are depicted in specifications trees. Then, a systems engineering management plan (SEMP) defines how the systems engineering procedure fits in the project, and how all the concurring disciplines come together to develop the documentation necessary to satisfy the requirements in the specification tree [56]. In the document-based approach, functional decomposition is executed to explain how functional requirements are to be fulfilled by the components of the system or building. Usually, these kind of relationships will be depicted in design documentation such as flow diagrams. However, flow diagrams of a document-centric approach lack interoperable functionality. In addition, requirements management is performed to parse requirements of the design specifications with design embodiments, to capture those requirements in requirements databases, and to trace requirements by identifying the systems or sub-systems that the specifications are referring to [27]. Current requirements

management tools have capabilities to verify requirements satisfaction and to reflect the traceability in the requirements database.

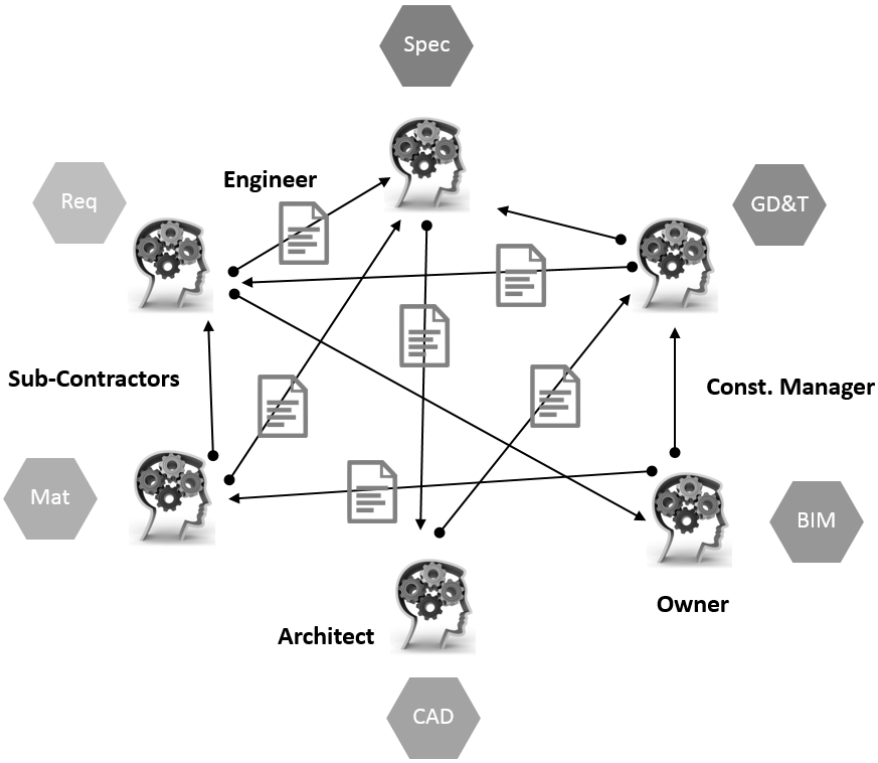


Figure 6: Document-Centric Approach

To summarize: though a document-centric approach may be quite rigorous, it has a critical limitation when assessing the consistency and completeness of project data. For this approach to be successful, the systems architecture must be clear and the stakeholder in charge of the document mapping must be consistent and constant in order to maintain a complete systems model. As [56] points out, The comprehensiveness, consistency, and relationships between requirements, design, engineering analysis, and test data are hard to evaluate due to the fact that information is spread across several documents. Understanding a particular view of the system and executing the necessary traceability and design-change impact assessments is clearly challenging. Applying this scenario to the AEC domain may lead to a deficient coordination of design requirements, which

could subsequently lead to poor knowledge integration regarding material systems and manufacturing processes, and finally to quality issues when the final product is delivered.

As it has been described above, the document-centric approach for systems engineering –although having many advantages, suffers from an important disadvantage: model inconsistencies. This situation was one of the main motivations for the development of the MBSE approach With the MBSE approach, many of the intermediate deliverables of the modeling activities seen in the document-centric approach can be generated automatically. However, as [27] explains, in the model-centric approach, the main product of those activities is an integrated, coherent, and consistent system model, produced using a dedicated systems modeling tool: the System Modeling Language (SysML). All other artifacts are secondary—automatically generated from the system model using the same modeling tool.

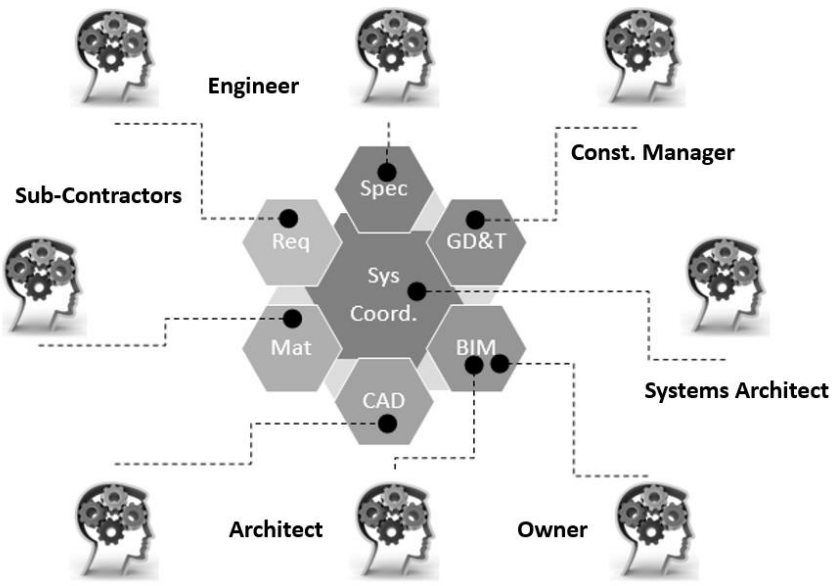


Figure 7: Model-Centric Approach

One of the important characteristics of a comprehensive model is that it enables stakeholders to take informed decisions. Decisions made within an MBSE framework

take place within a central repository, where each design decision is captured by a model element or a relationship among model elements. With the model-centric method, all diagrams and self-generated text objects are simply views of the underlying system model, they are not the model itself. And that difference is the core of the return on investment (ROI) that MBSE offers over the document-centric approach [27]. In the system model, as all modeling elements are programmatically and systemically integrated, any change that is produced will be automatically propagated to the rest of the model. This capability is possible because of the programmatic characterization of underlying dependencies of the model elements. It does not matter if the elements are depicted in a diagram that is user-defined or automatically created, or if the model is too large or complex. After all, the diagrams of the system models are just views of the real model, which keeps its internal consistency based on its seamlessly integrated approach.

3.3.2. Diagrams of the System Modeling Language (SysML)

This section introduces the different diagrams available in current the version of SysML as it is explained by the Object Management Group (OMG, http://www.omgsysml.org/). Also, an introduction for the use of SysML diagrams in the building domain is provided in a study of CAD-SysML integration for natural ventilation assessment [60]. This project starts by modeling generic natural-ventilated buildings in SysML, and examines the model through some scenarios produced from parametric geometric iterations. The aim of those parametric iterations is to model the natural ventilation system to survey different options of a building in a short period of time in early stages of design. The building is a five-story, open plan, office building located in Atlanta with stack-assisted cross ventilation.

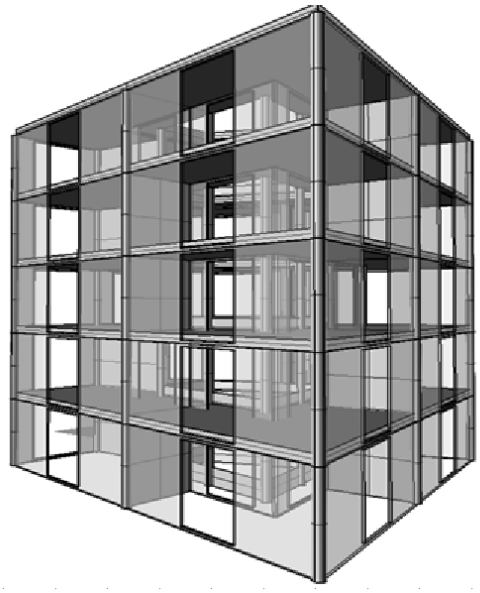


Figure 8: Building instance used as case study for the natural ventilation example

One of the main goals of the project is to visualize the geometric impact of decisions taken in the building energy performance analysis domain. The size of rooms and windows, height of stories, and building orientation will influence the results of ventilation performance and also affect the appearance of the building. For the geometric design of this development a parametric identification of the building was created in Grasshopper, a parametric modeling tool that works within the Rhinoceros 3D modeler environment. This Grasshopper definition contains all the basic elements of the office building: basic structure, floors, exterior walls, interior walls, openings, roofs. The Grasshopper definition also contains all the topological relationships among the elements.

Figure 9 shows the hierarchy of SysML diagrams. Then, the rest of the section provides examples of instances of these diagrams offered through the natural ventilation example [60].

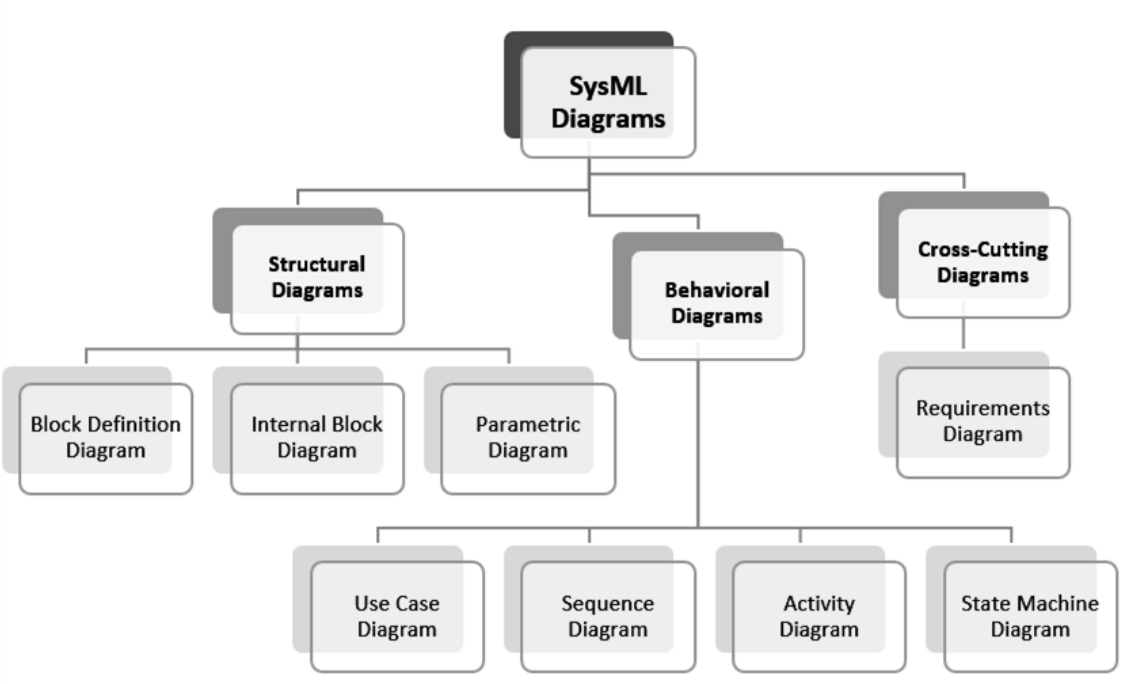


Figure 9: Hierarchy of SysML diagrams

SysML has an intuitive interface of multi-functional diagrams where the <<block>> is the basic unit of a structure. Every system structure can be represented by block definition diagrams (*bdd*) and internal block diagrams (*ibd*) [24] [25]. A block definition diagram defines the system hierarchy and system/component taxonomies and the internal block diagram describes the inner structure of a system in terms of its parts, ports, and associations – in other words, the *bdd* describes how assemblies and parts are related and nested semantically and the *ibd* depicts how the properties of elements (already defined in *bdd*s) are related [55]. The *bdd* is the most common of the SysML modeling elements and is intended to depict the structure of a system. For example, if the diameter of a hole in a plate and the diameter of a bolt are related, then this relationship is declared in an *ibd*, but the definition of the plate and the bolt take place in a *bdd* [55]. One of the characteristics of the bdd is the level of granularity or detail that users can obtain, depending on the target stakeholder for whom the diagram is intended. *Bdd* has to

be created based on the level of detail that is needed in the creation of subsequent system modeling diagrams. For example, if an activity diagram refers to the drilling and subsequent measuring of a hole in a part, then the *bdd* must define the part, and the hole that is in the part, and must provide the dimensions of the part and the hole so that these can be referenced in the activity diagram. The model elements that are displayed on *bdd*s—blocks, actors, value types, constraint blocks, flow specifications, and interfaces—work as stereotypes for the other model elements shown on the other kinds of SysML diagrams. These elements that appear on *bdd*s are known as elements of definition. These elements of definition are the foundation for everything else in a system model [27]. The main elements of a <<block>>, which are shown in Figure 10, are the parts and values. The parts represent subcomponents that are typed as "children" of the <<block>> and the values represent model parameters that are depicted in the model to drive mathematical analyses and simulations. There are three central kinds of relationships that can be created between blocks: associations, generalizations, and dependencies. In Figure 10, blocks are connected by using a "black diamond" association. This means the blocks are physically connected to the parent block by using a "has a" relationship. If a block was part of the parent block but did not physically connect to it, this association would be of the reference kind and it would be represented by an open diamond, which indicates a simple aggregation.

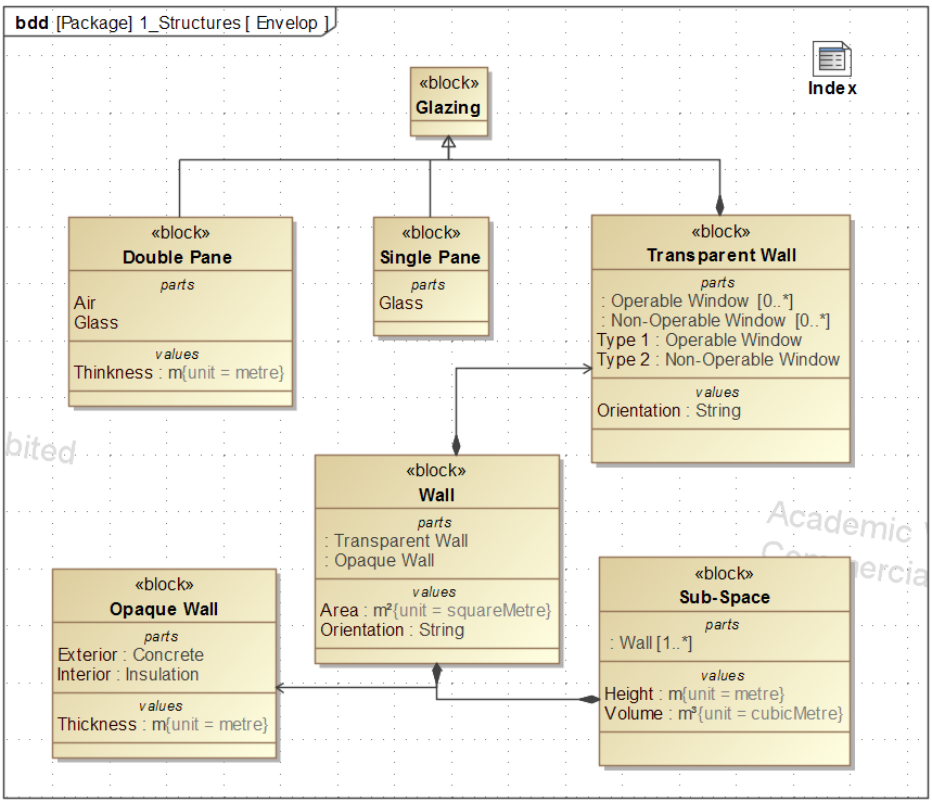


Figure 10: Block Definition Diagram (bdd) showing the structural decomposition of a building assembly, Valdes, Sun (2012)

The internal block diagram (*ibd*) depicts the internal view of a system block, and is usually instantiated from the block definition diagram, to represent the integration of all blocks within the main system block [24]. As an example, the *bdd* in Figure 11 represents a room network in a building story. The story contains four rooms and a central common space for all the stories depicted as “stack.” It is important to note that, even though the bdd contains only one block called “Sub-Space”, it is actually representing four different rooms because of its four composite associations to the parent block “Floor 1”.

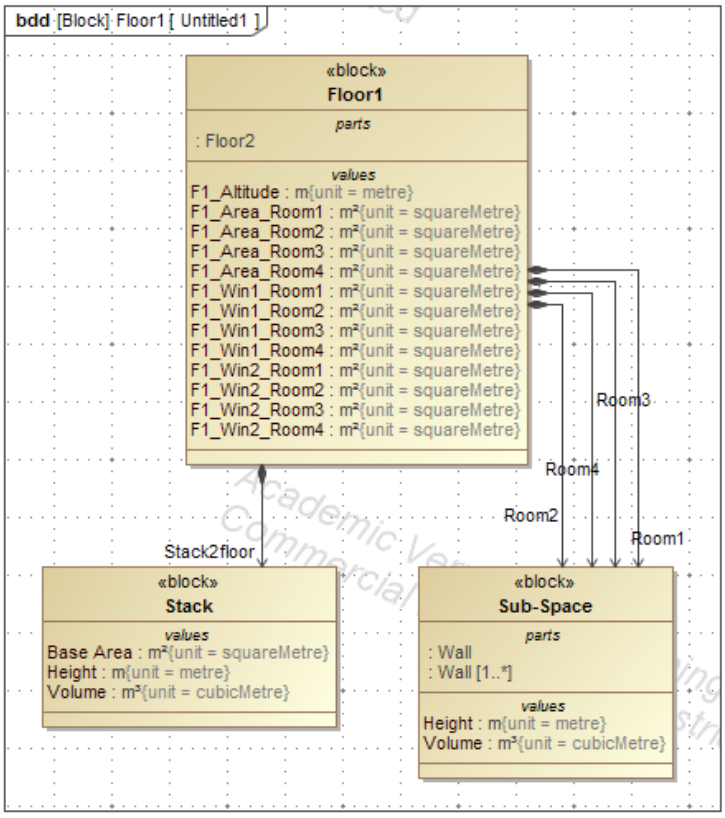


Figure 11: bdd of the space distribution of a building story with 4 rooms and a central common stack, Valdes, Sun (2012)

Figure 12 represents the same building story shown in the *bdd*, but it is characterized as an *ibd*. In this *ibd*, the internal structure of the air transference network is depicted using the same elements of the *bdd*. The main difference is that in the *ibd* the associations are showing an item flow instead of a hierarchical relationship among the parts. In an *ibd*, item flows are required to match the ports that they are binding together. It is important to note that all item flows and ports of the example in Figure 12 are typed as "air" as they represent parts of a natural ventilation model. It should also be noted that item flows between two ports are required to specify the direction of the flow.

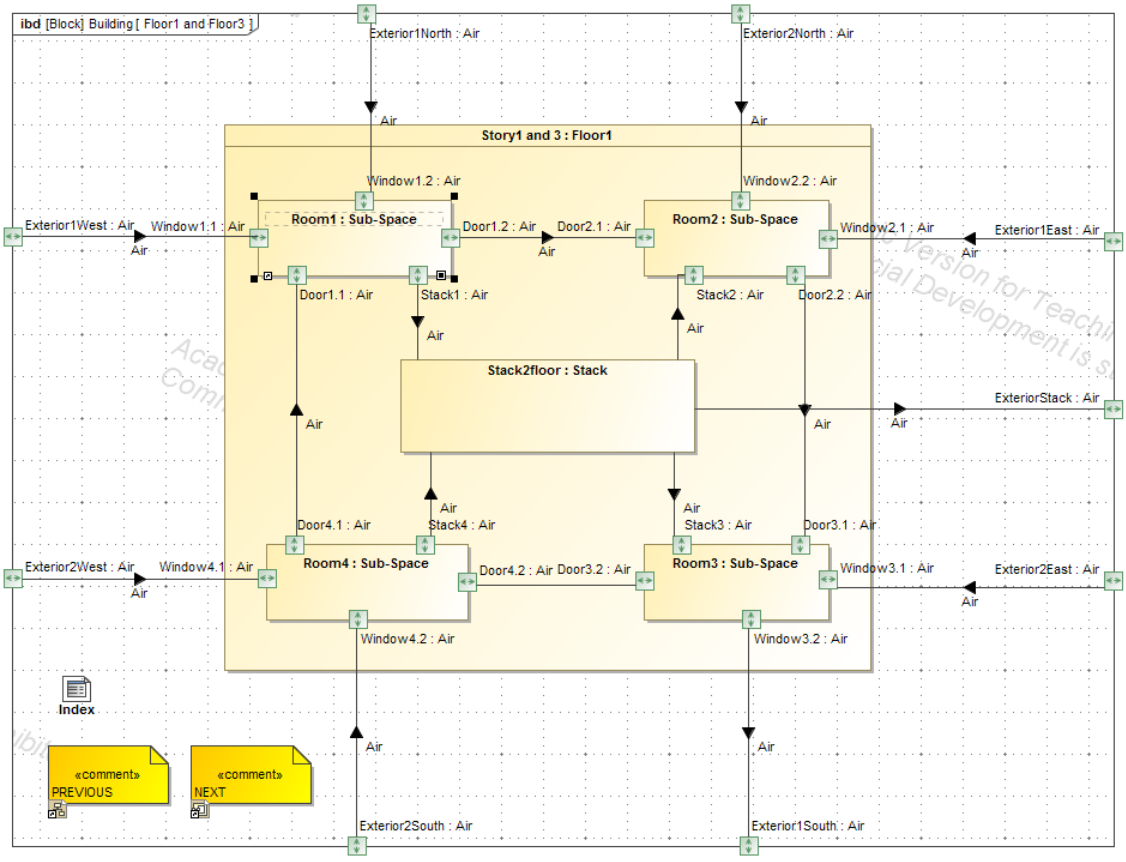


Figure 12: ibd of the same building story shown as bdd in the previous picture, Valdes, Sun (2012)

In order to integrate specifications and design models with engineering analysis models, parametric diagrams (*par*) represent constraints on attribute values which can be derived from material, performance, and reliability properties. As can be seen in Figure 13, a parametric diagram contains four basic elements: an instance block that represents the occurrence of a <<block>> element, and contains numerical values to perform mathematical or logical calculations; a constraint block that contains a mathematical or logical expression to be calculated during the parametric execution; a port that defines the type (e.g. real) of the specific value of the element; and a binding connector that links <<block>> data with the inputs of the expression in the constraint block through their port elements. Parametric diagrams enable a value property that might be deeply nested

in a containing hierarchy to be referenced at the outer containing level [25]. Also, parametric diagrams explicitly show the item exchange and the interdependencies between parameters and attribute values that drive the different components of a system. This facilitates the identification of sources of performance and the composition of a system with good performance. For example, the Aspect System defined by Augenbroe [61] represents a subset of a building model that is important from a functional perspective. This functional view is achieved by functional decomposition, as in *ibd*s, and needs to be agreed upon with all design stakeholders. Then, the Aspect System will be formulated as a measurable expression of performance.

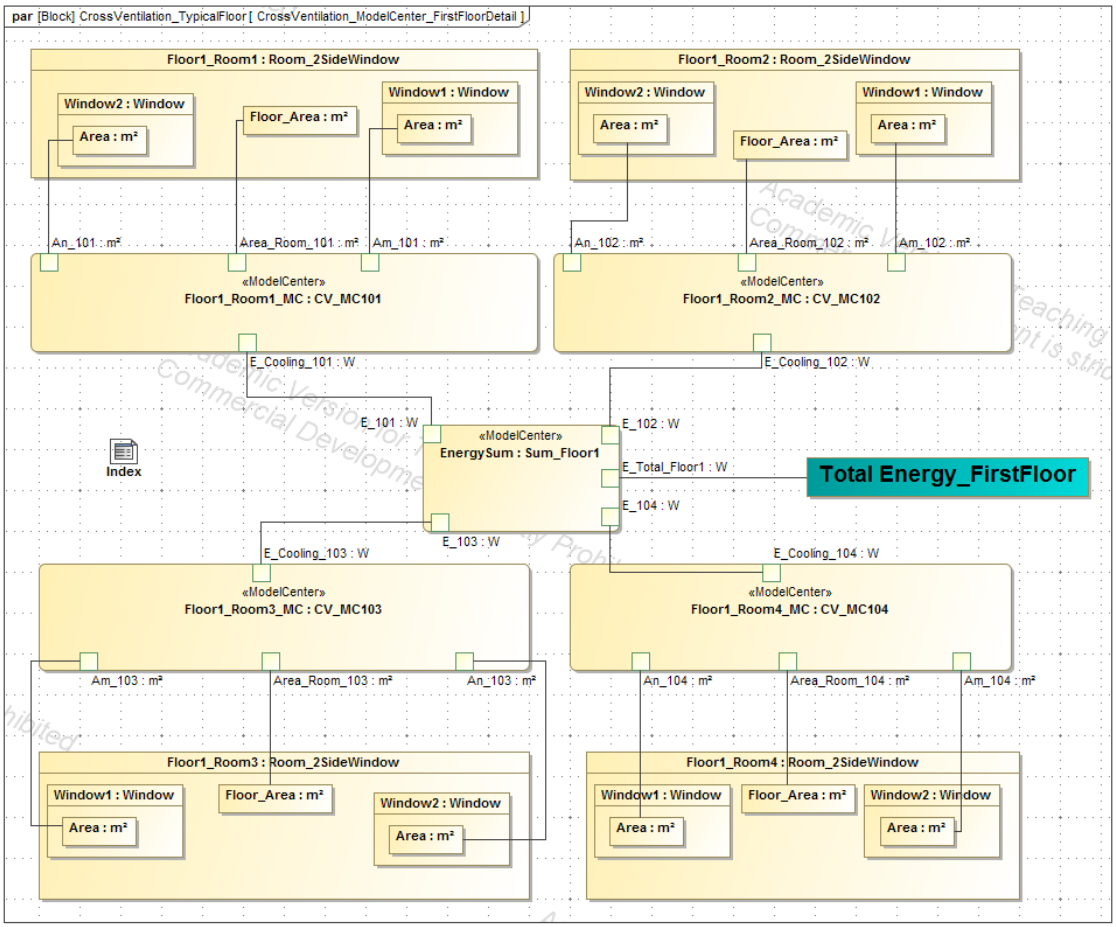


Figure 13: Parametric Diagram (par) example, Valdes, Sun (2012)

Behavior diagrams comprise the use case diagram (*uc*) and activity diagram (*act*) shown in Figure 14 and the sequence diagram (*seq*) and state machine diagram (*sm*) shown in Figure 15. A use case diagram provides a high-level description of functionality that is achieved through interaction among systems or system parts. The activity diagram denotes the flow of data and control among activities. A sequence diagram represents the interaction between collaborating parts of a system. The state machine diagram describes the state transitions and actions that a system or its parts perform in response to events. Activity diagrams can represent specific construction processes by means of the description of process phases and associated metrics. All these metrics can be traced step-by-step through sequence and state machine diagrams, and can be compared to formal specification of design standards through requirements diagrams (Figure 16).

The activity diagram in Figure 14 illustrates space cooling through natural ventilation. The control logic is when indoor air temperature is higher than the indoor set point temperature of 22 degrees Celsius and outdoor air has cooling potential (i.e. outdoor air is cooler than 22 C), the natural cooling system will control window opening areas. After the window area is adjusted, wind pressure drives outdoor air flow through open windows. Finally, the status of fresh air and air temperature in the room will be updated. This activity needs to repeat periodically, so the indoor environment will be monitored.

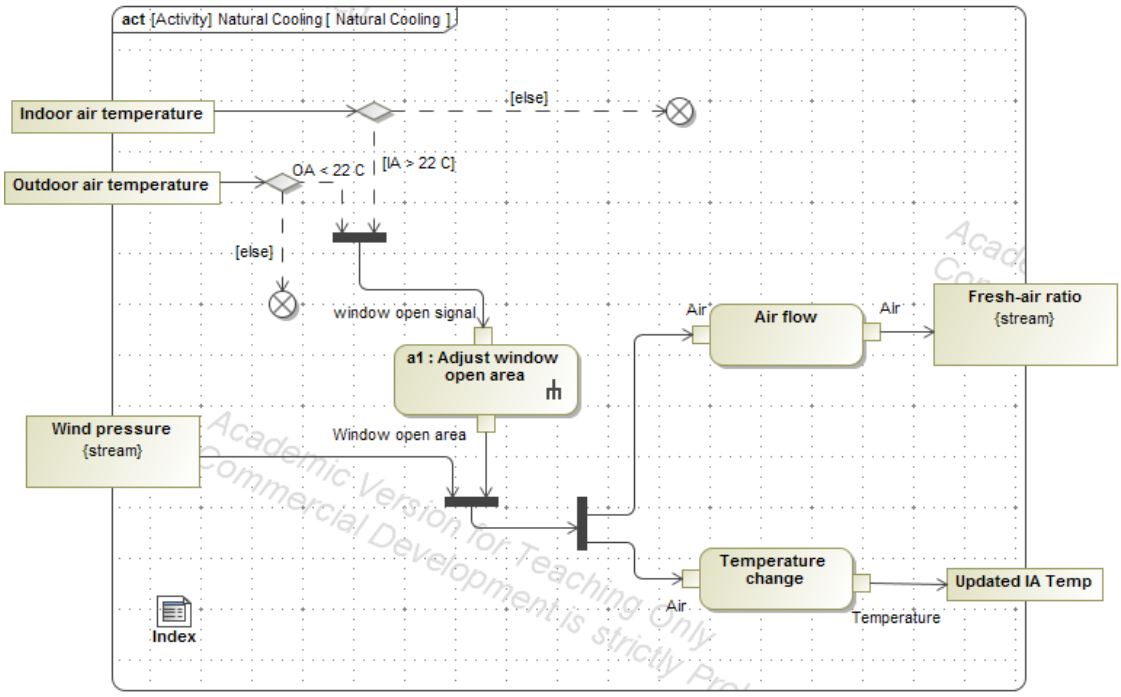


Figure 14: Activity Diagram, SysML, Valdes, Sun (2012)

The state machine diagrams represent several states that an object may be in and the transitions between behaviors and states. Actually, as it is understood in other modeling languages, it is common for this type of diagram to be named a state-transition diagram or a state diagram. A state characterizes a phase in the behavior of an element, and as in SysML activity diagrams, they will have initial states and final states [62]. The following example in Figure 15 represents a state machine diagram of a room used in the natural ventilation project. The sm diagram comprehends all elements that are activated when the room is being occupied. State machine diagrams are clear examples of behavioral modeling in SysML and its applicability in building simulations.

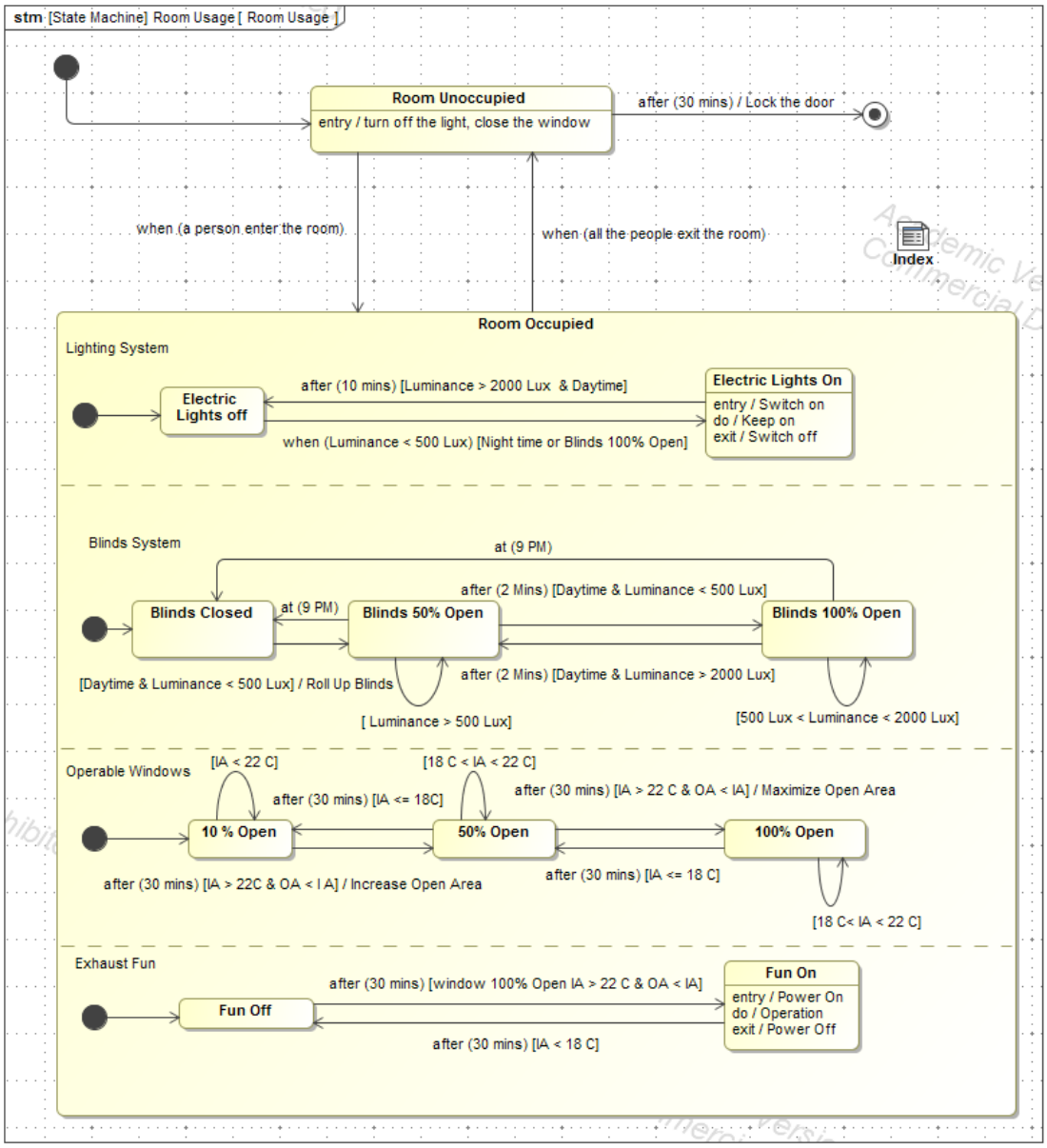


Figure 15: State Machine Diagram, SysML, Valdes, Sun (2012)

SysML contains a graphical methodology to represent text-based requirements and relate them to other model elements as critical dimensions obtained from the CAD model. The requirements diagram captures requirements hierarchies and requirements derivation, which then satisfy and verify those relationships. The requirement diagram will associate manufacturing knowledge, included in construction standards and material

systems know-how, with the model element that satisfies or verifies the requirements of the system model.

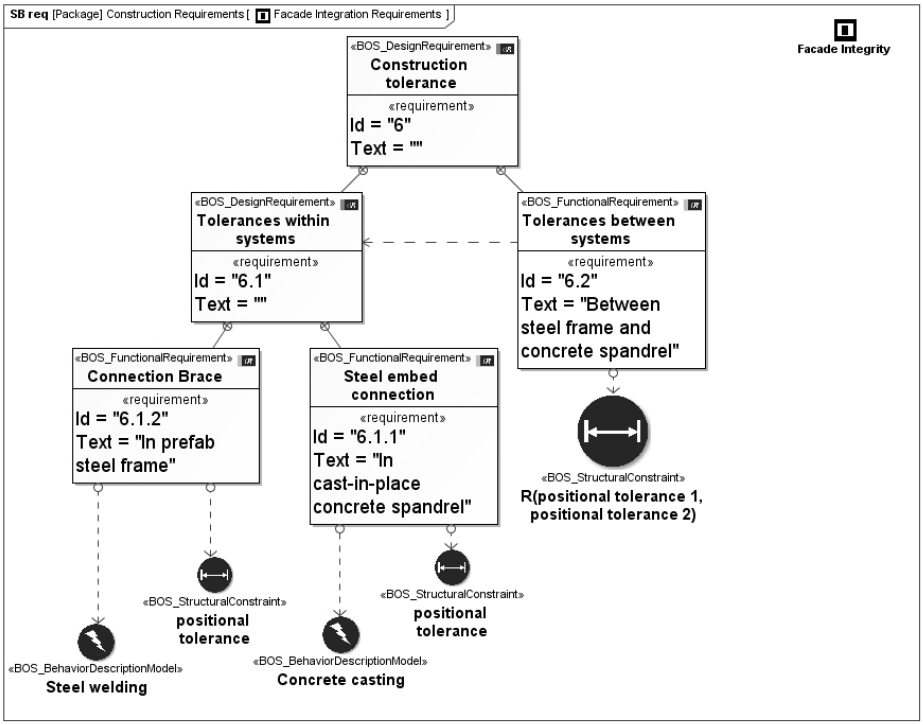


Figure 16: Requirements Diagram, SysML, Valdes, Cavieres DBL Symposium: GaTech (2013)

One of the most important functions of MBSE is to allow a formalized body of knowledge to support rule-based design, hence relieving designers of the monotonous activities affecting the engineering design process [63] [64]. One central aspect of this approach is the generation and management of complex product configurations that provide data to several discipline-specific tools involving geometry inputs or geometry manipulation. Knowledge-based geometry and tolerances modeling and allocation are examples of such configurations. Also, all the sub-processes and analysis that include geometry manipulation are highly interconnected and rely on each other to advance the product through its lifecycle. For example, most of the geometric design parameters of a building product depend on non-geometric rules and requirements modeling, and must be

verified against construction standards. However, the geometric data necessary to elaborate systems models within SysML still relies on manually-entered and updated procedures. This situation leads to several drawbacks of model integration such as invalid data, lack of consistency among models, and excessive design review procedures. In order to implement a SysML method in a highly geometry-based field like BIM, a proper artifact to automate BIM-SysML data translation in a product development lifecycle is required. With this enhancement, BIM could potentially ensure data consistency among models, leading to an increased building quality and a reduction in time and cost. However, as it is found in the construction realm, the multi-disciplinary nature of BIM results in vast amounts of project data, managed in different tools, corresponding to different domains but which can be coordinated through SysML. Although SysML has been successfully applied to several areas of the engineering design, such as implementation for analysis (CAA) and communication and collaboration with several stakeholders and external applications, the integration of SysML with geometry intensive platforms as BIM is still an ongoing research area.

3.3.3. Requirements Management in Systems Engineering

Software development is highly dependent upon the Requirements Engineering domain. Furthermore, within the requirements engineering process, the elicitation of customer's requirements is a crucial stage. Saiedian et al. [65] focuses directly on the factors that shape requirements elicitation. In this regard, the elicitation stage addresses requests and needs of consumers and presents a solution for their specific system. Additionally, Chituc [66] takes Requirements Engineering research into another context: long term digital preservation. There is limited research on Requirements Engineering

along the lines of long-term digital preservation. Chituc introduces the challenges and advantages that could be present if digital preservation research was done in depth. In relation to Requirements Engineering research, elicitation is also a common phase in the context of digital preservation, but it takes the form of questions, surveys, and interviews. However, due to the lack of integrated tools for gathering information about the consumer's and practitioner's needs, the elicitation process usually results in poor communication, resistance, and lack of perspective. Through an industrial study, Sikora et al. [67] propose methods to understand practitioners' needs concerning Requirements Engineering research and development. This study included qualitative interviews as well as qualitative data collected via questionnaires that reported five aspects of Requirements Engineering approaches. The interview and questionnaire results concluded that the use of natural language was prevalent in all industries, but many of the stakeholders felt that including models would be a more comprehensive approach. In that regard, graphical representations and prototypes have been found to be a technique that could reduce the amount of ambiguity between the consumer and the modeled system. Graphical representations extend from blueprints to hierarchies of problems, while prototypes enhance the understanding of problems to identify solutions. In addition, Dos Santos Soares et al. [68] offers new ideas about user requirements, mainly for software-intensive systems, focusing on diagrammatically documenting them. System Modeling Language (SysML), diagrams and tables, could effectively represent most of the meta-requirements for complex system. SysML requirements diagram display requirement relationships, while SysML tables show traceability and decomposition for software-intensive systems.

Holt et al. [69] debates how the application of Model-based Systems Engineering is becoming well understood at the systems level, yet there is a lack of research and subsequent application at the system of systems (SoS) level. This research proposes a Model-based Systems Engineering approach called COMPASS (Comprehensive Modeling for Advanced Systems of Systems), for requirements engineering, that could be applied to both the system of systems (SoS) level and its constituent system (CS) level. The four basic types of SoS requirements presented in this research include virtual, collaborative, acknowledged, and directed, which are essential in the application of MBSE. Winkler and Pilgrim focus on the current traceability research and practice in requirements engineering and model-driven development (MDD) to bring stakeholders together by identifying commonalities and differences in the two areas and finding unsolved challenges that affect both. Traceability is found in requirements engineering and MDD, because it is important in the verification and validation process. Also in traceability, Cuddleback et al. [70] investigated what factors influence a human analyst's performance when vetting a candidate requirements traceability matrix (RTM). RTM is a mapping process between elements of one artifact to another and is one of the most revered processes in construction. Since RTM is highly revered, there may have been some bias among the analysts in the study from a golden standard already established. The study found that the analysts move their RTMs towards the line that represents recall precision, meaning RTMs with low recall and low precision were improved drastically.

In a study by Ingmar [71], the main focus is discussing how a system is built with operational descriptions of the missions the system is to complete through a central model. A central model is essentially a means for developers to eliminate problems with

concurrency and incrementalism, basically as a control for engineering. Once the central model is established, the model is extended into a Common Project Model. The Common Model Project is the breakdown of a system's structure and behavior, shown in a way that manages problems.

Mancin [72] explains how to implement Model-Based Systems Engineering (MBSE) as systems become more complex in their design. Through the use of UML/SysML as independent modeling language, support analysis, design, development, verification, and validation phases, an executable model will be created for proper implementation. SysML is the main focus of his article, more specifically the analysis stems from the implementation of the SysML language and the use of the SysML diagram.

Capilla et al. [73] consolidated the findings of three other articles, creating a summary solution to research challenges as it relates to requirements and architecture. The first challenge was developing a strategy that allows requirements to transition to architecture models. A Strategy for Transition between Requirements and Architectural Models for Adaptive Systems (STREAM-A) was proposed to solve this challenge. This approach uses goal models based on the i-star framework to design and evolve systems. This system allows software engineers to perform the smooth transition from requirements to architecture models. Transitioning is not the only problem for systems, quality performance is also a concern when architecting with quality requirements. The proposed model to resolve this is called QUality PERformance (QUPER), which uses qualitative reasoning to make estimating quality targets easier and reasonable for quality requirements. QUPER has already been applied in the industry, so further application will

increase upgradability, performance requirements, and improve the decision making process. Finally, improving and understanding software quality requirements will continue to evolve and will always present a challenge. To limit the number of challenges in the industry, web applications (WebApp) can be used to indicate internal/external usability problems [73]. Usability problems will be assessed through the WebApp, which will provide recommendations for improvement. With diversity on the rise in technology, it becomes more complex to balance relevant quality attributes and support different levels of quality.

Goal-oriented requirements have received an increasing amount of attention because they are used to elicit, elaborate, structure, specify, analyze, negotiate, document, and modify requirements. Essentially, Van Lamsweerde [74] presents the various efforts, arguments in favor of goal orientation, and a case study that shows how the goal-oriented method works. Specifically, the Knowledge Acquisition in Automated Specification (KAOS) method is advanced, because the four sub-models it creates can be applied to any size project. The four sub-models essentially assist in analysis in developing the goals and their application. Identifying the consumer's goals in the early stages of the Requirements Engineering process is a benefit of the KAOS method. Goals allow for the consumer to see explicitly what they what the system to do. Goal-oriented Requirements Engineering is specific to goals, verifying the requirements to ensure the goals are identified and satisfied by the requirements. Goals are essential in goal-oriented requirements, hence the name, especially because of their ability to support goal modeling via qualitative or formal reasoning.

Sutcliffe [75], defines the role of scenarios in Requirements Engineering and the Scenario Requirements Analysis Method (SCRAM) that allows for proper prototyping of scenarios. Scenarios have been found to be helpful in counteracting human reasoning by testing hypothesis and assumptions in models. In relation to models, scenarios complement them by including all the goals of the stakeholder and making them clearer to show how the system might work. Ultimately, scenarios present real world applications leading to prototyping for models. Along with modeling requirements through scenarios, the SCRAM method is one of the most successful techniques when applied. This method consists of four phases, initial requirements capture and domain familiarity, storyboarding and design visioning, requirements exploration, and prototyping and requirements validation, which allows for the safe guiding process in organizing the requirements analysis. The requirements analysis is essential so that all the needs are included in the scenarios for the consumer to see what the system does explicitly. In the end scenario-based requirements engineering has provided numerous avenues to fulfill requirements for consumers.

3.3.4. System Modeling Language Integration Background

Model integration between SysML and other domain-specific applications has been a matter of long development. These integrations have dealt with several programming languages and data structures. However, the integration with CAD data structures has not been successful so far. This section introduces the most relevant integration project of SysML and other design and simulation tools.

One of the first approaches to create model integration within MBSE was GeneralStore of Reichmann [64]. This approach proposed a common execution language

to deal with models already developed. GeneralStore has been considered as a limited approach because it does not work during changing design stages. Hooman et al. [76] introduced a co-simulation approach to exchange information between models during runtime stages.

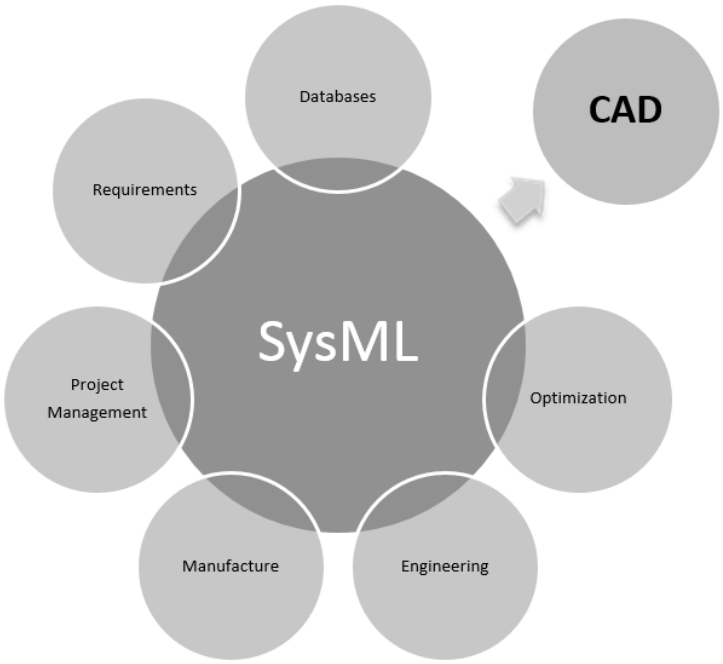


Figure 17: SysML Integration Status

Tolk [77] presented research that surveyed several issues about meta-modeling and mapping among different modeling languages. Also, Vanderperren and Dehaene [78], developed an integration between Matlab and UML. In addition, Brisolara et al. [79] also developed an approach to integrate SysML with Simulink. In 2007, Pop et al. [80] developed a SysML profile to integrate SysML with Modelica. Also in 2007, Nytsch-Geusen [81] introduced a profile to graphically describe Modelica models in SysML. Johnson et al. [59] [82] also worked on a SysML Modelica integration. Brucker

and Doser [83] developed a Meta-model-Based UML Notations for Domain-Specific Languages to explain how to create domain-specific formal semantics.

Huang et al. [84] developed an approach to apply simulation activities within SysML by means of mapping simulation models from Tecnomatrix plant simulation. Van der Velden et al. [85] introduced an adaptable methodology for automation application development. In 2008, Jobe et al. [86] proposed multi-domain integration in SysML through a Multi-Aspect component model called MAsCOM. Giese et al. [87] introduced an approach to produce low-level models in an automatic manner. This approach formally described the meta-models to automate the process. In 2010 Shah et al. [88] developed an approach to create multi-view modeling by means of SysML profiles and model transformations. Schamai et al. [89] similarly introduced integration approaches between SysML and Modelica by means of ModelicaML. Several other efforts have tried to improve the integration of SysML and Modelica. For example, OMG developed an approach to standardize a SysML – Modelica integration [90]. Furthermore, in the Modelica environment, efforts to generate code from abstract models have been produced by Dassault Systems (2011) and OpenModelica Consortium (2011). Additional integration methods either provide execution capability for executing SysML models, such as ParaMagic (InterCAX, 2011), which aids in executing SysML parametric diagrams based on composable objects [26], or they focus on integrating SysML with other modeling and simulation languages. Marchenko el al. [91] developed a new method of visualization and documentation of parametric information of 3D CAD models. In 2012, Rocca [92] presented research to explain the concept of Knowledge Based Engineering (KBE) through the integration of AI and CAD. Also in 2012, Mosier [93]

presented an overview of the NASA-integrated model-centric architecture. Valdes and Sun [60] developed an approach to parametrically assess natural ventilation performance in early stages of building design. This approach integrates metrics from a parametric model into a SysML model to perform critical analysis with Model Center. Figure 18 shows the integration environment of this approach.

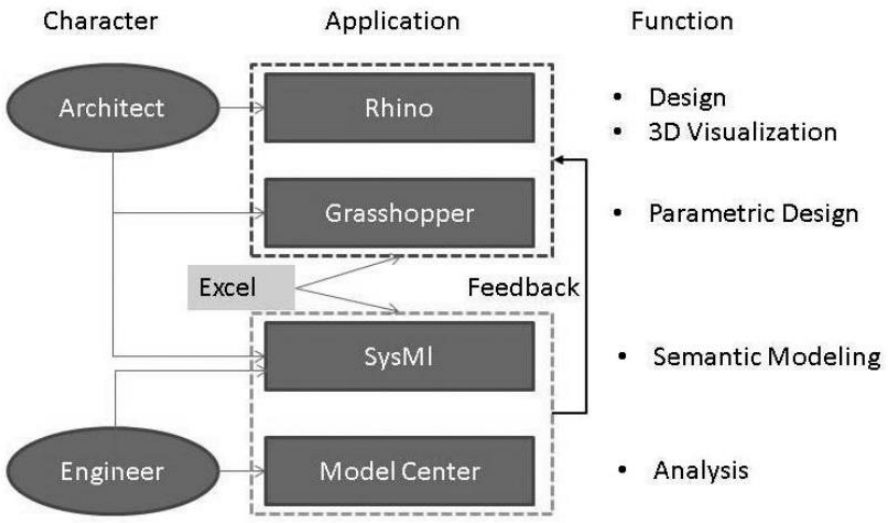


Figure 18: Integration Environment of NatVent Project: Valdes: Sun: (2012)

3.3.5. Consistency Management in MBSE Background

The MBSE realm integrates numerous embedded systems and applications that need to be properly coordinated and consistent. As we have seen, every embedded system will carry domain specific data that will be represented through domain-specific languages (DSL). In this multi-disciplinary environment, two consistency categories arise: internal consistency and external consistency. The first category deals with consistency within a model itself and the second deals with inconsistencies among multi-domains models created through different modeling languages [94]. As this project deals with the integration of CAD models and SysML models, the second category is our main interest. There have been more than a few projects related to consistency and consistency

checking in the MBSE literature. In 2007, Adourian et al. [95] proposed a methodology to check consistency between geometric and dynamic views of a mechanical system. Hehenberg at al. [96] developed an approach to analyze consistency issues in mechatronic design models. In 2009, Gausemeier et al. [97] developed a project of management of cross-domain model consistency during the development of advanced mechatronic systems. The UML language has been identified as an important ground for consistency issues by several authors. Chanda et al. (2010) developed a framework for semantic verification of UML diagrams, Simmonds et al. [98] proposed a method to maintain consistency between UML models using description logic, and Mens et al. [99] worked on another framework for managing consistency in evolving UML models. Also, the OMG has developed approaches to deal with semantics in UML.

CHAPTER 4: Tolerances in Building Construction

4.1. The Challenge of Modeling Construction Tolerances

Design errors and omissions, such as failure to predict and control geometric variability in construction, have enormous effects on cost and efficiency of today's building industry [32] [16]. Hence, in order to produce high quality, cost-effective buildings; planning and execution of processes of construction must consistently consider the option of geometric deviations during the design stages. However, to achieve this goal there are numerous challenges that have introduced in this dissertation:

- Multiple material systems with different bodies of manufacturing knowledge [2];
- Geometry does not necessarily comply with manufacturability while being designed and later updates to remedy inconsistencies will increase the likelihood of mismatches with other components [15];
- Lack of knowledge representation and allocation methods for each material system [57] [2];
- Lack of integrated manufacturing knowledge traceability from specifications to geometry [52];
- Lack of manufacturing and tolerances verification methods [52]; and
- Lack of consistency across different tools and models [94].

This dissertation, will consider five stages of the construction process: early stage; design validation; construction preparation, which include detail design of fabricated components; building erection and production of off-site components; and building management. Every stage of the construction process also contains sub-stages that are located horizontally on the graph in Figure 19. All the sub-stages, or tasks, are connected

with two basic kinds of dependency elements: single-direction arrows and double-direction arrows. While the first indicates that information will flow from task A to task B, the double-direction arrow indicates a mutual dependency between two tasks. Also, when a single arrow goes against the time line, this will mean that the specific dependency is part of a multiple-task loop. In order to introduce the level of influence of geometric variation at different sub-stages of the construction process, a simple color code is offered in the diagram. This color code introduces the following logic: the darker the color, the higher the geometric variation influence of the task. This research does not consider the demolition stage because its relationship with geometric variation has not been considered relevant.

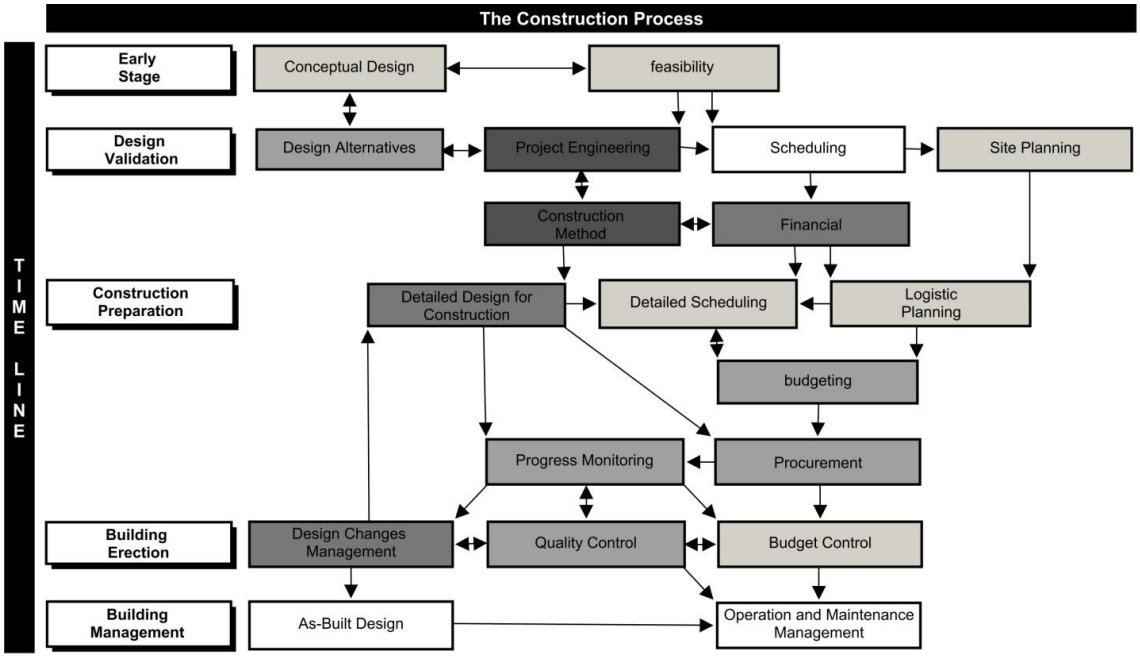


Figure 19: The Construction Process (author)

Besides understanding the multi-party nature of the construction process with regard to stakeholders and software environments, it is important to recognize how the geometric complexity of building assemblies leads to construction inaccuracies. A

building is a highly complex assembly that often has millions of different components. All components are interconnected in relationships that depend not just on levels of accuracy of the manufacturing and assembly strategies, but also behavioral considerations during building operation. Furthermore, every material system has its own set of manufacturing rules, behaviors, processes, and standards, which, when put together with other materials systems, creates unpredicted interactions that may reduce the expected performance of the building. This issue leads to new and increased demands on the ability to break down manufacturing and tolerances requirements to subparts and subsystems, and to be able to sum up the expected variation from subparts to a system level.

In addition, planning and production methods of construction have changed over time. The foundation for such changes has been the virtual product development through computational modeling and simulation using BIM tools [23] [57]. This advance enables scenarios of reliable analysis and complex calculations of the entire life cycle of the building. As a result of this computational development, many physical mock-ups have vanished and are now replaced by digital simulations [100]. To take real advantage of this technology, deviations produced during construction practice must be adequately represented in BIM models. So far, however, BIM tools do not provide enough computational support to consider all the geometric variation and tolerances of construction [57]. Rather, today's processes of construction are entirely driven by nominal CAD models and geometry with unassessed manufacturability, without considering the multi-level interaction of building components and processes. To address

this issue, it is essential to understand how different levels of interaction of building components affect the likelihood of a nominal outcome.

The following section will assess this inquiry by creating a construction tolerances taxonomy, which will be the basis for a further representation of different types of construction variability during the implementation stage.

4.2. Towards a Construction Tolerances Taxonomy

In building construction, causes of inaccuracies can be traced from a design and manufacturing perspective or directly from materials, parts, and assemblies during construction. This project proposes four categories that are accountable for geometric construction inaccuracies: materials knowledge, geometry knowledge, assemblies' knowledge, and processes knowledge. The first category, materials knowledge, considers physical, chemical, and mechanical properties and deformations based on material properties or surface roughness. For instance, in construction, there is an important phenomenon called hysteresis that is a permanent altering of an object's physical properties due to a certain repeated external influence over time. The most common causes of hysteresis are the influence of gravity, thermal expansion or contraction, and changes due to moisture exposure. When thermal influence occurs, deformation may be magnified by the fact that two assembly components experience different temperature gradients. Besides the effect in the appearance, this can negatively affect the material's strength. The second category, geometry knowledge, considers geometric variation levels of a single building component that belongs to a specific material system (for instance, a brick, a precast beam, or a sheet metal component). This category is probably the most important for the development of this dissertation. Often, single components do not fit

during aggregation because their defining geometry has not been evaluated by specific rules of manufacturability for the precise material system. Although designing with "non-nominal" or "as-built" geometry is not the aim of this dissertation, prior to manufacturing, the intended nominal geometry must be as close as possible to an ideal instantiation of the intended form and function. Furthermore, this geometry must be evaluated by using the manufacturing constraints that comes from the project requirements and specifications. In this category, geometric variability is to be addressed by defining a set of "critical dimensions" usually revised during quality control procedures, before building erection. The third category, assemblies' knowledge, includes assembly sequences, number of parts for assembly, prefabricated assemblies versus on-site assemblies, and automated assembly versus manual assembly. Considering that assembly procedures are often produced substantially by human labor on-site, this category is highly dependent on accumulated geometric variation known as tolerances stack. Tolerances stack is critical in construction and will be covered in the implementation section of this document. The last category, processes knowledge, considers values of machines and tools, process capabilities of the selected fabricators and contractors, skill levels of the human labor teams, and the percentage of on-site construction that the project will include. Although anticipating every aspect of geometric deviation is almost an endless task (for example, simulating the texture of a concrete brick), a proper approximation of model construction inaccuracies requires understanding the composite nature of buildings. This includes geometric variability of a single material system (parts, components) and geometric variability of a heterogeneous material system (assemblies). The following section will address this matter and will offer a simple

tolerances taxonomy by dividing the sources of inaccuracies between single domain construction tolerances and heterogeneous construction tolerances.

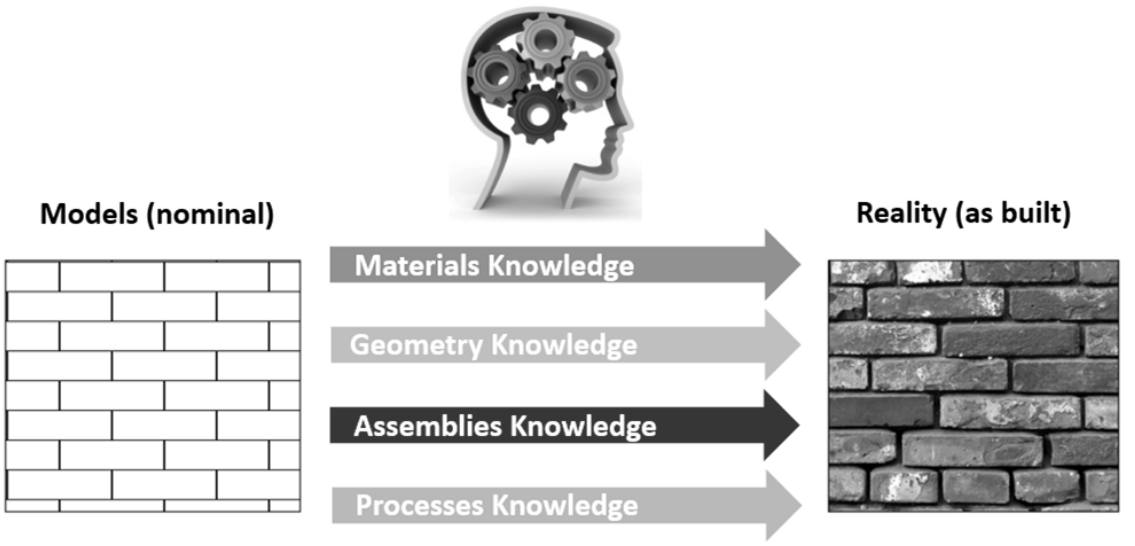


Figure 20: Representation of the different bodies of knowledge that define the accuracy of an assembly in building design

4.2.1. Single Domain Construction Tolerances (SDCT) and Off-site Sub-Assemblies

The construction industry contains numerous material systems that can be divided into two main categories: a distinctive material together with its associated manufacturing processes or at a certain stage during the building life cycle. An example of the first kind of sub-construction domain is the "structural steel domain." As its name implies, the structural steel domain comprehends all the processes of manufacturing and assembly that are related with this specific material. On the other hand, a material system stage is the group of processes related to a single or multi-material assembly that belongs to a specific moment of the building life cycle. For example, finishes are a sub construction domain that fits in this category. Both kinds of sub-domains will be associated to the Single Domain Construction Tolerances type (SDCT). One of the main

issues of the SDCT group is that they have their own internal tolerances, which are addressed separately, usually within different manufacturing-specific workflows and subcontractors. As can be seen in Figure 20, frequently building products that belong to a SDCT will have a very specific set of manufacturing rules and tolerances standards. That is, concerns such as the composition of the material or if the component is made on-site or in a factory will affect the expected deviation from nominal. Differences in expected variability are also increased when dissimilar material systems come together in some assembly condition that overlaps some of their three-dimensional features. As an example, in construction it is very common to have assemblies that combine on-site concrete casting with off-site manufactured steel structure. While a steel assembly could target geometric deviations within 1/32 of an inch [2], the cast-in-place component will be probably around one quarter to even one half inch of variability, according to SDCT rules. If we also add the natural geometric complexity of building products and their behavior, and the number of SDCT-SDCT interactions, the addition of all those variabilities will produce significant sources of building inaccuracies. All in all, variability assessment in SDCT will frequently fall within specification. The real problem of SDCT is in their aggregation. Assemblies of dissimilar material systems have led to the creation of a new variability category that will be introduced in the next section: Heterogeneous Construction Tolerances (HCT). Considering this and the increasing interest of the BIM community in pre-fabrication, this research focuses on variability interactions of multi-SDCT assemblies that combine knowledge-dissimilar off-site components (or sub-assemblies). The following sections will give a brief description of

the most important material systems used in current construction and their most typical sources of variability.

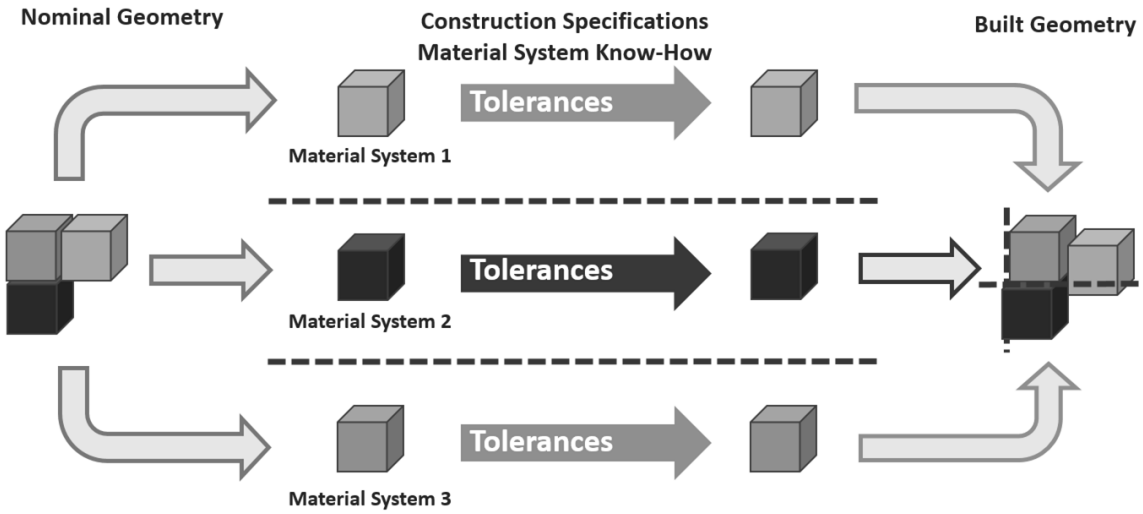


Figure 21: Tolerances incompatibility among different SDCT systems

Building Layout

The first SDCT of the building life cycle is the building layout. This SDCT mainly comprehends the variational location of the building within a construction site and the regulations about general paving and right-of-way laws. The most important geometric deviations that can be accommodated by using tolerances are right angle layouts of sides and site. A suitable approach to measure deviations of the building layout is to create a tridimensional survey grid from which all the tolerances are allocated. For the vertical layout, a critical dimension to consider is the accuracy of plumbness that can be represented as a percentage of the length. Building layout allowances are described in several guides and handbooks like: Handbook of Construction Tolerances [29], NIST handbook [101], Construction Science Research Foundation (1989); and ISO 2263-1, Measurement Methods for Buildings, (1989).

Concrete

Concrete SDCT contain numerous sub-sections related to cast-in-place concrete and precast concrete. A sub section of precast concrete describes all the pre- and post-stressed details of this SDCT. Important variation aspects that have to be prescribed include levelness of concrete and asphalt paving and variations in the slope and thickness of the sections. Another important tolerances aspect of concrete is geometric variation produced by inaccuracies of reinforcement placement in walls and columns, precast panels and beams, precast insulated panels, and reinforcement placement of prestressing steel. These kinds of inaccuracies, as in most of the material systems, are frequently caused by design errors due to the lack of material-specific knowledge during design stages. In addition, elevated and on-grade slabs will produce two types of variation. First there will be a tolerance of elevation and second a tolerance of flatness and levelness. Cast in place needs to be toleranced especially in its plumb and also in its sectional variations due to deformations of the framework. In addition, special attention is required for changes in height and right of way construction details. Some sources of variability that are difficult to include in GD&T are related to inaccuracies in the concrete mix preparation. The principle sources of describing allowances of concrete SDCT are the American Concrete Institute [102] [2], the American Society for Testing Materials [103].

Structural Steel

The steel SDCT contains mill tolerances for different steel shapes and numerous allowances for connections. Mill manufacturing tolerances should define values for camber and sweep in S and M shapes profiles, and the same for structural angles and tees. Special attention is required to specify tolerances of architecturally exposed structural steel, location of connections and welding threads, and elevator shaft tolerances. Besides

tolerances allocation, Steel structures must consider proper clearances to allow connections. The main sources of guides and allowances are suggested by the American Institute of the Steel Construction [104] and the American Society for Testing Materials [105] (2003, 2004, and 2005).

Unit Masonry

The unit masonry SDCT describes allowances of brick manufacturing, reinforcement, and assembly tolerances shows the diverse levels of variability from an inherited structure. Although there are some differences among construction material systems, this model applies to most of them. In the case of masonry, the first level of inaccuracies is the masonry unit itself. Differences in the sizes and surface evenness are the most significant issues. In this material system, the most important source of variation that needs to be addressed by tolerances allocation is the unit placement. Here, the thickness of the mortar and the overall plumbness of the wall are critical. In the graph, arrows indicate variation inheritance among levels that describes a summed tolerance or tolerances stack. Other issues related to unit masonry construction are relative alignment of rows, bearing wall level alignment and changes in the height due to variation of the mortar layer among rows. Also, prefabricated masonry panels need to allocate tolerances for out-of-square and out-of-plane recurrent issues. Most of the standards for unit masonry have been created by the American Society for Testing Materials (ASTM) [29]

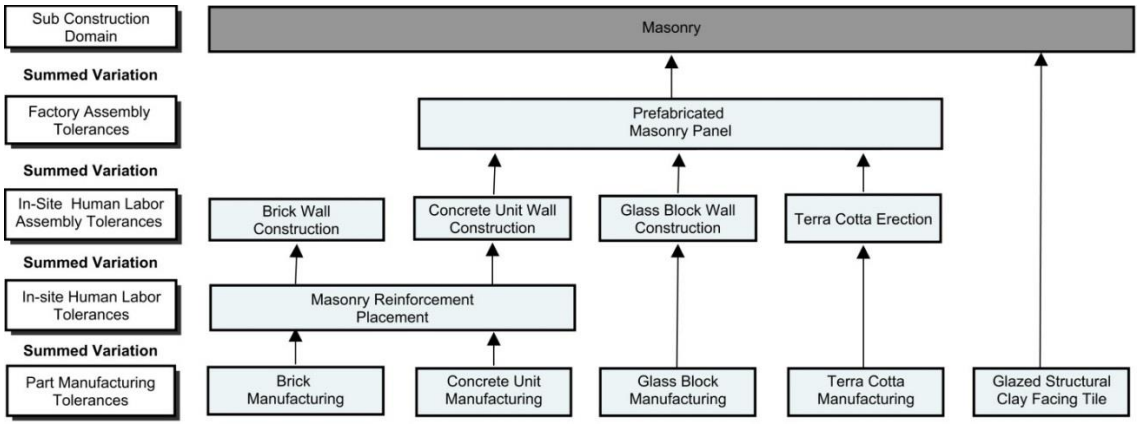


Figure 22: Example of SDCT Masonry with its levels of tolerances

Stone

The stone SCDT contains standards for different kind of stone construction as granite, marble, and limestone. The most important deviations related to this material that are easily described through tolerances allocation are the assurance of thickness and squareness of every block. The second most important issue that is very recurrent in cladding is the lack of flatness due to variations in material temperature. During granite and marble installation, relative alignment and plumbing must be toleranced to create adequate joints. Stone SDCTs have been proposed by the Marble Institute of America [106], and the Indiana Limestone Handbook [107].

Structural lumber

The structural lumber SDCT covers topics such as glued laminated timber fabrication and plywood in general, fiber board manufacturing, rough lumber framing, and wood floors. In laminated members we have to define dimensional tolerances, camber or straightness tolerances, squareness tolerances, rough lumber, and end trimming tolerances. In addition, in plywood, fiberboard and particleboard manufacturing, we find variational sources that describe size tolerances, squareness and straightness tolerances, and thickness tolerances. For structural timber that is assembled mainly through human

labor, special awareness must be considered for bows and twists of rough lumber framing and wood floor framing and subflooring. The main organizations that have defined standards for allowances of inaccuracies in this category are the American Institute of Timber Construction [108], the American National Standard Institute [109], and the National Institute of Building Science [110]

Finish Carpentry and Architectural Woodwork

The finish carpentry and architectural woodwork SDCT, defined in the Tolerances Handbook of Ballast [86] is also presented as a sub construction domain. This section contains several site-built wooden applications as well as frames, jambs, and window variational allowances. Manufacturing tolerances must be applied independently for rough lumber and dressed board lumber. This section is very sensitive to site-built cabinets, countertops, and stairs and trim. Tolerances described for joints that do not produce gaps are critical. Special awareness must be considered for continuity of doors and window frames, where allocation of clearances and joints are not allowed, and which require mitered joints. The codes and allowances for this category are basically the same as prescribed for the structural lumber SDCT with some specific additions as the Kitchen Cabinet Manufacturers (ANSI/KCMA) standards [111].

Curtain Walls

The curtain wall SDCT is also suggested for this tolerances category. The aluminum curtain wall fabrication and installation are the main sub sections of this segment. The PVC curtain wall standard has to be included as well. Both materials have similar sources of variation of their glassing framing. The main issues found in the curtain wall assemblies are height and width tolerances, maximum alignment of vertical

members, and control of diagonals of the glazing framing to ensure squareness of the structure. Also, curtain walls installation is very sensitive to the exterior alignment of different stories of the building and to the clearances given for embedded windows and doors. The main sources of this standard are the American Architectural Manufacturers Associations [112], and the ANSI Dimensional Tolerances for Aluminum Mill Products [113].

Finishes

The finishes SDCT, for being a "stage" sub-construction domain is quite diverse in its specifications. The scope goes from framing for gypsum wallboard, wallboard partitions, and acoustical ceiling, to stone and wood flooring and rods and bars. Being a very thin material, the main sources of inaccuracies of the light-gauge framing, which produce several other inaccuracies with other material installations, are plumbness and straightness. Also, for floor and wall tiles, wedging and thickness variation is critical. In this kind of material, the proper allocation of joint tolerances will allow a better finish. The same rules apply for the specification of wood flooring, which also generates variation due to moisture content changes. The principal sources of finishes allowances are ANSI [114], and ASTM [115].

Doors, Windows, and Glassing

The final SCDT is doors, windows, and glassing. This SCDT shares basically all the sources of geometric variation with curtain walls and finishes. The allowances of this SCDT comprise all the frame work tolerances for windows and doors, as well as all the standards of insulation of glassing. The sources of these standards come from several

guides including ASTM, ANSI, the Windows and Door Manufacturing Association WDMA [116], and the Steel Door Institute [117].

In general, every process of building construction will generate deviations from nominal geometry. These deviations are statistically studied, and annotated in construction standards, or informal know-how documents through minimum, maximum, and average values. These construction standards are guides where observations of past experiences about geometric deviations are consolidated and also where a cushion to allocate that variation is prescribed as a tolerance. As a synthesis, every expected, unintentional, geometrical deviation from nominal values that is estimated in advance should be prescribed as construction tolerances. However, the current methods for tolerances modeling still relies on off-feature, table-based allocation procedures. To overcome this old-fashioned approach, and considering that the problem is not the lack of manufacturing knowledge but its applicability, this dissertation aims for an integrated modeling framework where features and knowledge can programmatically coexist. This section has presented the main issues about geometric variation and tolerances for each of the most relevant construction sub-domains. The next section will discuss the knowledge and materials aggregation of multiple SDCTs that led to the development of heterogeneous construction tolerances.

4.2.2. Heterogeneous Construction Tolerances (HCT).

The previous section provided a summary of geometric deviations that apply to single material systems and construction components. Allowances regarding tolerances of these material systems are included in standards or informal know-how guidelines that will be used in this project as base knowledge for a software demonstration of an

integrated modeling environment. These single materials usually belong to a sub-group of domain tolerances and geometric variations. However, at some part of the building erection, assemblies will be built by merging different materials systems with off-site components (SDCT) and processes. These kind of heterogeneous assemblies produce geometric deviations due to the addition of materials with different mechanical properties or due to the addition of components fabricated by different subcontractors with different workflows, processes, and standards. These types of material-boundary geometric deviations must be represented by Heterogeneous Construction Tolerances (HCT). A central characteristic of HCT assemblies is that they can be easily field-adjusted [29]. This field adjustment process may well produce deviations in other parts of the building and in the tolerances stack. This situation might involve the specification of looser tolerances or clearances, but the excessive prescription of variability will also increase uncertainty in the assembly. A better method may be to allocate tolerances at a system level that, by means of simulations based on manufacturing knowledge, are capable of coordinating several SDCT at once. Another critical aspect of an HCT assembly, besides tolerances allocation and field adjustment, is the specification of clearances. In the Tolerances Handbook [29] a clearance is defined as the space between two components that is provided to allocate tolerances and movements. However, in HCT specifications, clearances are also required to allow human labor (for example, tightening a bolt in a curtain wall assembly). In a building, usually clearance offsets define the boundaries of a material system assembly and therefore define the interface between assemblies of different building material systems.

Besides clearances, the physical artifact of interfaces between material system assemblies in the construction environment is the joint. A joint is the material connection between elements of an assembly or sub-assembly. In addition to their functions of building continuity inside a building assembly and adding a clearance offset to allow deformations, the significance of a joint comes from its capability to make construction irregularities less noticeable [29]. Real assemblies need to create their connection using an interface that negotiates between edges or faces of different parts. There are numerous approaches to create joints among parts in a building assembly. Some of them will create a structural assembly and some will create continuity among layers (for example, the layers of the building skin). Many architects consider only movement due to thermal expansion and contraction, if they size joints at all. However, there are several other factors that influence correct sizing and placement of joints. Any change of plane or materials requires a joint. Wind loading affects joint placement not only for structural glazing applications but also for parapet walls. Moisture-related movement of materials also plays a part, concrete shrinks as it dries, brick grows as it absorbs water, and wood alternately shrinks and swells. Differential thermal movement between adjacent materials systems must also be accommodated with joints. All these aspects of joint design, if checked one at the time, do not assure a successful outcome because they depend on each other. Rather, specification of joints must be addressed in a coordinated fashion by integrating them in a system level. As it has been explained in section 3.3.2, the SysML environment enables the formal representation of these kinds of building behaviors by means of activity diagrams and state machine diagrams. Then, these diagrams can use linked CAD data to evaluate how behavior affects geometry, in context.

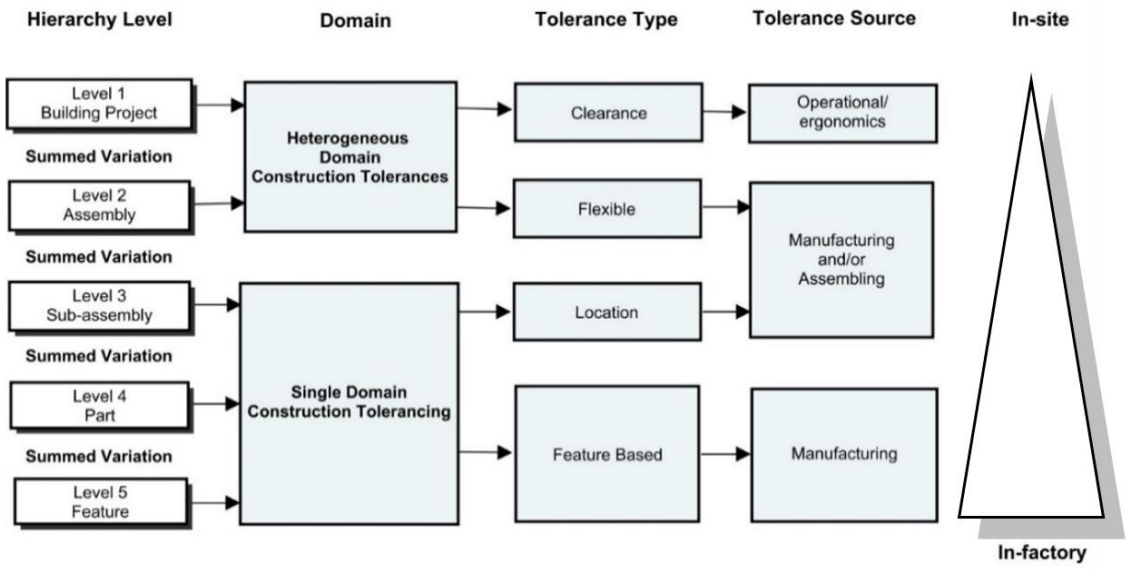


Figure 23: Hierarchical diagram of construction tolerances

Figure 23 offers a hierarchy of the different kinds of geometric tolerances within the construction industry. In the first column, from bottom to top, the chart presents several levels of geometric deviations that are nested during the construction process, starting from the minimal feature-based deviation and adding variation until the building project level, these levels are connected to their domains as suggested in this research, which are SDCT and HCT, respectively. Accordingly, features, parts, and sub-assemblies are mainly described as SDCT and assembly and building project level are described by HCT. Each SDCT and HCT defines different tolerances types. SDCT primarily generates positional tolerances and feature based tolerances. HCT generates flexible joints and ergonomic clearances. Furthermore, these tolerance types are associated with specific geometric variations sources or necessities. In this regard, the sources of feature based tolerances type is associated to manufacturing deviations, positional and flexible tolerances are associated to manufacturing or assembly deviations, and clearances are associated to operational and accessibility conditions**.**

CHAPTER 5: Knowledge and Tolerances Representation in Construction

5.1. Current Approach for Drawings and Specifications

Design drawings and specifications are based upon consideration of the design, assembly, and loads and forces to be resisted by the all materials involved in the building project. These design drawings and specifications clearly show the work that is to be performed and give the following information with sufficient dimensions to accurately convey the quantity and nature of components to be fabricated and assembled [118]. This list, created by the American Institute of Steel Construction (AISC), offers a comprehensive approach that can be used for any material system in construction. The list of items includes:

- The size, section, material specification, and location of all members
- All geometry and working points necessary for layout and assembly
- Floor elevations, top views, context drawings
- Column centers and offsets
- The camber requirements for structural and pre-stressed members
- Tolerances for each member and assembly [118]

Design drawings and specifications include any special requirements for the fabrication and erection of the all the components. Specially, structural design drawings, specifications, and addenda have to be numbered and dated for the purposes of further identification [118]. One important issue in construction is that contract documentation usually differs in complexity and completeness. Nonetheless, the fabricator and the

constructor must be capable of relying on the precision and completeness of the contract documents. This allows the fabricator and the constructor to provide the owner with bids that are adequate and complete. It also enables the preparation of the shop and erection drawings, the ordering of materials and the timely fabrication and erection of shipping pieces.

In some cases, the owner can benefit when reasonable latitude is allowed in the contract documents for alternatives that can reduce cost without compromising quality. However, critical requirements that are necessary to protect the owner's interest, that affect the integrity of the structure or that are necessary for the fabricator and the erector to proceed with their work must be included in the contract documents [118]. Some examples of critical information include:

- Standard specifications and codes that govern design and construction, including bolting and welding
- Material specifications
- Special material requirements to be reported
- Welded-joint configuration
- Special requirements for work of other material systems
- Connections or data for Connection selection and/or completion
- Restrictions on Connection types
- Openings for other trades
- Surface preparation and shop painting requirements
- Shop and field inspection requirements
- Non-destructive testing requirements, including acceptance criteria

- Special requirements on delivery
- Special erection limitations
- Column differential shortening information
- Special fabrication and erection tolerances [118]

5.2. Representation of construction tolerances

During specification of manufacturing processes, a tolerances modeler could create a target tolerance, based on standards, obtained from statistical procedures that every construction system creates within its domain. Tolerances can also be embedded in a building model considering maximum and minimum statistic values and could be represented as a callout on shop drawings. These approaches neither estimate specific tolerances for every situation nor coordinately integrate industry standards or domain specific know-how into BIM platforms. Rather, they require an individual with the proper expertise to estimate and allocate the allowances. Furthermore, current tolerances approaches do not generate pre-visualization of the outcomes to understand the real impact of decisions taken in the design stages. It is very common, for example, to consider tolerances as attributes of individual features or groups of features, which bear no relation to other features or tolerances. This condition allows allocation of tolerances to one feature at the time but it does not allocate tolerances of complex assemblies with several levels of nested datum frames, as it is the common scenario in construction. As a result, actual tolerance applications do not meet the requirements about flexibility and complexity management that building construction requires. This is part of the motivation for creating a system-level approach to model tolerances in construction. This section

will cover the evolution of the most relevant standpoints about tolerances representation and their considerations for being used in the construction industry.

5.3. Mathematical approach to represent tolerances

This section summarizes the basics of mathematical and geometric representation approaches that have been considered as critical for single domain construction tolerances (SDCT) and heterogeneous construction tolerances (HCT). It is important to emphasize that this project will not create a new mathematical method of tolerance calculation or a statistical treatment of tolerances. Rather, this dissertation proposes a novel modeling framework by which a mathematical method for representing manufacturing knowledge can be embedded in a system model and tied to a geometrical (CAD/BIM) representation of building components to assess manufacturability and calculate feature-based tolerances.

The mathematical models for calculating and representing geometric variation and tolerances have been developed using the tolerance zone approach (statistical), the variational geometry approach, or other variational models. With the aim of creating mathematical formulations for geometric variation and tolerancing, all the variational models appear to be suitable for implementing in current solid modeling tools as the ones encountered in BIM tools. The variational model of an object is constructed from its nominal boundary model by allowing each of the bounding surfaces to be varied within some specified tolerance zones [119]. According to Hoffman [120], any tolerance specification corresponds to a set of inequalities, which are of the following type:

Equation 1: Hoffman's tolerances formula

$$L \leq f(x) \leq U$$

Where:

x = parameter vector of a part (critical dimension)

f = tolerance function

L, U = lower and upper bounds of the tolerance zone

Numerous authors defined mathematical approaches for describing this tolerance function and its proper representation. Hillyard and Braid [121] created the concept of variational geometry that is a dimension-driven, constraint-based technique. They used this concept to analyze inconsistencies in the specification of dimensions and tolerances in CAD models. Lin et al. [122] promoted the variational geometry approach from the viewpoint of the user interface and computational efficiency. Requicha [37] introduced the variational class notion for representing tolerances in solid models. Turner and Wozny [123] developed a model based on the variational approach where specified tolerances are used to directly define the valid regions covered by the model variables. Gupta and Turner [124] expanded the previous model to a surface-based variational model in which the model variables are linked to the coefficients of the equations of each surface and the vertex coordinates are computed from the intersection of surface equations. Liu and Dong [125] presented a solid boundary-based tolerance representation model that is comparable to Turner and Wozny [123] model. Whitney and Gilbert [126] presented a tolerance representation method using matrix transformations to propagate tolerance data that is suitable for tolerance analysis of assemblies. Efforts have also been made to reproduce functional requirements in tolerance representations. Rivest, et al [127] proposed to represent tolerance from a manufacturing point of view while Jayaraman and Srinivasan [128] did so from an assembly point of view. The possibility

of using statistics and probability methods for allocation of tolerances has also been explored with the intent of developing tools for tolerance synthesis.

Besides the general purpose variational models mentioned above, there are other important mathematical considerations, specially related to HCT, which will be addressed during this research (for instance, the joint design formula and the accumulated tolerance equation, both introduced by Ballast [29]). For the joint design formula, the size of the joint depends on several factors such as movement-expected tolerances of the assembly, and the flexibility of the sealant (if any). All these factors will vary from case to case due to numerous other issues such as thermal expansion or contraction, gravity, and deflection. From these factors it is possible to derive equations such as the following to size a specific joint:

Equation 2: Joint design equation

$$J = \frac{100(e\Delta tL + S)}{M} + T$$

Where:

J = Joint Width, in

e= Coefficient of thermal expansion in/in/F

Δt = Expected temperature change

L= length of the material joined

M= movement capability of the sealant in inches (if any)

T= nominal tolerance of the material (offset inches)

S= Other expected movement caused by seismic forces or other non-thermal causes

The second example approach for HCT that must be considered in any computational implementation of geometric variation and tolerancing is the accumulated tolerances factor or tolerances stack. Dimensional variation during manufacturing accumulate or stack-up statistically and propagate through an assembly in a kinematic fashion, causing critical features of the products to have degrees of variation. The tolerance stacking problem arises in the context of assemblies from interchangeable parts because of the inability to produce or join parts exactly according to nominal. Either the relevant part dimension varies around some nominal value from part to part or it is the act of assembly that leads to variation. To calculate this factor it is necessary to consider variations of all the components of an assembly, which can be in different directions with different magnitudes. As an important note, tolerances calculations need a specific theoretical datum from where they are measured. This means, from a theoretical datum plane, the calculation applies only to the direction described as normal to the datum plane. Thus, they are unidirectional. Also, where tolerances of individual components are different in "+" and "-", in order to get independent calculations, two different equations will be necessary. The basic accumulated tolerances equation, based in the Root Sum Square (RSS) technique is represented as follows:

Equation 3: RSS basic formula

$$\boldsymbol{T = \sqrt{t_1^2 + t_2^2 + t_3^2 + t_4^2 + t_n^2}}$$

Where:

T= Total tolerance

t_{n^2} are the single tolerances of each element that participates of the specific assembly in inches.

The following stack-up analysis process has been adapted from [99]:

1. Identify the measurements that in sequence control a critical dimension or feature parameter. This critical task must be achieved by applying material systems-specific conditions to the calculated assembly. In this section, allocation of tolerances must be designed. This is not automatically allocated because identifying critical dimensions through assembly analysis is not trivial and requires artificial intelligence capabilities.
2. The mean assembly measurement is attained by summing the mean of the dimensions in the chain as it has been explained in the previous paragraph.
3. The total variability will be projected by the accumulation of variations of each component in the stack-up process.
4. Variation of the assembly is ideally compared to the engineering limits (lower limit and upper limit) to assess the amount of rejects or non-conforming assemblies. One significant attention about tolerances is categorizing the proper data to compare the analysis results. In this task, comparison to actual measured data is preferred. However, in the absence of measured data, comparisons must be performed against data from similar parts or processes [129]. This later statement is what led to the development of know-how data, or domain-specific knowledge that comes from experience rather than from formal documented standards. For this dissertation, and because the author has been successfully exposed to this informal (yet proven) know-how data, an important focus is to create a knowledge modeling environment to organize and formalize these insights.
5. Design changes in parts and assemblies may be made after evaluating the analysis outcomes.

Tolerances analysis is a quantitative tool for predicting the accumulation of variation in an assembly by performing a stack-up analysis. The weakness of the previous model is the assumption that all distributions are perfectly nominal-centered and perfectly Gaussian, or normal. Because these two assumptions allow for a simple calculation of compound probability, they do not represent the vast majority of manufacturing processes and quality control systems which exist in the real world production environment.

The following table shows the most common tolerances stack-up models of this project. For example, the Worst Case (WC) delivers the extreme limits of the sum of absolute values of tolerances to obtain the worst combination of tolerances stack. Also, as was seen in the previous paragraph, the RSS adds the variation by means of a Root Sum Square approach. This approach provides preliminary insights about statistical distribution of the tolerances problem. In the following equations, the total variation of the RSS model is divided by three to fit within three standard deviations range.

Table 1: Tolerances stack modeling equationsadapted from [129]

Model	Stack Formula	Key Use	Application Kind
Worst Case (WC)	$\sigma_{ASM} = \sum \lvert T_i \rvert$ Not Statistical	Extreme limits of variation	Critical Systems. Most costly model
Statistical (RSS)	$\sigma_{ASM} = \sqrt{\sum \left(\frac{T_i}{3}\right)^2}$	Probable variation	Reasonable estimate. Some rejects allowed
Six Sigma (6σ)	$\sigma_{ASM} = \sqrt{\sum \left(\frac{T_i}{3C_p(1-k)}\right)^2}$ Cp= Process capability index. k= Drift factor	Long Term Variation	Drift in mean over time expected. For high quality
Measured Data (Meas)	$\sigma_{ASM} = \sqrt{\sum \sigma_i^2}$	Variation using existing part measurements	After parts are made. What if? study

In the previous table, the Six Sigma equation accounts for high quality by altering the stack-up equation to include the process capability index (Cp) and the drift factor (k). Cp represents the ability of a process to produce output within specification limits. As Cp increases, the contribution of that dimension decreases, causing the total variation to decrease. The drift factor (k) measures how much the mean of a distribution has been observed to drift during production. This drift factor ranges between 0 and 1. During simulation, where there is no data about drift factor, it usually values 0.25.

From the previous Table 1, Worst Case (WC) will compute extreme limits by summing absolute values of the tolerances to obtain the worst combination of wrong dimensions.

The statistical model will add variations by root-sum-squares (RSS). As this approach considers statistical probabilities of possible dimensions combinations, the predicted values using this approach are more reasonable. RSS predicts the statistical distribution of the assembly feature, from where percentage of rejects can be obtained [99].

The following example shows an assembly of nine components containing the same precision of $T = 0.01$.

Equation 4: WC scenario example

$$WC: T_{ASM} = \sum |T_i| = 9 \times 0.01 = \pm 0.09$$

Equation 5: RSS scenario example

$$RSS: T_{ASM} = \sqrt{\sum T_i^2} = \sqrt{9 \times 0.01^2} = \pm 0.03$$

It can be seen that WC predicts more variation than RSS and that difference will increase as the number of components of the chain increase as well.

In the reverse case, if we had a $T_{ASM} = 0.09$ and we want to calculate the reversed stack analysis, the component tolerance can be determined from the assembly tolerance:

Equation 6: reversed WS analysis example

$$WC: T_i = \frac{T_{ASM}}{9} = \frac{0.09}{9} = \pm 0.01$$

Equation 7: reversed RSS analysis example

$$RSS: T_i = \frac{T_{ASM}}{\sqrt{9}} = \frac{0.09}{3} = \pm 0.03$$

It can be seen that WC requires much tighter tolerances than RSS to meet the assembly requirement.

Also demonstrated by [129], two other important examples of mathematical models to assess variability in mechanical or construction assemblies are:

- Prediction of rejects during manufacturing
- Calculation of the percentage of contribution of a possible geometric deviation in a part or assembly

Usually, most of produced parts will be grouped close to the mean value, causing the charts to increase in the middle. As you go further from the center, fewer parts will fall there, causing the frequency chart to decrease to zero at the extremes. In the following equations, UL and LL give the upper and lower limits of dimensional variation, as they have been obtained from design requirements. Any normal distribution may be converted in a standard normal curve distribution, where the mean will be 0 and the standard deviation will be 1. In this case, instead of plotting the frequency versus size, the number of standard deviations from the mean are plotted. Thus, it is possible to determine the fraction of assemblies that will fall out of the engineering limits [129].

This process is carried out as follows:

1. Run a tolerance stack-up analysis to get the mean and standard deviation of the assembly dimension X, which has design requirements X_{UL} and X_{LL}.
2. Obtain the number of standard deviations from the mean to each limit

Equation 8: standard deviation from mean at upper limit

$$Z_{UL} = \frac{X_{UL} - \bar{X}}{\sigma_X}$$

Equation 9: standard deviation from mean at lower limit

$$Z_{LL} = \frac{X_{LL} - \bar{X}}{\sigma_X}$$

3. Were $\bar{X}$ and σ_X are the mean and standard deviation of the assembly dimension X, and $\bar{Z}$= 0 and σ_Z = 1.0 are the mean and standard deviation of the transformed distribution curve.
4. Using standard normal tables, look up the fraction of assemblies lying between Z_{UL} and Z_{LL} (under the curve). As explained in [129], this is the predicted fraction of assemblies that will meet the requirements. What is outside the limits is 1.0 – yield. These are predicted rejects that are expressed as parts per million (ppm).

Percent contribution gives the designer the ability of calculate how every feature variation contributes to the resultant assembly variation. With this tool, it is possible to decide where to concentrate efforts for reducing construction variability. The percent contribution factor is simply calculated as the ratio of a component feature dimensional standard deviation to the total assembly standard deviation.

Equation 10: percentage of contribution formula WC

$$WC: \%Cont = 100 \frac{T_i}{T_{ASM}}$$

$\%Cont$: Percent contribution

T_i: component feature dimensional deviation

T_{ASM}: Assembly dimensional deviation

Equation 11: percentage of contribution formula RSS

$$RSS: \%Cont = 100 \frac{{\sigma^2}_i}{{\sigma^2}_{ASM}}$$

$\%Cont$: Percent contribution

${\sigma^2}_i$: component feature dimensional standard deviation

${\sigma^2}_{ASM}$: Assembly dimensional standard deviation

Although this dissertation does not propose new mathematical models for tolerances assessment, or any other kind of contribution to the mathematical domain, the previously explained equations, mostly developed by [129], will be converted in <<constraint>> blocks and seamlessly applied to specific CAD features to assess tolerances of manufacturing and assembly activities. In this dissertation, these equations are assumed to be the most suitable models for being implemented in a construction-oriented tolerances model. The following section will briefly introduce the Monte Carlo method to calculate variations and tolerances for construction.

5.4. Statistical tolerances analysis through Monte Carlo method

The Monte Carlo approach has been standardized by the Guide to the Expression of Uncertainty and Measurement (GUM) [130]. To determine the quality level for assemblies before actual construction, an exploration of variation using an uncertainty approach is required. This strategy allows complex parts of buildings to be analyzed and improved before the first physical structure is built. A reliable way to apply a mathematical model to this approach is by means of Monte Carlo simulations. This

method can be applied to situations where it is possible to create formal equivalence between the preferred result and the anticipated behavior of a stochastic system. Through the Monte Carlo method, it is possible to obtain better results in less time than using deterministic techniques [131]. In many cases, the calculations of tolerance deviations of very complex assemblies cannot be realized using deterministic approaches. Having simulated and calculated the tolerances of a given part of assembly, the next step will be the allocation of tolerances analysis results in the CAD model. The next section introduces the fundamentals of this matter.

5.5. Model Simplification to represent tolerances

Currently, there is a misinterpretation about tolerances capabilities of current CAD packages. Some of these tools are believed to have automatic tolerances capabilities. Yet, what they actually do is create a callout as a placeholder from a part or feature, indicating the plus/minus allowance. In contrast, what is really important is to know how those callout values were calculated, and where to access the material system knowledge that led to those calculations. A better approach is to divide the efforts for representing tolerances into two main groups: system dependent and system independent. Accordingly, the first category focuses on representation of tolerances information within a specific geometric modeling system, and the second category focuses on geometric modeling and tolerances allocation as separate tasks. In this dissertation, based on assumptions of understanding geometric variability and geometry as parts of the same entity, this study focuses on the system dependent option. Furthermore, an improved approach of representing geometric variability in a solid model is to embed manufacturing knowledge as calculated values of the nominal geometry of the object. In

order to accurately assess manufacturing compliance and tolerances in a timely manner, one must utilize simplified model views that retain the important details and eliminate the irrelevant ones. Here is where a Systems Engineering approach, based on the SysML language, performs most properly. SysML enables a model simplification process that does not disrupt the integrity of the solid model. Furthermore, based on domain specifics profiles, this process can filter the geometric information, thus creating model views with a sub-set of the instantiated meta-classes, which are geometric features of the original model.

The minimal element that can represent geometric variation and carry tolerances is the feature. Features cannot be understood as independent from each other just as variational information cannot be independent from nominal geometry. There are several sub-categories of features based on their relationships. These are: lower-level features (e.g. the basic topological entities, faces, edges, and vertices) and higher-level features, which are the combination of the lower-level features (or the combination of other higher-level features) having certain functional relationships among themselves. This separation between higher and lower features is crucial to achieve model simplification without producing inaccurate results. For the implementation of a knowledge-based tool to assess manufacturing compliance, this dissertation will use higher-level features as its basic modeling meta-class, and lower-level features will only be instantiated as value holders.

Existing model simplification techniques that are useful from a physics-based simulation point of view are broadly classified in four categories, based on the type of simplification operators used in their respective techniques. The first simplification

category is surface entity developed by Sheffer [132] and Lee [133] [134]. The second category is volumetric entity developed by Andújar, Brunet, and Ayala [135]. The third category is explicit feature and dimensional reduction created by Joshi and Dutta [136] and Zhu and Menq [137]. The last category is dimension reduction based operations developed by Rezayat [138]; Donaghy, Armstrong, and Price [139]; and Thakur and Banerjee [140]. There are also some recent experiences of simulating variational geometry in the automotive domain. Wickman et al. [141] joined a commercial virtual reality tool with a variation simulation software to visualize non-nominal variation in a photo realistic atmosphere. Another method for visualization of non-nominal variation was offered by Maxfield, Zhao, Juster, and Fitchie [142]. This method was meant to meet the demands concerning packaging and visualization that can be used for faster investigation of the variation in complex assemblies. Lo, Lindkvist, and Soderberg [143] introduced a general procedure to compute and visualize the total volume in space a part or assembly creates when it is affected by displacement or motion.

5.6. Allocating Manufacturing Knowledge and Tolerances on Solid Models

A representation of a solid is defined as a mapping from a mathematical model of a solid onto a set of symbolic structures or representations. If a computer representation is to be used to calculate geometric properties, it must possess certain formal properties. These characteristics are: well-formedness, generality, completeness, and efficiency of storage data [144]. The development of solid modeling has been a matter of significant research and growth. Many approaches have attempted to represent solids in a truthful way, the most significant being spatial occupancy enumeration, Constructive Solid

Geometry (CSG), and Boundary Representation (B-Rep). Among these three approaches, B-Rep has been the most advanced, and also the most common found in 3D modeling applications. B-Rep is built from two main sources of information. One is dimensional and locational (geometry), and the other is about relations and rules among its elements (topology); both structures depend on each other to achieve well-formedness and unambiguity of the shape. Solid modeling systems, found in computationally complete representations of 3D solid objects, are used to represent nominal geometry. Technically, these systems permit any well-defined geometric property of a solid to be calculated automatically. This allows solid modeling systems to provide the geometric data necessary for conducting design and construction activities such as finite-element analysis or digital manufacturing. Even though the representation of the nominal shape of mechanical parts with computers is successfully performed with solid models, representation of geometric variation and tolerances, or representation of manufacturing rules to ensure a smooth fabrication and assembly processes in construction, have not been equally advanced.

A simple solution of representing manufacturing knowledge in a solid model is to hold tolerances as attributes of geometry of the object as it is modeled. In order to accurately represent geometric tolerances in a timely manner, simplified models that retain the important details and eliminate the irrelevant ones are most desirable. In this scenario, the implementation that is proposed in this dissertation specifically deals with this issue. Fully represented solid modeling data is converted into system data by decomposing the features tree of the CAD into a sub-set of <<block>> instances that carry only what is necessary to perform a tolerances calculation. These filtered yet

consistent data are assessed by formally represented pieces of manufacturing knowledge called <<constraint>> blocks.

When linking manufacturing knowledge and tolerances allocation with Solid Modeling, there are two important aspects that need to be addressed: (1) how to create a variational model based in object parameterization and (2) how to assess manufacturability and allocate tolerances on a solid model based on construction knowledge:

1. In Solid Modeling, the set of features that is involved in any tolerance specification or geometric variation is a sub-group of connected elements. This sub-group contains several parameters that can be managed using the object parameterization capability of any parametric package available in the market. This approach of object parameterization is a starting point to describe a variational model in Solid Modeling. The object parameterization of Solid Modeling has two main approaches: direct parameterization and indirect parameterization. In the first case, the user will directly assign all the parameters of the model as object dimensions to produce geometric variations or instances. By using indirect parameterization, the user defines the model, and then attaches dimensions. This has the effect of defining the dimensions in terms of the model parameters [145].
2. The second aspect is manufacturability and tolerances allocation based on manufacturing knowledge. Computer-based tolerances representation in commercial solid modelers is application-oriented and usually different from ISO/ANSI/ASME standards to describe GD&T [36]. Most common systems are variational geometry constraint-based systems in which tolerances are specified on sketches. Tolerances

are specified as the variation of dimensional constraints (for example, distance between two points) and geometric constraints (for example, parallel lines). Tolerances are represented as the variation of the position of control points, for example, at an intersection of lines or center of a circle. Although these approaches are useful to allocate tolerances values, as it has been previously explained, they lack methods to compare such numbers with domain-specific knowledge within a feature-based geometric context. Therefore, having the ability to describe tolerances values is not the problem. The challenge is to understand how such values are constructed and how these values affect and are affected by other features of the assembly.

Besides the implementation of geometric approaches to describe tolerances in solid models, their integration into a system model through SysML is another important challenge. The following section will address this matter by introducing the main issues and the keys for such an integration.

CHAPTER 6: Methodology

Complexity of product models for construction happens both at the high abstraction level, where requirements have to be modeled and maintained, and at the low abstraction level, where detailed design is performed by means of specific design parameters from different domains and stakeholders. This complex scenario, based on a highly heterogeneous body of information, makes it difficult for average BIM operators and building designers to integrate knowledge and tools among construction domains. These skills are more typical of software developers and computer scientists. For this reason, it is critical to develop software that seamlessly integrates different domain-specific applications to eliminate the need for hard coded, ad hoc solutions every time that integration is required.

A proper methodology for modeling and representation of construction tolerances needs to satisfy two basic set of requirements: compatibility requirements and computability requirements [146]. The first set is required to generate consistency with construction practice. This set basically digests the representation of all types of dimensions, representation of material systems, SDCT and HCT, tolerances stack, material conditions, and manufacturing processes. The second set, computability requirements, adds support for model-to-model transformation, feature-based integration among applications, extraction of critical dimensions from CAD features, model consistency assurance, and inspection of feature types to allocate tolerances. All told, compatibility requirements are tied to the acquiring and representation of manufacturing know-how and standards for construction and computability requirements are tied to the unambiguous, consistent representation of this knowledge in a SysML-CAD

environment. Besides the application developed in this project, additional software required to create the integrated environment includes:

- MagicDraw (Version 17.03 used for this project) provides the System Modeling environment;
- SysML plugin for MagicDraw provides the SysML profile that works on the UML[9] environment;
- Siemens NX (Version 8.5 used for this project) is the CAD package used for the implementation;
- Maple (Version 17 and 18 –beta-- used for this project) is the mathematical engine that calculates tolerances analysis and allocation for this project;

The following diagram shows the software environment for the present implementation. From left to right, the CAD application will be queried by a set of pre-established routines (NX client[10]) created in Maple. On the right side, the developed tool will be allocated in Magic Draw by means of a JAVA implementation. Here, other domain-specific tools can be also integrated through SysML profiles. For example, a cost analysis tool could be coordinated with a tolerances analysis tool to evaluate the cost impact of manufacturing decisions.

---

[9] The Unified Modeling Language (UML) is a general-purpose modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system

[10] NX client created by the author has been released in the beta version of Maple 18, commercially available

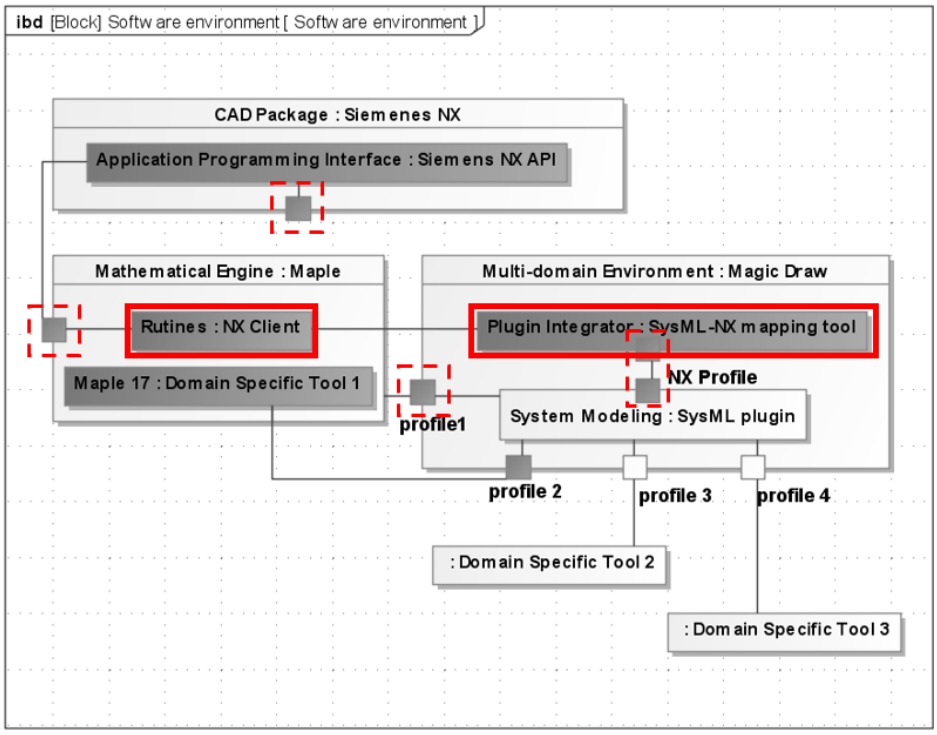


Figure 24: Software environment of the implementation. Solid red lines define new pieces of software developed for this dissertation and dashed lines represent specific integration between different tools.

6.1. From domain issues to functionalities proposed for the modeling framework

Figure 25 shows the general hierarchy of construction variability issues identified from the literature and from interaction with manufacturers. The dark colored boxes, depicted under “rule- or knowledge-based” sources of variability are the main focus of this dissertation, and that have been listed in Section 4.1. of this dissertation.

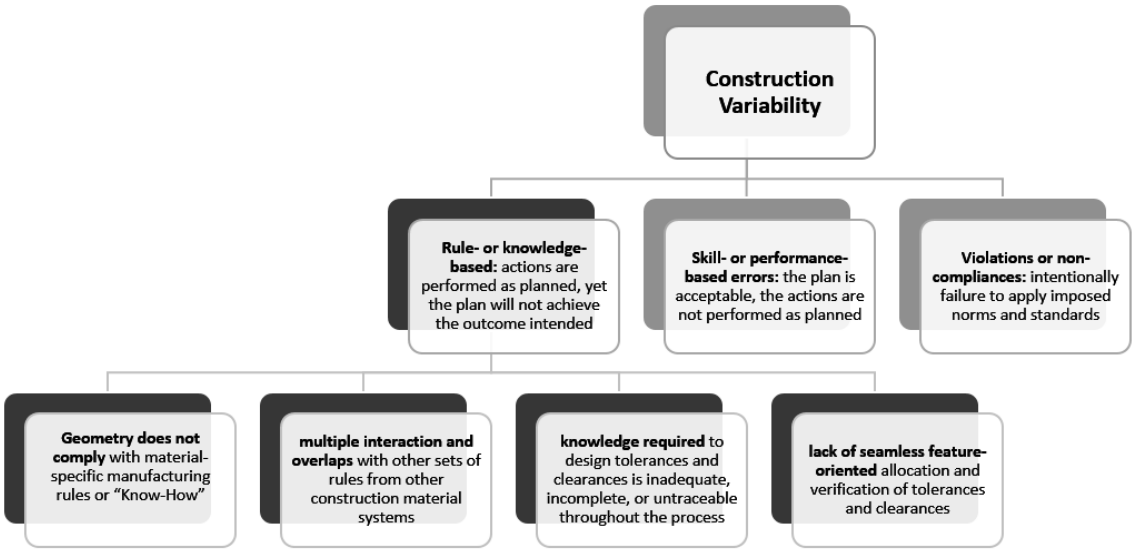


Figure 25: General hierarchy of construction variability issues adapted from [12]

The following figures show the current modeling methodology with and without the proposed framework. In Figure 26 the depicted diagram represents the current approach to inform design in building construction. In this case, material-specific knowledge is never formally integrated with the assembly geometry. Assumptions about material interactions and components design, rather than formal feature-based assessments, create room for inconsistencies between design specifications and manufacturing-compliant geometry. In this dissertation, a formal connection between material-specific knowledge and geometric features is the proposed way to assure the full validation of the building requirements. Also, a proposed tolerances assessment will ensure that components and assembly are compliant with manufacturing rules and know-how. In order to implement this approach, the interactions diagram needs to incorporate a new element that will open several other kinds of relations in the process.

A geometric constraint is proposed as the negotiating point between material-specific knowledge and design geometry. Furthermore, a geometric constraint can be the

formalization of a piece of manufacturing knowledge. For example, a basic formula to calculate the minimum bending radius of sheet metal is $r = t$, where r is the radius of the bending and t is the thickness of the sheet metal part. Then, the mathematical expression $r = t$ represents a portion of domain-specific (sheet metal) manufacturing knowledge. This basic piece of manufacturing knowledge can be automatically evaluated in a geometric feature if CAD parameters and knowledge are linked together. In order to make this geometric constraint operational, most of the exchanges depicted in Figure 27 must be programmatically formalized. In this dissertation, the material system-specific knowledge will be formalized as a specialization of Systems Engineering requirements, which will be programmatically linked to their formalizations as constraints. Also, another internal loop of the process, which involves design specifications, geometric constraints, and geometric features, must be automated. As it can be seen in Figure 27, design specifications will inform geometric features, as previously depicted in Figure 26. However, parameters of geometric features will populate the domain-specific constraint, which will verify that the design specification is in compliance.

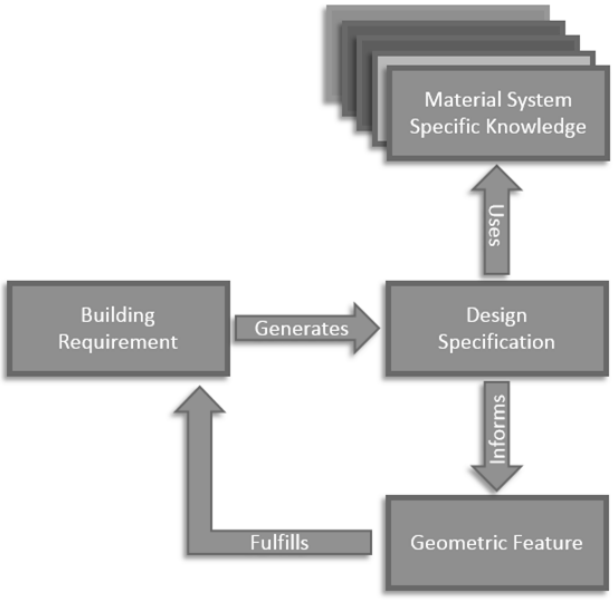


Figure 26: Modeling approach without the proposed implementation

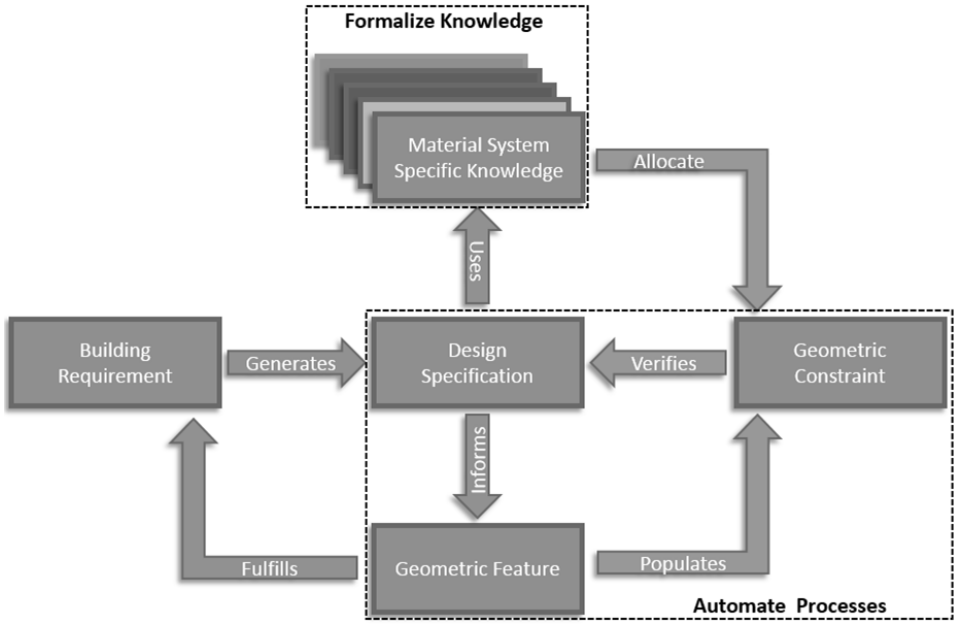


Figure 27: Modeling approach with the proposed implementation

In order to implement the proposed general modeling framework presented above, the following list of general functionalities will be developed in this dissertation:

- **Model-to-Model Transformation:** structural, feature-based decomposition of parametric CAD models into system models.

- **Model Integration Approach:** parametric, real time, seamless software integration for knowledge allocation, analysis, and verification to reduce human data translation.
- **One Truth, multiple model views:** centralized project requirements, geometry, and design specifications in an interoperable modeling environment.
- **Domain Expert Advice:** automated allocation of material-specific knowledge for components and assemblies based on geometric features and material systems.
- **Machine Readable/ Executable:** CAD geometry programmatically integrated to manufacturing know-how through knowledge-based mathematical and logical constraints.
- **Model Consistency Approach:** On-demand model-to-model and tool-to-tool consistency assessment and model data update.

In the previous sections, this paper has described the nature of construction tolerances, created a taxonomy of the domain problem, and has explained the implications of a system-level computational implementation. The next sections will convert the previous set of system requirements into specific activities that have been programmatically implemented during the development of this dissertation.

6.2. SysML-CAD integration

In order to create a knowledge-based modeling environment that assesses manufacturing compliance of geometric data, a SysML-CAD integration is proposed. Many efforts have attempted to integrate domain-specific engineering views into the SysML environment. However, most of these approaches do not integrate with geometric-based applications. Recently, there have been several initiatives in Model

Based Systems Engineering (MBSE) and Knowledge-Based Engineering (KBE) to face this integration disadvantage. However, the majority have proposed ad hoc solutions that are only useful within a short term development. According to Rocca [92], these integrations lack defined guidelines and standard procedures. Integration and consistency issues between MBSE and CAD can be analyzed from a general high level perspective regarding Systems Engineering (SE) and also by taking a closer look at the low level integration of tools and programming languages. In this dissertation, we will understand high level as general objectives of a specific computational method and low level as the detailed executable computer implementation.

From the most general judgment about SE, one of the obvious and most important challenges is dealing with multiple views of a complex system (in this case a building). Each view represents a specific set of information that will interact with other views of the same system. For example, a building component can be diagrammatically described at a general level in SysML by decomposing its features tree (Figure 28) and will also have a geometric low level description within a CAD representation (Figure 29).

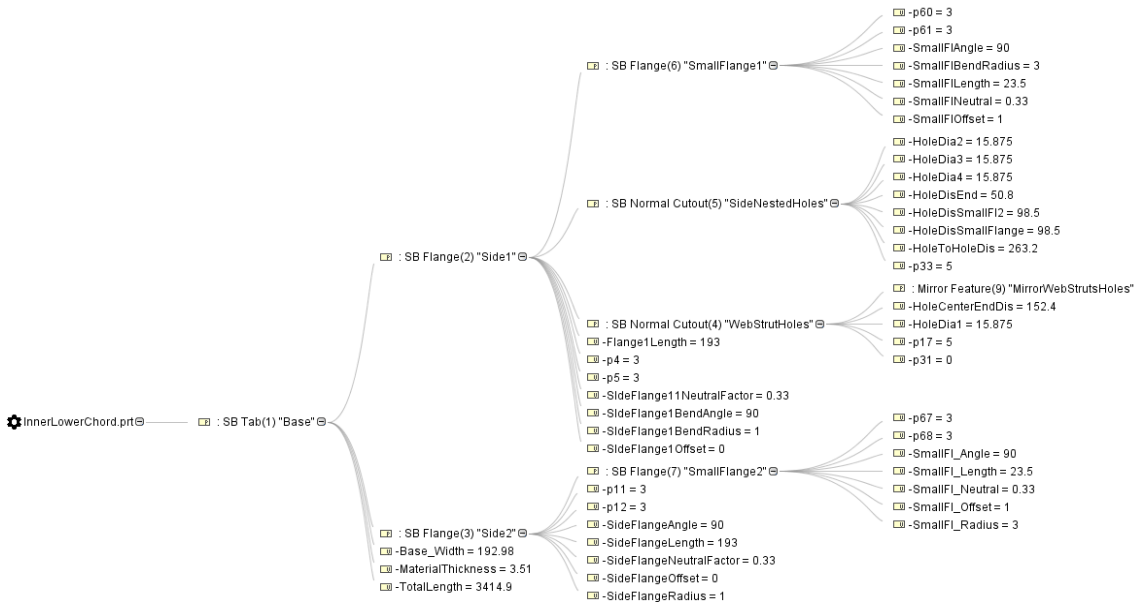


Figure 28: Features tree view of the building component "InnerLowerChord"

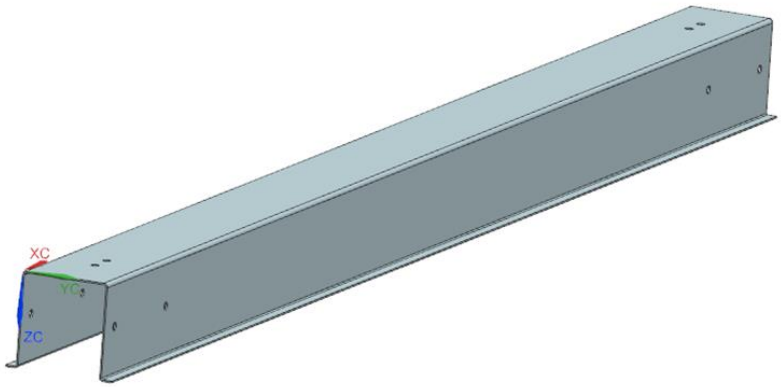


Figure 29: CAD representation view of the building component "InnerLowerChord"

Because this multi-view approach will create dependencies between models from different domains, a consistency issue among the corresponding design models arises. Specifically, this happens because two or more views can affect a shared attribute of the design and for that reason, the association between models' elements and the parallel changes must be consistent. As Shah et al. [88] stated, maintaining consistency between multiple data sets and tool-specific models becomes an issue when analyzing different system architectures during the design process. Additionally, due to the fact that models

are created within different domains and languages, defining the general rules to manage consistency across domains is a significant challenge. Specific problems that arise from consistency issues are the inability to share models in a collaborative environment and the inability to identify model consistency issues until late in the design process.

Apart from the issues regarding the multi-model approach of SE, there are also several challenges that need to be addressed related to the implementation in SysML language. First, because SysML is a general purpose modeling language, it lacks the detailed, formal semantics needed for formal domain-specific analysis and automated tool support [83]. For the same condition of generality, any model can be represented through SysML language. This situation makes it difficult for domain experts to describe models in SysML, thereby reducing the acceptance of SysML for specific domains [88]. This situation is especially common in the AEC domain, where the semantics of system modeling are not readily apparent to the professionals in this area. As it turns out, to ensure the success of this project, it is necessary to address the low integration of the SysML language with direct geometry and geometry-based data management. The next section will review the necessary elements for the functional integration of CAD and SysML.

6.3. SysML-CAD semantic integration through Domain Specific Languages (DSL)

One of the most significant characteristics of Systems Engineering is its ability to deal with embedded systems from different domains. As Shah et al. [88] explain, these multiple domains cover different information maintained in numerous views for each of the various subsystems. Considering this heterogeneous condition, model consistency is

difficult to achieve because different views require different or transformed data. For example, to run a finite element analysis of a building, geometric details such as components' shape and their relative positioning in the space are critical. On the contrary, for a material quantity take off, specific component dimension metrics such as weight or length may be required. However, specific positioning in the Euclidian space is irrelevant. In this quantity take off case, geometric representation is transformed to a set of independent metrics that operate in a non-geometric modeling environment. Therefore, this contradicts the principles of interoperability, where data remain the same throughout different applications. As Mosier [93] stated, a significant gap is observed because of the lack of integration across domains of design tools through domain-specific development activities. For this project, a CAD representation of a building assembly must be consistent with its SysML representation. However, these two modeling approaches differ in their programming and semantic languages. As a proposed solution, one important aspect of the SysML approach is the ability to create domain-specific semantics through Domain Specific Languages (DSL). DSLs make simpler commonly used features of a domain and decrease the need for lower level constructs. Also, DSLs enhance computer interpretability since the information in a valid model is encoded at the meta-level instead of the model level [88]. In this project, a SysML profile that represents the CAD data structure at the meta-level will be created as an NXProfile within the CAD model in Siemens NX. This SysML profile will help the CAD-SysML integration to automate low level and highly manual tasks, the integration of applications and datasets, documentation and report generation, and the simplification and standardization of more complex processes such as system-level tolerances allocation.

6.4. Representation of CAD data structures in SysML

Manufacturing compliance analysis can potentially reduce cost and time generated by construction errors or design omissions. In order to obtain reliable results from this methodology, an automated, seamless integration between geometry (CAD) and a system modeling tool (SysML) is critical. Yet, there are several issues related to the different nature of both architectures (CAD-SysML) that need to be elucidated. Two fundamental considerations of knowledge-based models such as SysML are the lack of geometry handling rules and the highly general modeling environment where these rules operate. First, geometry handling rules do not generally exist in any traditional system-based application because their evaluation involves excessive information about space, solids, and relative positioning of assembly, parts, and features. This is why data structures of CAD systems are extremely complex and resource consuming. However, integrating specific portions of geometric data with performance-based parameters can accomplish operations that otherwise require manual input, which are time consuming and error prone. Second, the design process of building products requires the interaction of multidisciplinary teams and vast amounts of mixed project data. Every part of the design process is carried out through domain-specific models, tools, and knowledge that create a heterogeneous complexity. Because of its generality, SysML is able to represent and integrate many of these domain-specific bodies of knowledge by using interfaces called profiles. For an integration of SysML with a CAD tool, a profile must represent key aspects of the data structure of that specific CAD package. This data structure is also called meta-model, and for a CAD package as Siemens NX, the basic meta-model is

centered on the traditional assembly/part/feature paradigm, as is shown in the next section.

6.5. General description of the present project:

The present dissertation proposes the development of a Knowledge-Aided Modeling Framework that integrates a parametric CAD tool with a System Modeling application to assess manufacturability and tolerances in construction. The CAD tool provides robust geometric modeling capabilities, while System Modeling allows the specification of feature-based manufacturing requirements aligned with construction standards and construction processes know-how. With this approach, manufacturability assessment and the identification of conflicting interactions between tolerances requirements of building material systems are performed.

The methodology for the implementation of the proposed modeling framework is composed of the following six activities, which will be developed in detail further in this document.

1. **Structural Decomposition:** This includes the creation of a feature-based representation of the CAD model in the SysML environment. It follows the project>assembly>part>feature>parameter approach to describe geometry. Also, it creates a data graph based on CAD meta-model, which defines the languages and processes from which to form a model.
2. **Knowledge Acquisition:** This corresponds to the domain-specific knowledge, and its formalization, necessary for a manufacturing compliance analysis or optimization/verification processes of an assembly or section of a building. The knowledge acquisition process will be carried away manually by adding specific

rules as <<requirements>> in SysML, which will be further specified as <<Design Specification>> or <<Manufacturing Specification>>. However, all knowledge created will be stored and be ready for use by searching within the domain-specific knowledge folder in the SysML Model. This folder will have manufacturing requirements that lead to manufacturing specifications represented as mathematical expressions such as <<constraint>> blocks.

3. **Knowledge Allocation:** CAD features decomposed in numeric parameters from CAD data will be connected to <<constraint>> blocks that carry domain-specific knowledge about materials or processes for the imported CAD file. The allocation process will be executed automatically. The created application will query the imported CAD model by looking at its features <<stereotype>> and will offer the user options to link <<requirement>> blocks and <<constraint>> blocks that match the feature types.
4. **Parametric Execution:** The application created in this dissertation will execute all the domain specific <<constraint>> blocks using numerical data obtained from the CAD models. This geometric information will be stored in <<instance specification>> blocks in a specific, user-defined folder within the SysML model. The <<instance specification>> blocks store results of parametric executions so that the user can compare them and pick the best analysis scenario for a given analysis context.
5. **Specifications Verification:** Routines coded for this implementation in SysML and Maple will evaluate and verify the consistency between CAD metrics and the formal definition of manufacturing requirements about tolerances. This

verification will be evaluated by defining two customizations of the NX value property stereotypes: <<Validation Value Property>> and <<Performance Value Property>>. The main difference between both stereotypes is that the <<Validation Value Property>> is typed as Boolean, and the <<Performance Value Property>> actually carries a real value derived from the geometric data. Although not present in the CAD geometry, these indicators will be critical to assess the consistency and manufacturing compliance of the CAD model.

6. **Knowledge Compliant Geometry Update:** This stage defines a series of functions that will consolidate changes produced in the model on either the CAD or the SysML side. In an integrated framework, changes might be produced in different domain-specific applications. For this implementation, if changes that were positively evaluated by the application were produced on the CAD side, there will be an "update SysML model from NX" command in the SysML menu. Conversely, if changes were made in the SysML side, there will be an "update NX model from SysML" command. Both commands will use the consistency checking engine that is presented in section 7.13 of this document.

6.6. Explanation of the Modeling Framework Through a Case study: Cylindrical Fit

The aim of this first case study is to navigate through all the different components of the implementation by using a simple example of manufacturing integration between dissimilar material systems. The detailed explanation of the methodology will be divided according to the implementation activities presented in the previous section. This

classification of stages will also integrate pertinent references to previous works and will show real screen captures of the proposed software interface and constructs.

This case study presents a double cylindrical fit with two different materials: concrete and steel. Radial clearance between two telescopic tubes is very important and, for a great number of applications in construction, a high degree of precision is needed when two tubes are expected to slide one within the other. Without a high degree of precision, the wrong clearance within the fit of the two tubes can cause the telescopic action between the two tubes to lock up. In an effort to prevent this, a quality sliding motion is needed.

Evidently, the allowable clearance between the two mating tubes is a function of the length of engagement. Thus the longer the engagement, the more radial clearance can be tolerated. This is true for two tubes that are expected to slide freely relative to one another (telescopic tubing) as it is for a metal bushing sliding up and down a precision shaft.

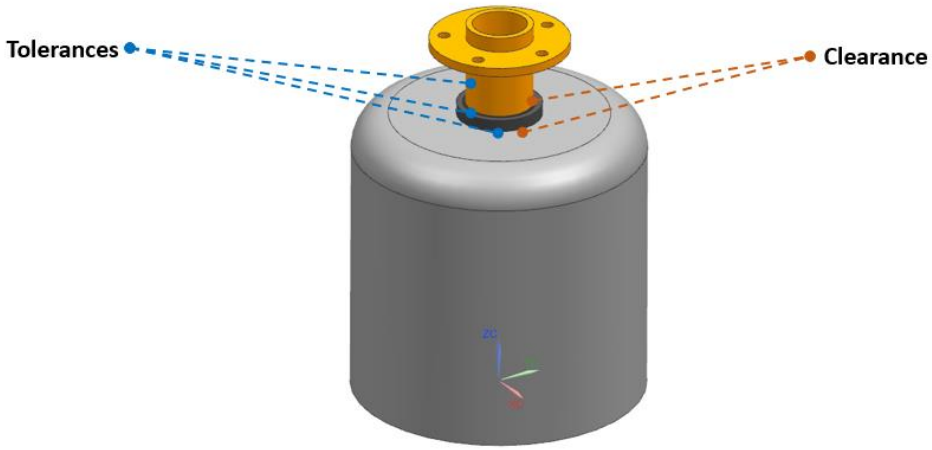


Figure 30: Case Study1: Double cylindrical fit of a multi-material assembly

For this simple example, there are two types of variations that must be addressed during the study. The two variations are the manufacturing tolerances of the three mating

components, specified independently, and a radial clearance among the three components to allow the sliding effect that the functional requirements specify for the assembly. In both cases, the tolerance specifications refer to Single Domain Construction Tolerances (SDCT) as described in a previous section of this document. In the current practice of building construction, these kinds of specifications are generally overlooked or only prescribed as feature-independent rules via a general callout in the construction documentation. In contrast, this research proposes these kinds of specifications as parametrically constrained by the specific instance of a feature within its assembly conditions .

Table 2: Manufacturing data available previous to the tolerances analysis

	CAD_Part	Dimension Name (CAD expression)	Nominal Value	Lower Limit Tolerances	Upper Limit Tolerances	Cent. Dimension	Plus/Minus Tolerances		Tolerances and Clearances OK
1	Ballast	InsideDimBushing	6	?	?	?	?		?
2	BallastInsert	BallastInsertDia	6.5	?	?	?	?		?
3	CircularPlate_BaseTube	PipeDia	6	?	?	?	?		?
4	**Clearance:** Ballast-BallastInsert					?	?	WC	?
5							?	RSS	?
6	**Clearance:** BallastInsert_CircularPlate-BaseTube					?	?	WC	?
7							?	RSS	?

Considering that all the fields shown in previous table are critical for anticipating the manufacturing performance of the assembly, it is evident that having only nominal values from the CAD domain will not create enough context for the tolerance and clearance analysis. Rather, this sole exercise will require the integration of different pieces of knowledge as shown in the following

Table 3. Besides the CAD data, at least three other stages with their own specific knowledge will be developed (Material-specific manufacturing knowledge, tolerances/clearances assessment, and tolerances/clearances validation). The material-specific manufacturing section of the table will define Lower Limit Tolerances (LL) and Upper Limit Tolerances (UL). Usually, the common practice for construction uses the same value for both limits. This approach is called +/- (plus/minus) tolerances. However, in modern engineering, these values are independently calculated based on estimations that consider geometric and material characteristics, which define the material-specific manufacturing knowledge. In this case study, for example, the proper specification of a bushing condition will most likely define UL and LL, both at negative values from the nominal parameter. Then, the following basic stage for manufacturing compliance will be the assessment of the values obtained from the integration between the CAD feature parameters and the material-specific manufacturing knowledge that define their upper and lower limits. In this stage, besides calculations of centered dimensions and +/- tolerances, the specification of assembly clearances will be performed. An important note at this point is to establish the fundamental difference that exists between tolerances and clearance. Tolerances refers to the limit of unintentional deviation of a dimension from its nominal value and clearance is the amount of intentional deviation between two mating dimensions in a fit. Finally, the tolerances validation stage will confirm, by using performance indicators established from the combination of different material systems knowledge, if the manufacturing allowances will be optimal for the assembly.

Table 3: Integration of different analysis stages to finally validate a clearance prescription

	CAD Data			Material Specific Manufacturing		Tolerances Assessment			Tolerances Validation
	CAD_Part	Dimension Name (CAD expression)	Nominal Value	Lower Limit Tolerances	Upper Limit Tolerances	Cent. Dimension	Plus/Minus Tolerances		Tolerances and Clearances OK
1	Ballast	InsideDimBushing	6	?	?	?	?		?
2	BallastInsert	BallastInsertDia	6.5	?	?	?	?		?
3	CircularPlate_BaseTube	PipeDia	6	?	?	?	?		?
4	**Clearance:** Ballast-BallastInsert					?	?	WC	?
5							?	RSS	?
6	**Clearance:** BallastInsert_CircularPlate-BaseTube					?	?	WC	?
7							?	RSS	?

The following sub-sections will restate and develop all activities required to transform the overall approach exposed in this section into a system-level computational implementation. All the introduced commands and functionality have been developed exclusively for this dissertation.

6.7. Structural Decomposition: Meta-modeling CAD geometry into SysML

A meta-model is a detailed classification of the constructs and rules required for creating semantic models, which means the implementation of specific independent descriptions of the underlying algorithmic ideas [147]. A SysML profile can represent a meta-model as an ontological structure. The profile will specify a vocabulary of concepts of the original specific domain as stereotypes. It will also order them in relation to each

other by means of formal rules and add specific properties to the concepts (e.g. metrics) so that users can perform meaningful analysis and calculations. As Shah et al. [88] explained, these graph representations are called meta because they are themselves models that define the languages in which the models of the views are described. There are a few approaches to describe meta-models, with the most general based on the UML language. OMG has also defined more specialized representations such as the Meta-Object Facility (MOF), which, according to OMG (2006), specifies an approach to define, manipulate, and integrate metadata and data in a platform-independent way. In the same sense, the Common Warehouse M?(CWM) and the Information Resource Dictionary Systems (IRDS) are examples of meta-modeling languages.

Although Siemens NX is not a traditional BIM tool, it is a well-known parametric solid modeler for aerospace and mechanical engineering. Siemens NX has very robust feature recognition and feature learning capabilities. These capabilities are important for automating tool setup and process allocation, which is one of the objectives of the implementation.

Because of the highly complex and heterogeneous body of knowledge that can be represented, Siemens NX has an extremely fine grained meta-model. Consequently, an approach that automatically converts the meta-model of Siemens NX into a SysML profile through a model-to-model transformation does not seem appropriate without an important meta-model simplification. This is one of the bases of this project – model integration between SysML and Siemens NX simply transforms what is reflected in the profile. However, the meta-model simplification leaves room for full extensibility through more wide-ranging or domain-specific profiles. For example, a Siemens NX

profile for FEA would be different than a Siemens NX profile for manufacturing processes optimization. As Marchenko [91] proposed, models help to understand the nature of a design method by ignoring some of the not-so-important details. Thus, when modeling a design process, determining the proper level of abstraction is fundamental for the model to be beneficial to its users. For this project, the SysML profile created to accomplish the model integration is a simplification of the meta-model of Siemens NX. Although extensive, the NXProfile contains only the basic elements of the feature-based CAD representation. In a very general view, these elements are assemblies, parts, features, and parameters (Figure 31).

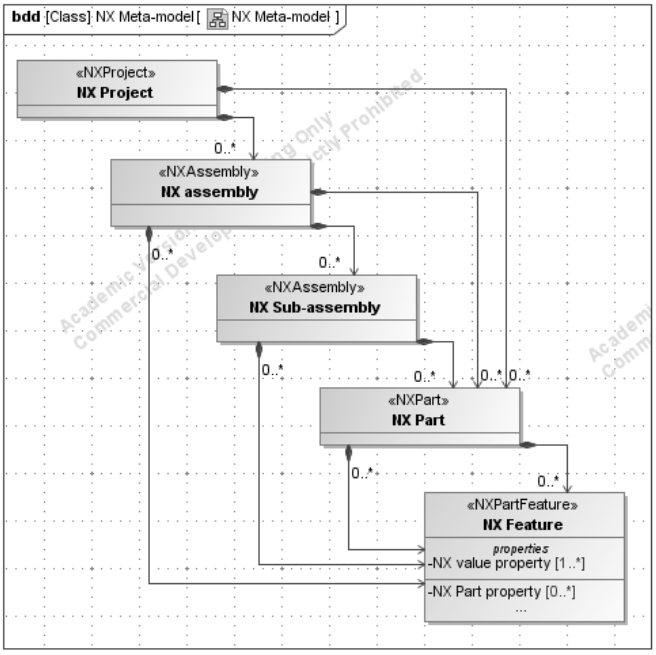


Figure 31. Basic hierarchy of modeling elements of a conventional solid modeler.

As previously stated, Siemens NX is a multi-task CAD package that manages more than just geometric data. Rather, Siemens NX can perform several other tasks during the life cycle of a product model. For example, it can perform feature-based design (e.g. sheet metal), stress and finite element analysis, kinematics simulations,

Computational Fluid Dynamics (CFD), and Numerical Control (NC) manufacturing iterations. For each one of these activities, it is possible to create specific meta-models that can be converted into SysML profiles. One key SysML profile stereotype created during this project to embed this Siemens NX information into a SysML model is the <<NXPartFeature>>. In the context of this dissertation, a feature is the minimal information required to represent geometric variation and tolerances. There are several sub-categories of features based on feature-to-feature relationships. These are lower-level features (e.g. the basic topological entities, faces, edges, and vertices) and higher-level features which are the combination of the lower-level features (or the combination of other higher-level features) having certain relationships among them (e.g., a hole is different than a cutout). In construction, this separation between higher and lower features is crucial to achieve model simplification (filtering) without creating inaccurate results.

Several specializations of the <<NXPartFeature>> stereotype have been created to successfully integrate datum coordinate systems, extrusions, geometric Boolean operations, NX sketches, and numerous other CAD elements as SysML entities. The following graph shows a reduced example of the basic elements of the NXProfile created for this project, which will be further explained in more detail. The white triangle associations represent hierarchical generalizations where the highest-level element within SysML correspond to the block class. Every node of the profile represents a specific stereotype. A stereotype is a kind of extensibility instrument of SysML. Stereotypes can be understood as object-oriented classes, which are used to extend the language of

SysML with the aim of creating new model elements from existing ones, with detailed attributes suitable for domain-specific applications.

Even considering that parameters could exist in any kind of element in a system model, for this implementation the imported CAD parameters will be assumed to be dimensional value properties of parts or assemblies. NX has just one file type (.prt) to describe parts and assemblies. For that reason, it is important to find an approach to represent parts and assemblies independently. To do so, special information has been added to the different stereotypes of the NXProfile. The most important additions were created in the specification of <<NXPart>>, <<NXPartProperty>>, and <<NXAssembly>> stereotypes. They are: currentPartPath, directory, and uniqueID. CurrentPartPath and directory show the location of the last updated file, and uniqueID contains the global unique identifier of the file generated within NX. This data is crucial to keep SysML and NX elements synchronized, even if the file names change.

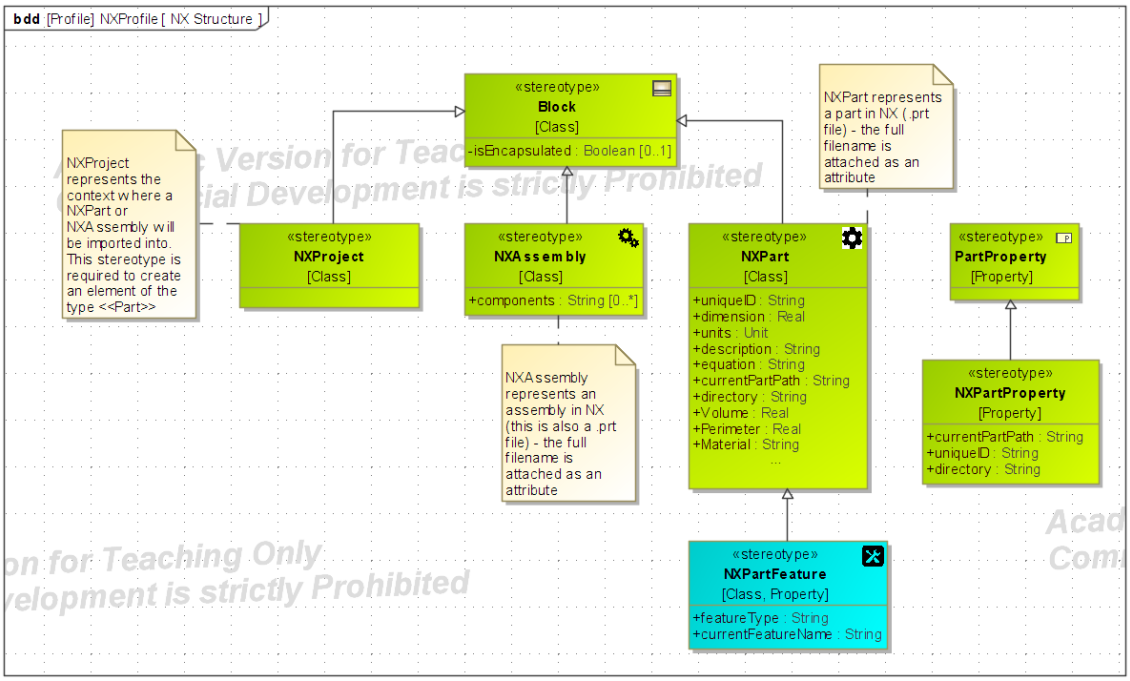


Figure 32: High level meta-model of the CAD data structure

Figure 32 shows the high-level description of the basic elements of the CAD package Siemens NX data structure as explained in Figure 31. However, in order to apply material-specific manufacturing knowledge, we need to further specialize these elements with detailed information about specific features of the manufacturing field that we are trying to represent. Figure 33 shows an extension of the Siemens NX meta-model to describe specific features of the sheet metal domain, which corresponds to one of the material systems used as a case study for the implementation developed in this dissertation. As depicted in the picture, the top level of the DSL corresponds to the <<stereotype>> element, which is a meta-model element before being instantiated. This element is called NXPartFeature and represents any CAD feature that is imported from Siemens NX. The problem of how to automatically apply a domain-specific knowledge to a feature with this high level of generality then arises. To address this issue, new subtype elements have been produced, called specializations, on which we can create custom fields of information that will generate the proper context to automatically apply the required knowledge for a meaningful manufacturing analysis. In Figure 33 a family of features are created under the <<NXSheetMetalFeatures>> element. These elements inherit all properties, visible and invisible, of their super-type, as also include other properties that define the specifics of every feature. For example, a basic feature of the sheet metal domain is flange, which is represented in the NX Sheet Metal Features diagram as NX SM Flange. This element, which is a specialization of the NXSheetMetalFeatures, contains all the parameters that define a flange: BendRadius, NeutralFactor, BendAngle, BendReliefWidth, BendReliefDepth, and others.Any time that a feature is typed as flange in a NX model that is being imported into a system

model, this stereotype will obtain the values of such a feature and populate its fields to create a flange instance.

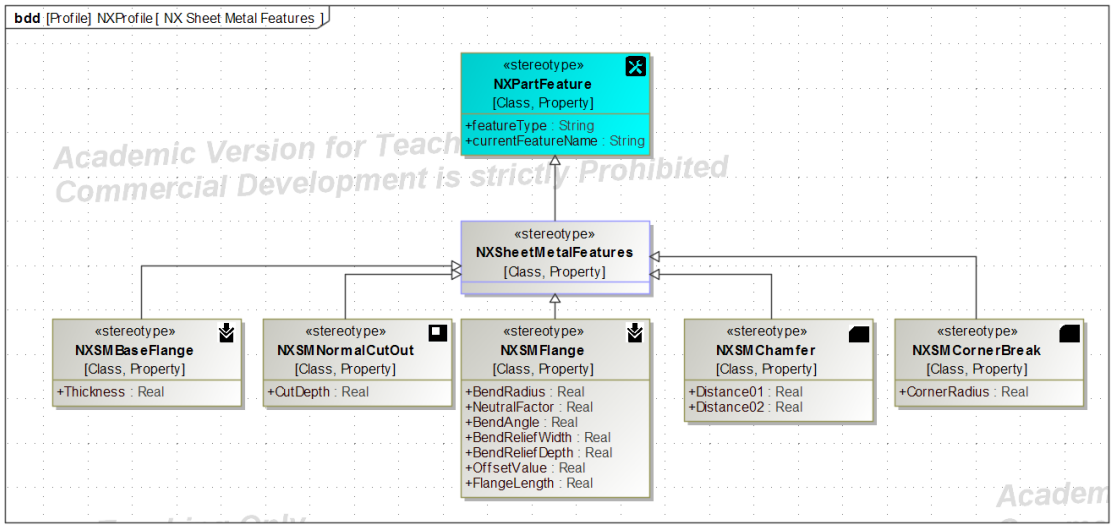


Figure 33: Extended NXSheetMetal meta-classes using stereotypes that carry domain specific properties and constraints

Figure 34 shows an imported sheet metal element in the system model environment (SysML). As can be seen in the diagram, all elements have been automatically allocated stereotypes from the sheet metal domain, and their parameter fields have been instantiated with numeric values. Also, custom icons were created for all new elements of the developed NXProfile (Figure 35). The black diamond association between components is called composition association and it defines a structural relationship between parents and children. This capability of creating a topological hierarchy of features from a CAD component is not naturally present in the NX

package[11], and is an added functionality of this implementation to replicate the real structural topology of a material system. For example, as depicted in the diagram, the .prt level inner lower chord (file level) is a the top of the hierarchical structure. This chord has a child typed as base flange, which is the basic element for constructing a sheet metal component. This base flange has two children typed as <<NXSMFlange>> named Side 1 and Side 2. In a fourth level of hierarchy, Flange 1 has a child feature typed as <<NXSMNormalCutout>> named SideNestedHoles. Using this example, we can establish a basic domain-specific hierarchy for sheet metal fabrication that is compliant with the real processes of the field. That is, it cannot be a flange without a base flange, or it cannot be a hole without a base flange or a flange. In this manner it is ensured that the structural decomposition does not defy the basic rules of solid modeling with respect to consistency with the built environment.

---

[11] As it is shown in Figure 35 the expressions list in NX does not introduce any kind of indentation or other structure that describes the hierarchy of features.

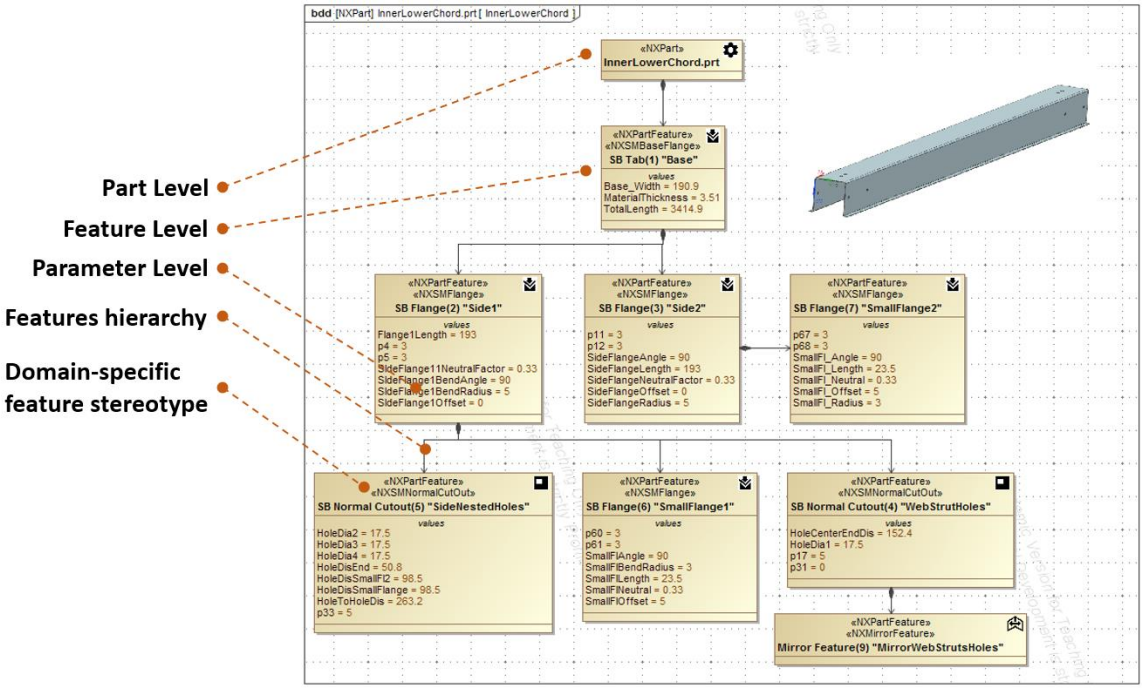


Figure 34: Elements hierarchy of a sheet metal component imported into a system model

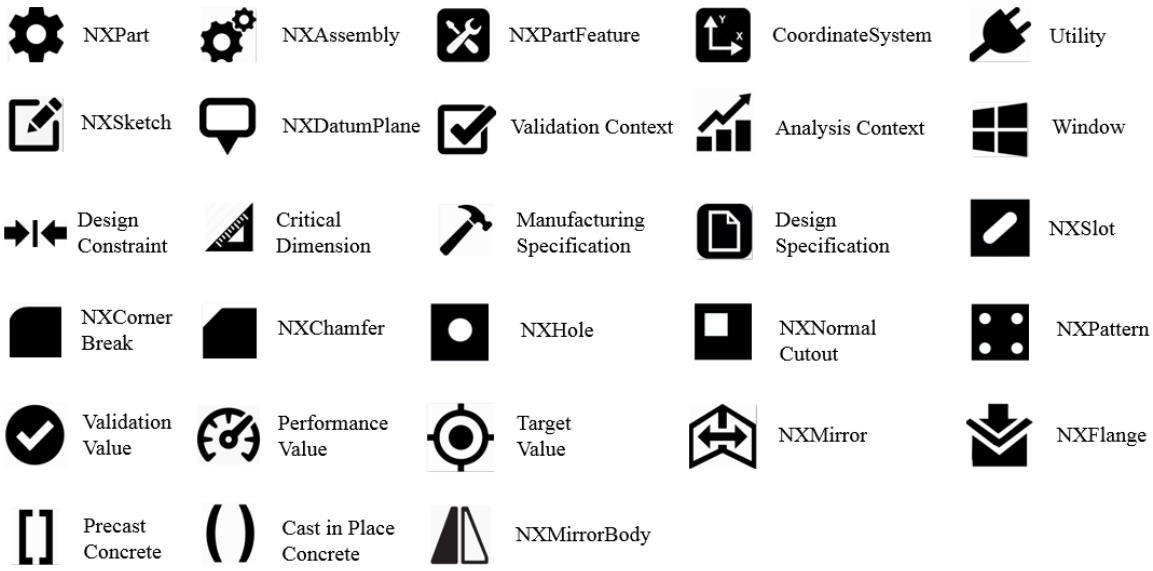


Figure 35: Custom Icons legend of the implementation

The following section presents the main commands and functionalities developed for this implementation.

Import CAD Model:

This command creates a SysML model from a CAD model. When importing a single file, it creates a folder with the name of the NX file. This folder contains two elements, which are a block structure of the NX file and an empty folder to store instances from further parametric executions. However, when importing an assembly, the process is a little different because blocks (classes) cannot contain any packages as children. Then, in the assembly importing procedure, we have added just one folder at the top of the structure tree. All sub components (Also of <<NXPart>> stereotype) will be incorporated inside the same package.

The following steps detail the procedure to import the CAD model into the SysML model:

1. Right click on the folder where the CAD model will be imported;
2. Go to the command "Import CAD Model"[12] as shown in Figure 37;
3. Select a CAD file to be imported on the SysML model; and
4. Repeat for all different CAD files that will be part of the parametric execution[13].

At this point, all CAD components will be decomposed as feature trees in the SysML environment. The outcome of the import command will be a SysML instance of

---

[12] As an alternative, the command "Import CAD Model with Feature Type Filter" performs the same action. However, the model will be filtered before imported.

[13] Upon completion of a multiple System Integration with several domain specific tools, these files could come from different CAD packages

the CAD model that will include all the geometric data based on the data structure of the NXProfile SysML DSL. Figure 36 shows the outcome of a successful import in the containment tree of the MagicDraw environment. The highlighted CAD component has been automatically created in the MagicDraw modeling environment as a SysML model, which can be later dragged into SysML diagrams to graphically access specific design parameters or for reporting activities.

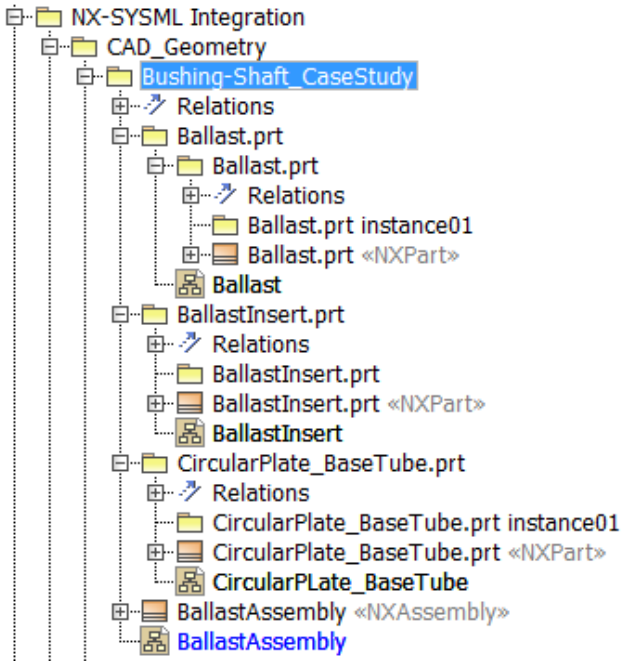


Figure 36: Containment tree with imported CAD geometry

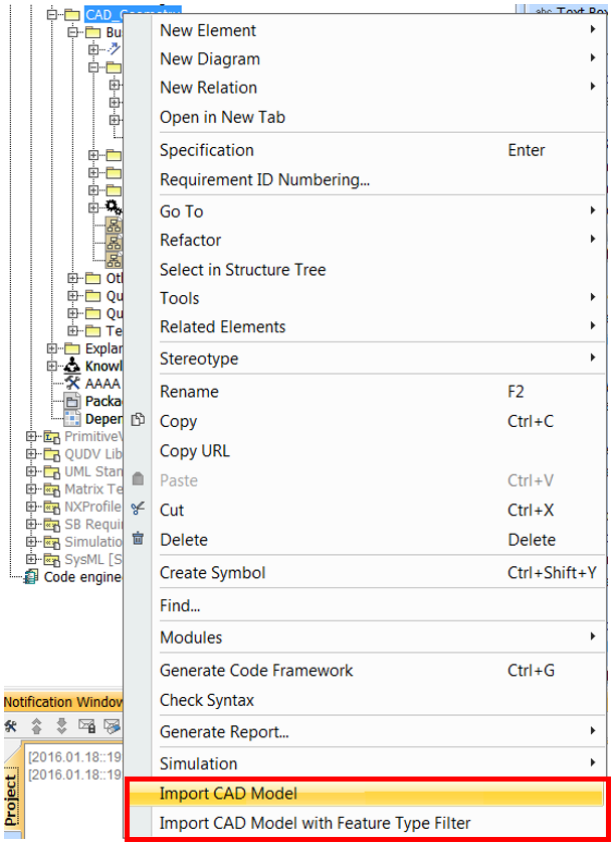


Figure 37: The two importing commands of the created application are highlighted in the red square

Import CAD Model with Feature Type Filter:

A specific subset of CAD data that is required to run a domain-specific analysis is called “View.” In order to create Views, we need to filter the data that we obtain from the CAD model. To do so, we have created the stereotypes filter. The stereotype filter creates a SysML model from CAD the same way that importing a full model would, creating a folder with the name of the NX file, which contains a block structure of the NX file. However, it offers a check box window to specify what feature types need to be imported and what feature types will not be included in the view. In the case that we are importing NX assemblies, the import operation will create just one folder at the top level (assembly level). Considering that the complexity of building models is very high, this is

a critical capability of the system. For example, in Figure 38, the datum planes have been deselected, as they are not required at this time for manufacturing compliance analysis.

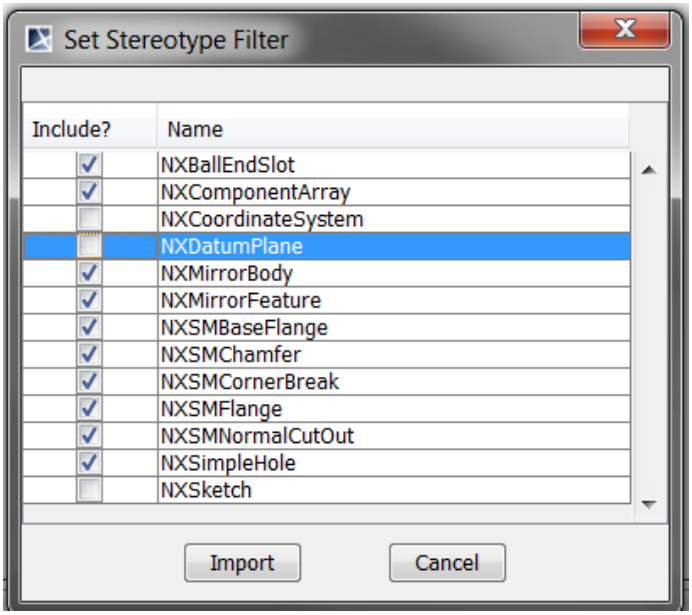


Figure 38: Stereotype Filter to manage Model Granularity

The following is the procedure to import the CAD model with the stereotype filter into the SysML model:

1. Right click on a SysML folder (stereotype <<package>>);
2. Click on “Import CAD model with Feature Type Filter;
3. The Feature Filter will show up in the screen as shown in Figure 38;
4. Uncheck the boxes in front of the features you are not importing and click “Import;” and,
5. The model will be imported into the SysML environment. This model will have a hierarchy indentation, and if some feature was “filtered,” the remaining block will be attached to the element in the next level up in the hierarchy.

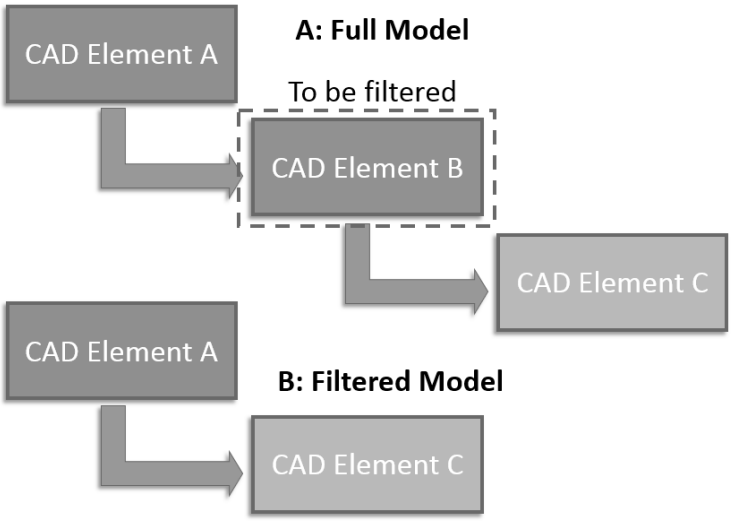


Figure 39: Filtered versus full model hierarchy structure

A feature filter will offer the option of controlling the level of granularity of the model (Figure 39). The filter will automatically and dynamically[14] create a list of all the stereotypes present in the NXProfile and will offer the user a checkbox for each of them. Thus, the user can check just the stereotypes needed for the specific modeling task. This capability is meant to control levels of granularity of data-rich building models. However, no matter how "coarse" the SysML representation of the CAD model is, after filtering, the CAD and SysML models will always be consistent. Figure 40 illustrates a model where only the file level hierarchy (<<NXPart>> or <<NXAssembly>>) have been imported, and all the features itemization has been filtered. Figure 41 shows the same

[14] Every time that the features filter is requested, it will query the NXProfile to see if new stereotypes have been added or deleted.

fully unfiltered model where all CAD data has been imported into the system model environment.

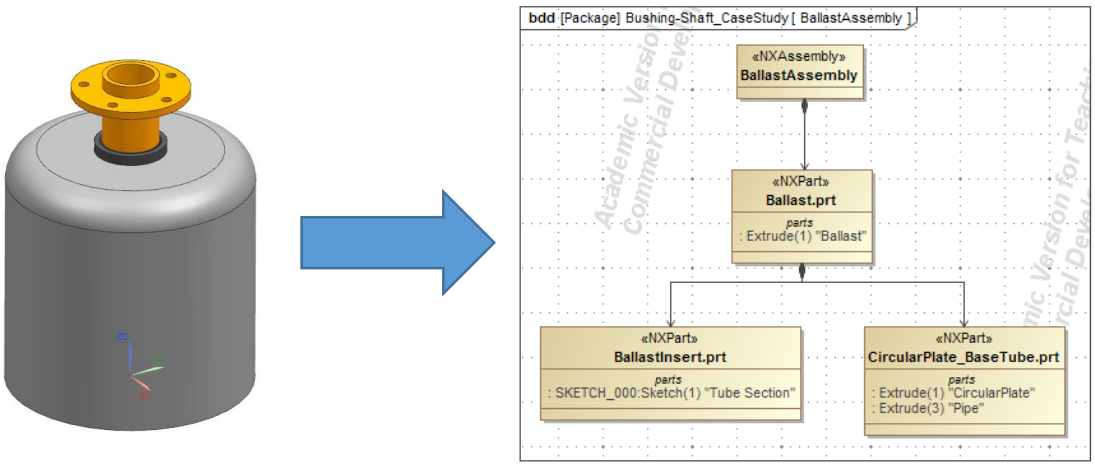


Figure 40: Ballast assembly: CAD representation (left) and component level SysML representation (right) after model-to-model transformation.

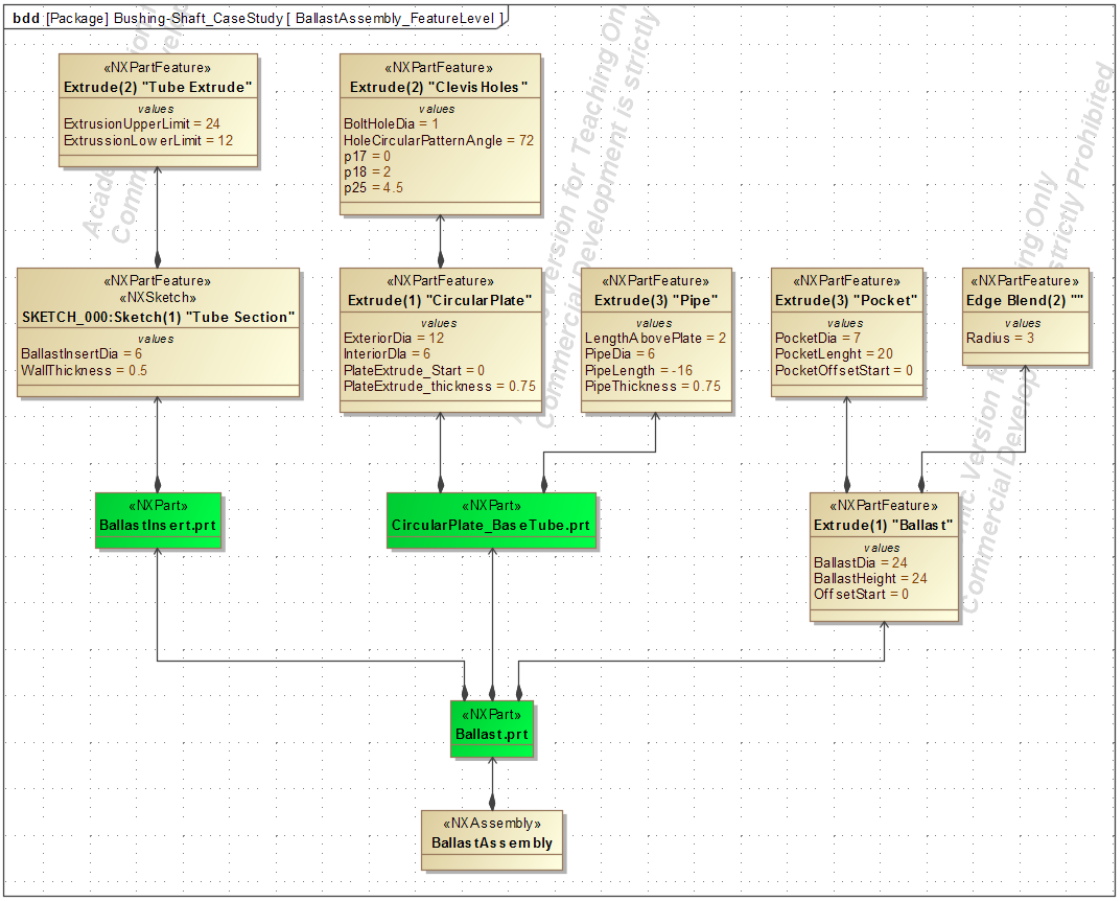


Figure 41: Model-to-model transformation output: full CAD structure

Link CAD Model to Existing SysML Model:

The application developed in this dissertation allows users to link existing SysML elements to NX files by using the "Link NX file" command. The linked elements could be single components or even assemblies. To perform this task, the user must right click the SysML element that will be linked and choose the proper command as shown in Figure 42. This command has been created because usually building projects start from a description of requirements prior to a geometric instantiation.

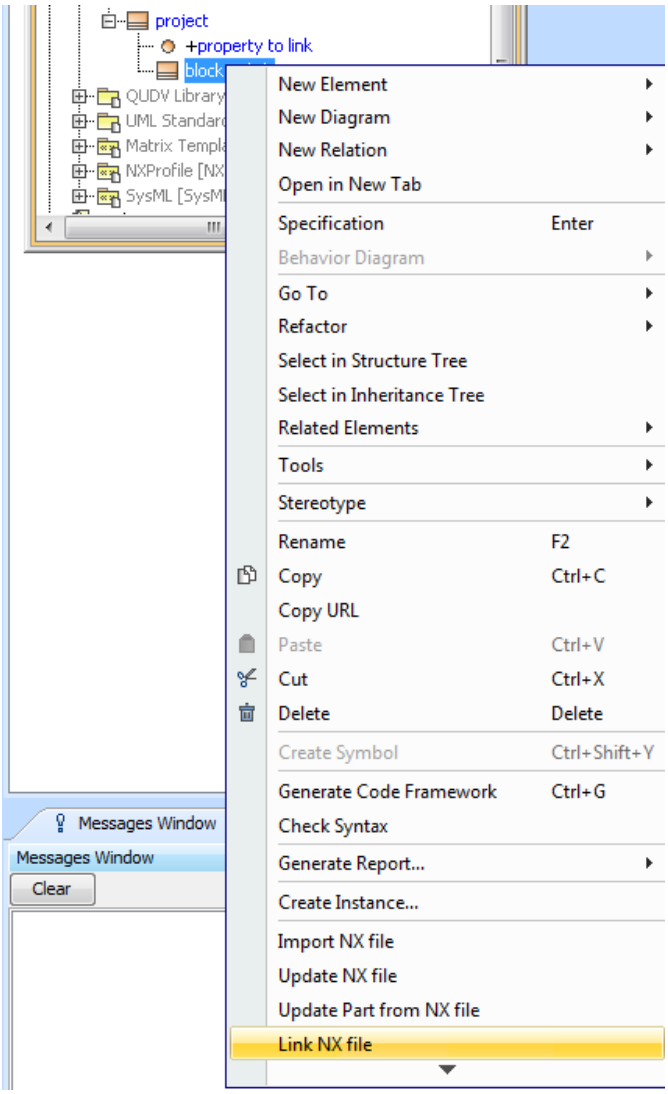


Figure 42: Link NX file command

When linking an existing NX file with an existing SysML modeling element (a <<NXPart>> or a <<NXAssembly>>), it is likely the two elements will differ in name. In this case the application will prompt the user with a window to pick the name that the user wants to keep as the name of the file and the SysML element as shown in Figure 43.

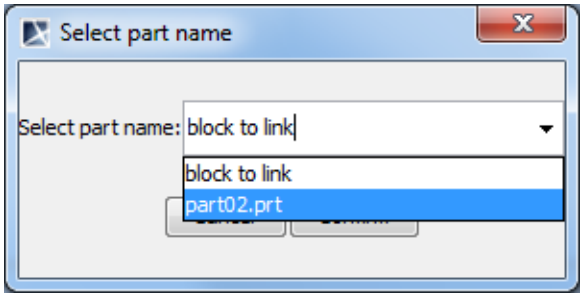


Figure 43: Name disambiguation while linking a NX file with a SysML element

Also, if the user tries to link an NX file within an <<NXProject>> or a <<NXPart>> element, and the NX file to import already exists in the project context, the implementation will prompt the user with an error as shown in Figure 44.

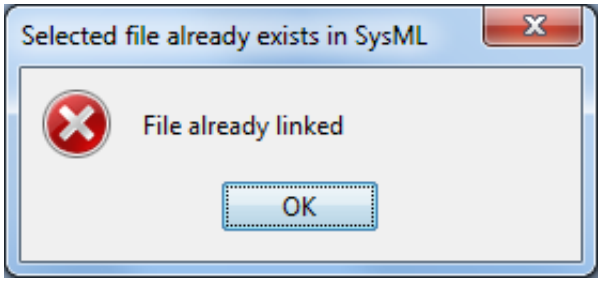


Figure 44: File already linked error

6.8. Knowledge Acquisition

This important stage of the manufacturing compliance method includes the acquisition and formalization of domain-specific knowledge necessary to execute parametric analysis on imported CAD models in the SysML environment. This stage is one of the only processes of this implementation that is not developed in an automated fashion. The reason is that this stage requires the user to build pieces of knowledge,

encapsulated in SysML requirements and specification, directly from construction standards, material systems reports, books, or other kinds of written manufacturing know-how, which are not machine-readable. The acquired knowledge will be stored in specifications and constraints. It will be available for reuse by searching within the domain-specific knowledge folder in the SysML Model or by automatically allocating it during the knowledge allocation stage. This folder will have manufacturing requirements that lead to manufacturing specifications represented as mathematical expressions in <<constraintBlock>> elements.

For this implementation the SysML stereotype <<requirement>> has been used and specialized as a text-based knowledge container, and the SysML stereotype <<constraintBlock>> has been used and specialized as a mechanism that ensures that the knowledge is being applied and the CAD geometry is in compliance. Figure 45 shows the portion of the NXProfile and the meta-classes that deal with the knowledge acquisition, knowledge allocation, and parametric execution stages. There are three different kinds of stereotypes necessary for this analysis task: a specification, a constraint, and a repository. These elements respectively come from the meta-classes <<Requirement>>, <<ConstraintBlock>>, and <<ElementsLibrary>>. For this implementation the <<Requirement>> class has been specialized into two stereotypes: <<Design Specification>> and <<Manufacturing Specification>>. Both elements are represented by custom icons for easy readability, and are verified by two different kinds of <<ConstraintBlock>> stereotypes. The design specification will be verified by a <<Knowledge-Based Constraint>> and the manufacturing specification will be verified by a <<Critical Dimension>>. Both verification procedures require an association

element between the requirement and its associated constraint to automatically evaluate whether the CAD geometry is in compliance with the domain rule. With this double stereotype approach, analysis for SDCT and HCT will be kept separated, as they belong to different stages of the overall analysis. Then, an element typed as <<Manufacturing Knowledge>>, which inherits all the properties of an <<ElementsLibrary>> will act as permanent storage for the created knowledge. As explained later in this dissertation, other critical elements such as <<Analysis Context>> will be stored in the same kind of libraries for easy access and allocation.

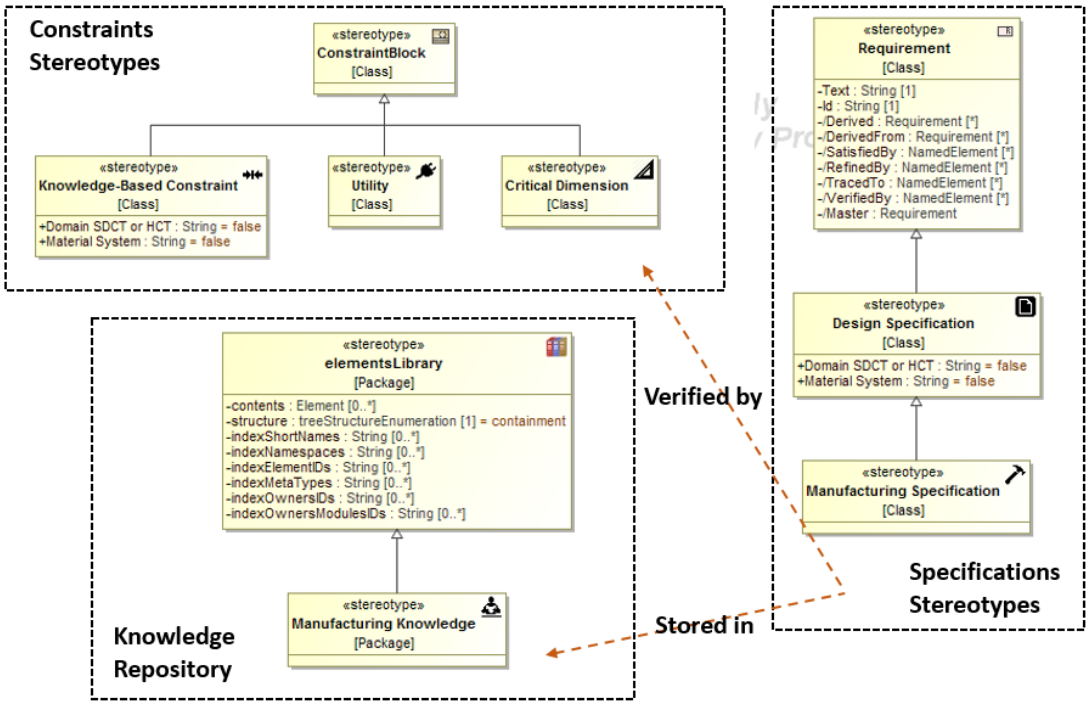


Figure 45: Meta-classes of material-specific knowledge

In the previous example of the cylindrical fit, the following equations represent the Domain-Specific Knowledge (DSK) that the designer needs to be aware of when prescribing clearances for any kind of cylindrical fit based on steel components.

Bushing/Shaft tolerances calculation:

Milling tolerances for SDCT steel components require the specification of Upper Level tolerances (UL) and Lower Level tolerances (LL), which are described in different manufacturing standards [104]. These DSK will be described in SysML as a construction specification by means of <<Requirement>> elements type.

Plus/Minus (PM) tolerance from LL and UL tolerances specification:

Equation 12: plus/minus formula from UL and LL

$$PM_t = \frac{(LL - UL)}{2}$$

Where:

PM = Plus/minus tolerances value

LL = Lower level tolerance specification

UL = Upper level tolerance specification

Centered dimension calculation from LL and UL tolerance specification:

Equation 13: centered dimension formula from UL and LL

$$cT = X + \frac{(LL + UL)}{2}$$

Where:

cT = Centered tolerance dimension

X = Nominal dimension

LL = Lower level tolerance specification

UL = Upper level tolerance specification

Integrating DSK of tolerances and clearances calculations into the implementation:

In the SysML Language, as previously explained, constraints blocks are used to define equations or other logical expressions. As a block, a constraint block is an element of definition—one that defines a Boolean constraint expression (an expression that must evaluate to either true or false) [27]. Most often, the constraint expression defined in a constraint block is an equation or an inequality (a mathematical relationship that is used to constrain value properties of blocks). This is done mainly for two reasons:

- To specify assertions about valid system values in an operational system, and
- To perform engineering analyses during the design stage of the life cycle.

The variables in a constraint expression are called constraint parameters. Generally, they represent quantities, and so they are stereotyped most often by value types. For example, the following figures shows a constraint block (left) named Bushing_Metals_Tolerances, which contains four constraint expressions that will assess a design specification or a design requirement about tolerances for a bushing component. In this case the constraints are grouped in a single constraint block. The notation for a constraint block on a block definition diagram (bdd) is a rectangle with the stereotype <<constraint>> preceding the name. However, as most of the modeling elements in this implementation have been customized to be applied in construction, these constraints blocks will be specialized as <<Critical Dimension>> or <<Knowledge-Based Constraint>> (Figure 46, Figure 47). The constraint expression always appears between curly brackets ({}) in the constraints compartment. The constraint parameters in the constraint expression are listed individually in the parameters compartment [27].

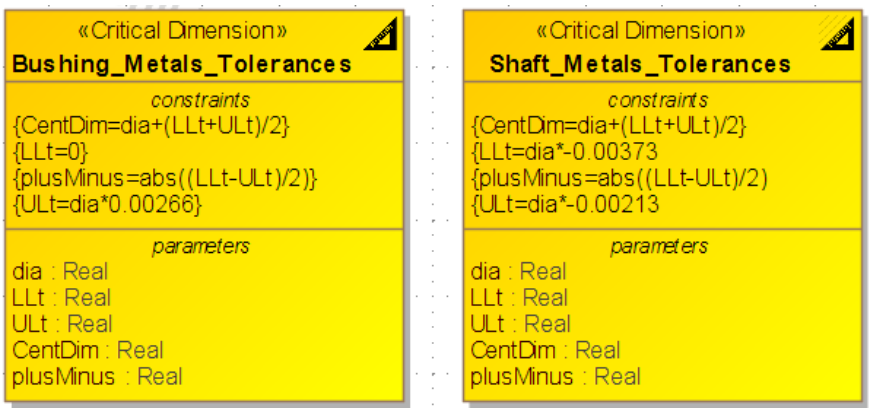


Figure 46: Constraint block of bushing and shaft tolerances specification for steel milling component

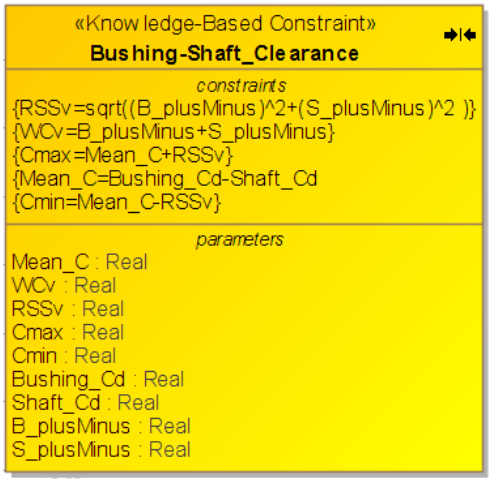


Figure 47: Constraint block that delivers clearance assessments for WC and RSS of a bushing assembly.

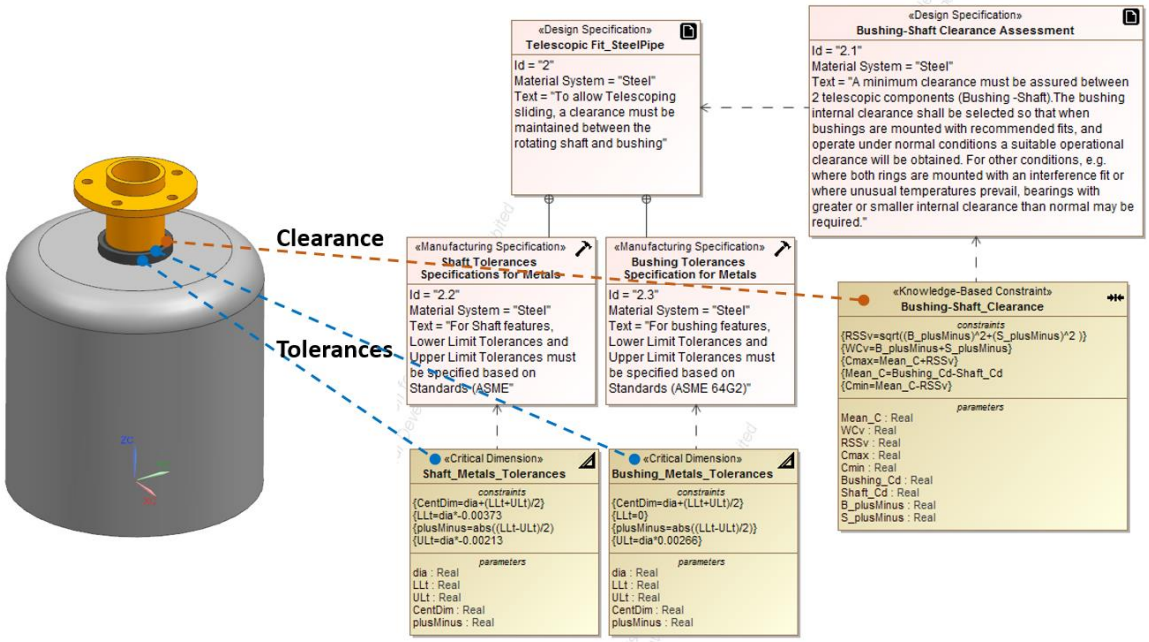


Figure 48: Constraint blocks from design and manufacturing specifications must be allocated to their targeted features

The capability of the constraint block to carry several equations in a single unit allows designers to apply several related calculations and analyses at the same time. For general equations, such as centered dimension equations, the parameters are real numbers. Since this research deals mostly with dimensional values, all of the value types for this implementation are also real. Thus, all values can be connected to parameter ports that are specified as real numbers. However, different value types can be created when the user intent is to simulate or assess the model from a behavioral standpoint.

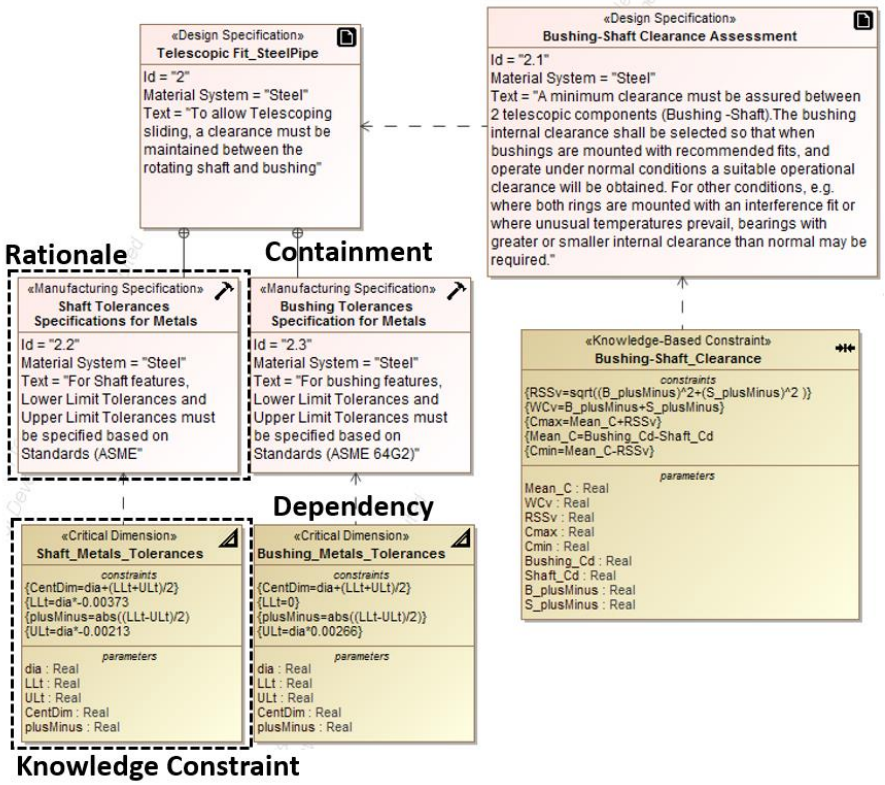


Figure 49: Material-Specific Knowledge: Reusable manufacturing specifications: diagram

In addition to sub-types of constraints such as critical dimensions or knowledge-based constraints, Figure 49 depicts the main elements defined for the material-specific knowledge representation. The manufacturing and design specifications, which are text-based, are intended to convey the rationale of a specific piece of knowledge. In addition to this information, these knowledge modeling elements contain an identification number to be sorted or organized in domain-specific libraries within the system model (Figure 50). Another important element for the topological description of domain-specific knowledge is the association. As shown in Figure 49, for this dissertation, the description of relationships between constraints and specifications has two distinctive kinds of associations: the containment association and the dependency association. The former kind refers to the ability of organizing manufacturing and design specifications

hierarchically. This means there are main specification elements that can have children specification elements. For example, in Figure 49 the specification Bushing-Shaft Clearance assessment has a derived relation to the Telescopic Fit_SteelPipe. Furthermore, this Telescopic Fit_SteelPipe has two containment relationships to the elements Shaft Tolerances and Bushing tolerances. The same indentation approach can be seen in Requirements and Specifications of Figure 50 and in the reusable manufacturing specification in the table presented in Figure 51.

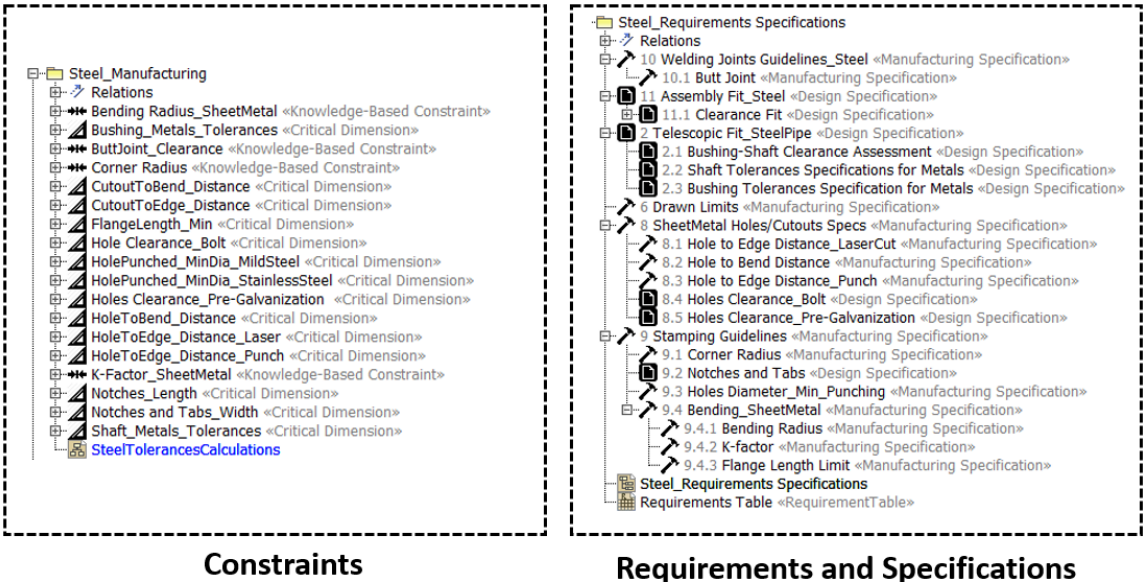


Figure 50: Examples of constraints and specification libraries as they appear in the containment tree of the MagicDraw user interface

#	ID	Name	Text	Requirement Type	Owner
1	2	Telescopic Fit_SteelPipe	To allow Telescoping sliding, a clearance must be maintained between the rotating shaft and bushing	Design Specification...	Steel_Requ...
2	2.1	Bushing-Shaft Clearanc...	A minimum clearance must be assured between 2 telescopic components (Bushing -Shaft).The bushing internal clearance shall be selected so that when bushings are mounted with recommended fits, and operate under normal conditions a suitable operational clearance will be obtained. For other conditions, e.g. where both rings are mounted with an interference fit or where unusual temperatures prevail, bearings with greater or smaller internal clearance than normal may be required.	Design Specification...	2 Telescopi...
3	2.2	Shaft Tolerances Specif...	For Shaft features, Lower Limit Tolerances and Upper Limit Tolerances must be specified based on Standards (ASME	Design Specification...	2 Telescopi...
4	2.3	Bushing Tolerances Spe...	For bushing features, Lower Limit Tolerances and Upper Limit Tolerances must be specified based on Standards (ASME 64G2)	Design Specification...	2 Telescopi...
5	6	Drawn Limits		Manufacturing Spec...	Steel_Requ...
6	8	SheetMetal Holes/Cutouts Specs	Follow these manufacturing guidelines for holes in sheetMetal applications. Generally, dimensioning should be done from a feature to an edge. Avoid feature-to-feature dimensions over two or more planes. Feature-to-bend dimensions may require special fixtures or gaging	Manufacturing Spec...	Steel_Requ...
7	8.1	Hole to Edge Distance_LaserCut	A good rule of thumb for hole placement is to keep the hole at least one material thickness away from any edge. If the hole gets too close to an edge a bulge can form . Also note, if the hole is used for fastening two pieces together, extra web should be used to account for the added stress.	Manufacturing Spec...	8 SheetMet...
8	8.2	Hole to Bend Distance	Any cutout features, located too close to a bend may be distorted. The distance of such features from the bend should be equal to at least 3 times the sheet thickness plus the bending radius.	Manufacturing Spec...	8 SheetMet...
9	8.3	Hole to Edge Distance_...	A good rule of thumb for hole placement is to keep the hole at least 3 material thickness away from any edge. If the hole gets too close to an edge a bulge can form . Also note, if the hole is used for fastening two pieces together, extra web should be used to account for the added stress.	Manufacturing Spec...	8 SheetMet...
10	8.4	Holes Clearance_Bolt		Design Specification...	8 SheetMet...
11	8.5	Holes Clearance_Pre-G...	Typical coating thickness on bolts can range from 0.045 to 0.09 mm, which can make standard bolt and nut tolerances difficult to maintain for correct assembly. If bolts are galvanized, then the nuts should be oversized to accommodate the 0.09 to 0.18 mm increase in bolt diameter after galvanizing.	Design Specification...	8 SheetMet...
12	9	Stamping Guidelines	Basic Considerations for Stamping SheetMetal parts	Manufacturing Spec...	Steel_Requ...
13	9.1	Corner Radius	Corners May be sharp, however to reduce tooling costs, specify radii of ½ material thickness or a minimum of .015".	Manufacturing Spec...	9 Stamping...
14	9.2	Notches and Tabs	should not be narrower than 1.5X the material thickness. Length of notches can be up to 5X length of material thickness.	Design Specification...	9 Stamping...
15	9.3	Holes Diameter_Min_Pu...	-Minimum diameter of holes should be equal or greater than 1.2 X material thickness, and 2X material thickness for stainless steel or high tensile materials.	Manufacturing Spec...	9 Stamping...
16	9.4	Bending_SheetMetal	All the bend features of sheet metal parts need to be compliant with bending allowance specifications. The length of the neutral axis between the bend lines, or in other words, the arc length of the bend. The bend allowance added to the flange lengths is equal to the total flat length. This specification is applied ONLY to a flat pattern when the allowance has not been applied during modeling.	Manufacturing Spec...	9 Stamping...
17	9.4.1	Bending Radius	The radius of a bend in a piece of material that occurs between the bend lines. The radius is measured from the bend axis to the inside surface of the material and is therefore sometimes specified as the inside bend radius. The bend radius is typically equal to at least the sheet thickness.	Manufacturing Spec...	9.4 Bendin...

Figure 51: Reusable manufacturing specifications: table version

6.9. Knowledge Allocation

As explained in the previous section of this document, manufacturing specifications and design specifications are cumulative, verifiable, reusable, and they are stored in domain-specific knowledge libraries within the SysML modeling environment. As shown in Figure 52, the knowledge allocation within parametric diagrams will allow the user to link the formal representation of manufacturing knowledge (geometric constraints as equations) with numeric values obtained from the imported CAD model. The expressions created for such an association will be solved by the mathematical

engine. Then, the newly calculated numeric values will be reallocated in the geometric features parameters to create a manufacturing knowledge-compliant geometry update.

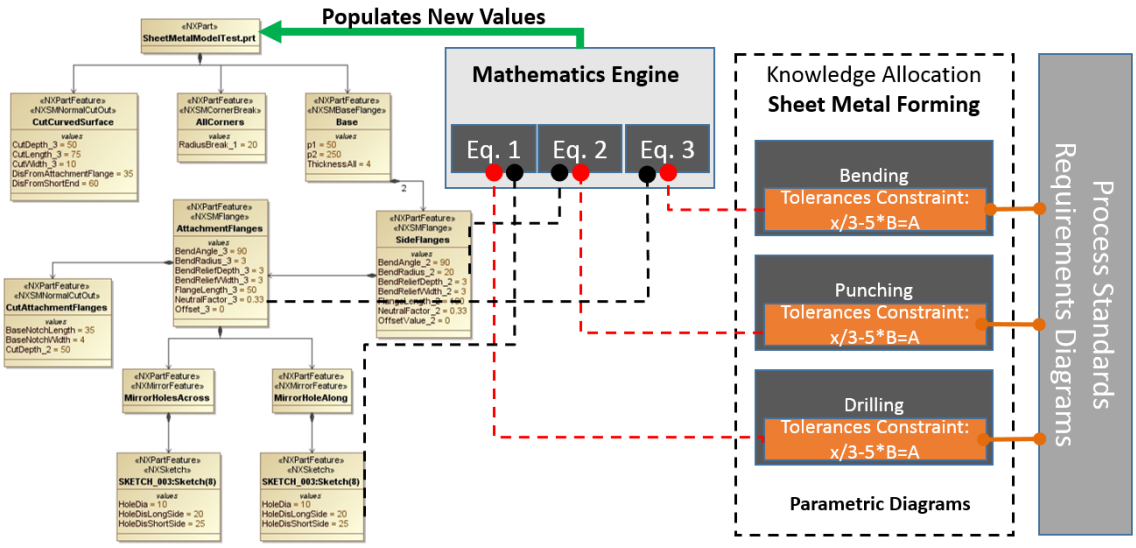


Figure 52: General description of the integration between processes standards and CAD features through a mathematical engine

Based on the features stereotypes held by the imported CAD geometry, the application will recommend, through a dependency matrix, the manufacturing specifications or design specifications that the user should include to assess the manufacturability of the intended part or building assembly. This activity is performed by looking at the material or feature type of the imported CAD component. However, critical parameters of each feature must be identified by the user. For instance, in a sheet metal component like the one shown in Figure 53, a manufacturing specification that assesses a Flange Length Limit has been automatically suggested by the developed application. However, this allocation matrix did not specify which of the flange Side 1 parameters must be linked with the manufacturing constraint that verifies the manufacturing specification. As stated in most of the literature related to Systems Engineering (SE) and Geometric Dimensioning & Tolerancing (GD&T), tolerances must

be designed, and that takes time. The value of applying a modeling approach as the one proposed in this dissertation comes when parametric iterations are desired, when conflicting system interactions need to be identified, or when an analysis context for repetitive procedures has been stored for reusability. The first step will be the identification, based on features stereotypes of the profile, of the critical dimensions that must be analyzed for tolerances allocation and manufacturability. Critical dimensions are at the <<NXValueProperty>> level of the <<NXProfile>> and these dimensions carry parametric information seamlessly coordinated with the CAD model. The following dependency matrix depicts the result of a knowledge allocation procedure. In Figure 53, the small arrow dependency icon refers to a "verified by" relationship. For example "Side 1" is verified by a bending radius design specification (hammer icon).

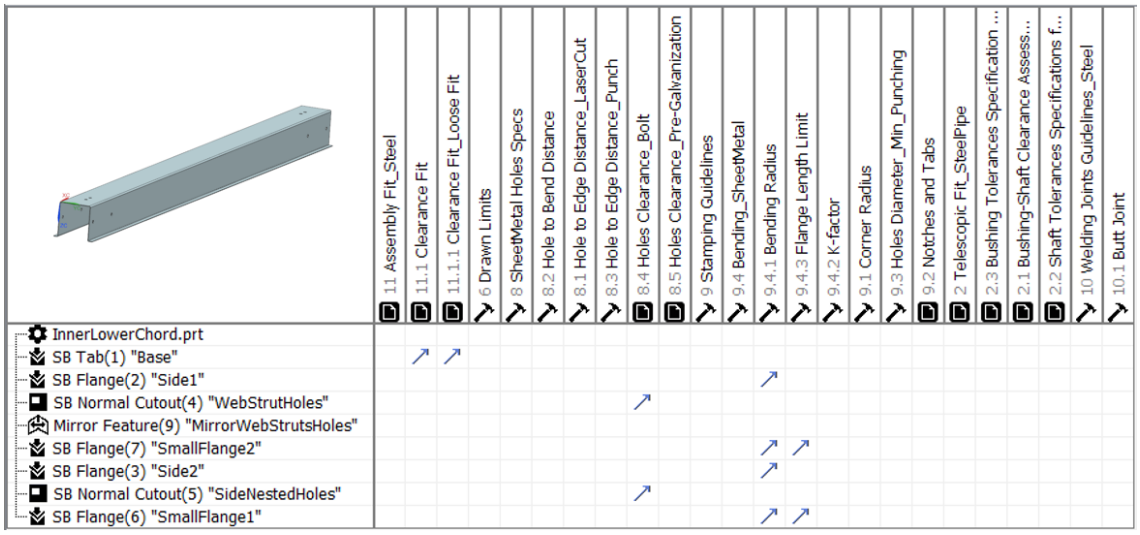


Figure 53: Dependency matrix for knowledge allocation

When allocated, manufacturing specifications or design specifications are displayed in-context in the feature level diagram of a block definition diagram (bdd). These specifications require an <<Allocation>> type of association for traceability and verification. This capability offers the option of visually assessing all modeling features

to confirm that a requirement is fulfilled, or to verify that a design or manufacturing specification is met. Figure 55 shows a block definition diagram where the light grey elements are manufacturing or design specifications and the light brown elements are the feature-based decomposition of the green element “LowerChordStiffener.prt.” In this knowledge allocation diagram, all specifications have been reduced to four fields of information, and other elements such as text-based rationale or constraints have been hidden. Figure 55 depicts the manufacturing specification from top to bottom, including a stereotype <<Manufacturing specification>> with the associated icon, a specification named “Hole to Bend Distance,” an identification field for the specification (Id), and the material system where the specification has been taken from.

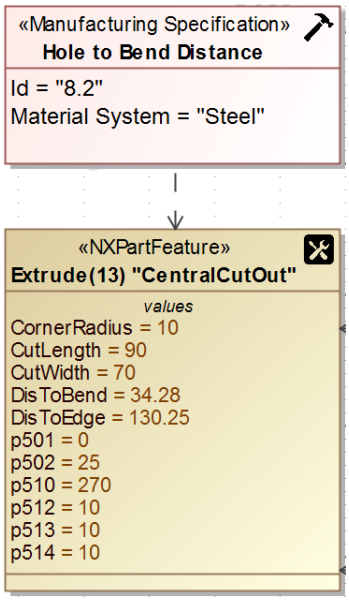


Figure 54: In-context manufacturing specifications allocation through a dependency association

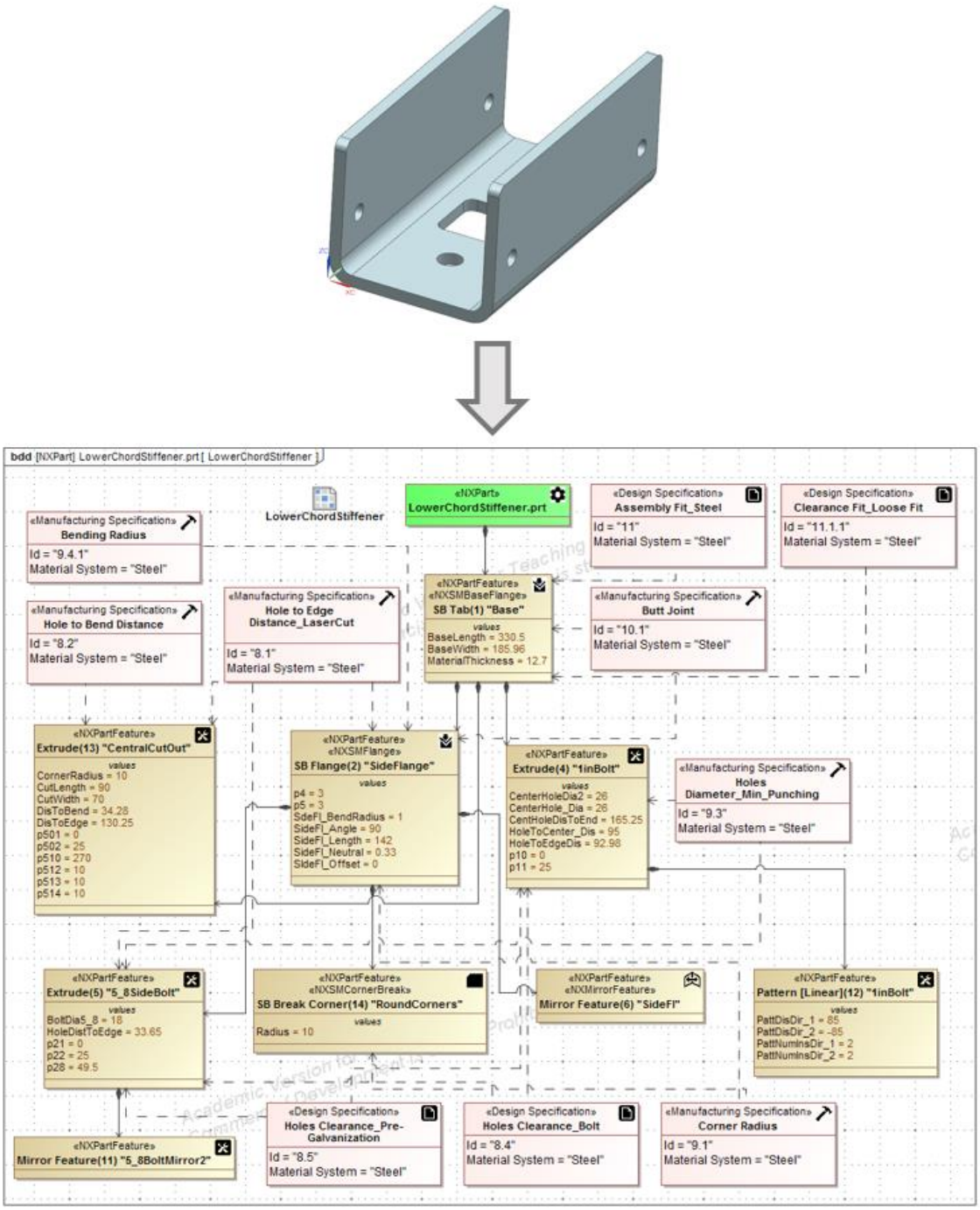


Figure 55: In-context manufacturing and design specifications

When domain-specific knowledge has been properly allocated to a feature-based decomposition of a CAD component or assembly, the model is ready to be executed in a

parametric diagram by means of an <<Analysis Context>> element, which will be explained in the following section.

6.10. Parametric Execution

Parametric models limit the properties of a system. The parametric engine of SysML, which for this dissertation is powered by Maple 18, enables the mathematical evaluation of a system model and uses profiles as a base meta-language to instantiate all required elements. Also, constraints are conveyed as equations or logical expressions, and the parameters of the equations are linked to the properties of the system being evaluated [27]. Furthermore, all parametric models capture the description of one or more engineering views of a design. As explained in [27], a parametric model which captures multiple engineering views such as the ones created in this implementation — performance, validation, or target values—can be used to calculate several design alternatives, to support trade-off analysis, or optimize a design based on multiple criteria. Accordingly, for the present dissertation, the main use of parametric diagrams will be the development of analysis contexts where domain-specific knowledge will assess the manufacturability of parts and assemblies.

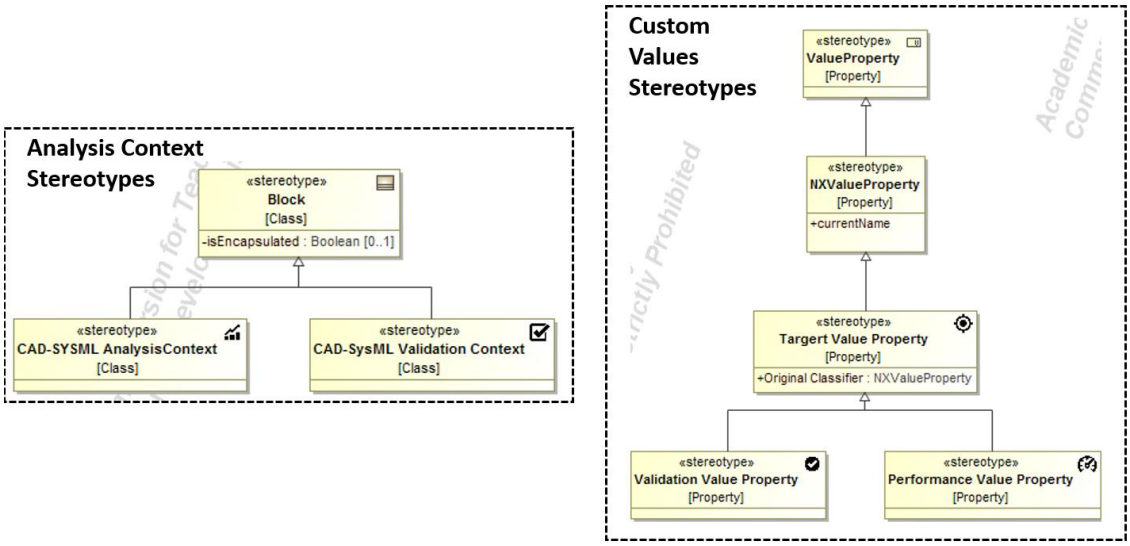


Figure 56: Main stereotypes developed for parametric execution of manufacturing knowledge and tolerances evaluation in a SysML model profile

The parametric execution of this implementation is only possible when started from a parametric diagram (par) (Figure 57), which contains SysML blocks <<block>> that carry information from the CAD model, and SysML constraint blocks <<constraint>> that carry domain knowledge represented as mathematical or logical expressions. Both element types will be contained in properties of a third kind of element, the <<CAD-SysML Analysis Context>> or <<CAD-SysML Validation Context>>. The analysis context stereotypes are specializations of SysML blocks that are used to create system boundaries defining where to execute a domain-specific evaluation. Both kinds (the <<CAD-SysML AnalysisContext>> and the <<CAD-SysML ValidationContext>>) can represent any of the custom values stereotypes as depicted in Figure 56. These custom values stereotypes are:

NXValueProperty: Original value property developed in this implementation to represent any numeric parameter of an imported CAD feature.

Target Value Property: Sub-type of the NXValueProperty, used as a stereotype of an outcome for a parametric calculation. Target value properties will update the value of its custom property "Original Classifier," which is a parameter directly imported during the structural decomposition stages. Target value properties are critical for this implementation, as they will finally upgrade the CAD geometry for manufacturing compliance. Target values will be covered in more detail in the specifications verification stage.

Validation Value Property: Elements typed as Boolean will verify that the CAD geometry has met the manufacturing and tolerances specifications. This stereotype has been specifically created to assist in decision making and system evaluation during the specification verification stage.

Performance Value Property: These elements are numerical outcomes of parametric calculations that do not come from a specific CAD parameter. Rather, these elements represent the instantiation of domain-specific knowledge required to assess manufacturability. For example, in Figure 57, the three green elements at the right, which are parameters of the analysis context, are metrics used to verify the status of a bushing-shaft clearance. These elements do not directly belong to any imported feature, but they assess a mating condition (clearance) between two different CAD components.

Reusability of Analysis Contexts

Figure 57 presents an example of a <<CAD-SysML Analysis Context>> template for a bushing-shaft evaluation in a SDCT environment. The element to the right, a <<Knowledge-Based Constraint>> called Bushing-Shaft Clearance, calculates several performance value properties of the required assembly clearance, such as minimum and

maximum conditions, mean values, and also RSS and WC tolerances assessment. This knowledge-based constraint derives its input values from two <<Critical Dimension>> elements that calculate LL and UL tolerances for the individual features, as well as centered and plus minus (+/-) tolerances values. As can be seen in the diagram, the dashed box “Linked CAD Data” is still empty, which means it can be populated with different CAD embodiments that match the described context. Also, as the analysis context can be represented as a single element, as depicted at the left side of Figure 60, it is possible to copy, paste, or store it in libraries for reusability. This capability is intended for industries that execute repetitive analysis of similar topologies or tasks that involve trade-off evaluation of specific assembly conditions.

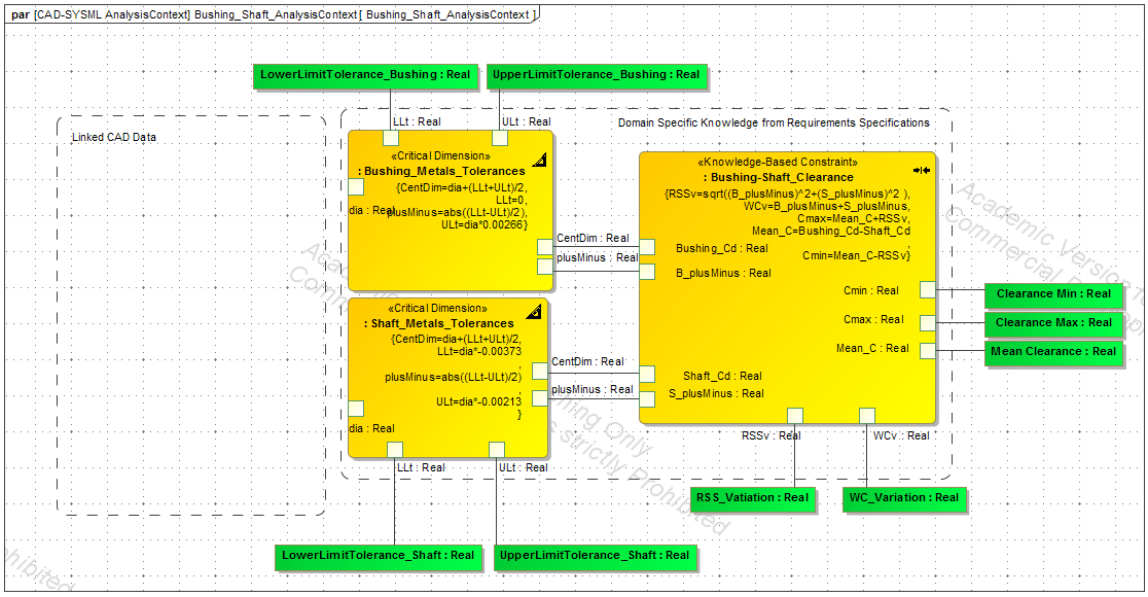


Figure 57: Parametric diagram used as analysis context template before geometric data allocation

Analysis Context Execution

In t Figure 58 the “Linked CAD Data” has been already populated with features and values that were previously imported from a features decomposition of a CAD model. As seen in Figure 57, the analysis context, with its critical value properties and

constraints, was already created as a template. Therefore, for a parametric execution, the user only has to connect the required CAD metrics into the "Domain-Specific Knowledge from Requirements Specification" dashed box of the parametric diagram. In the same way that all elements of the analysis context are connected together, the linked CAD data will use binding connectors typed as real numbers. A binding connector specifies an equal ("=") relationship between the connected elements, and also ensures that units on both sides of the association are compatible.

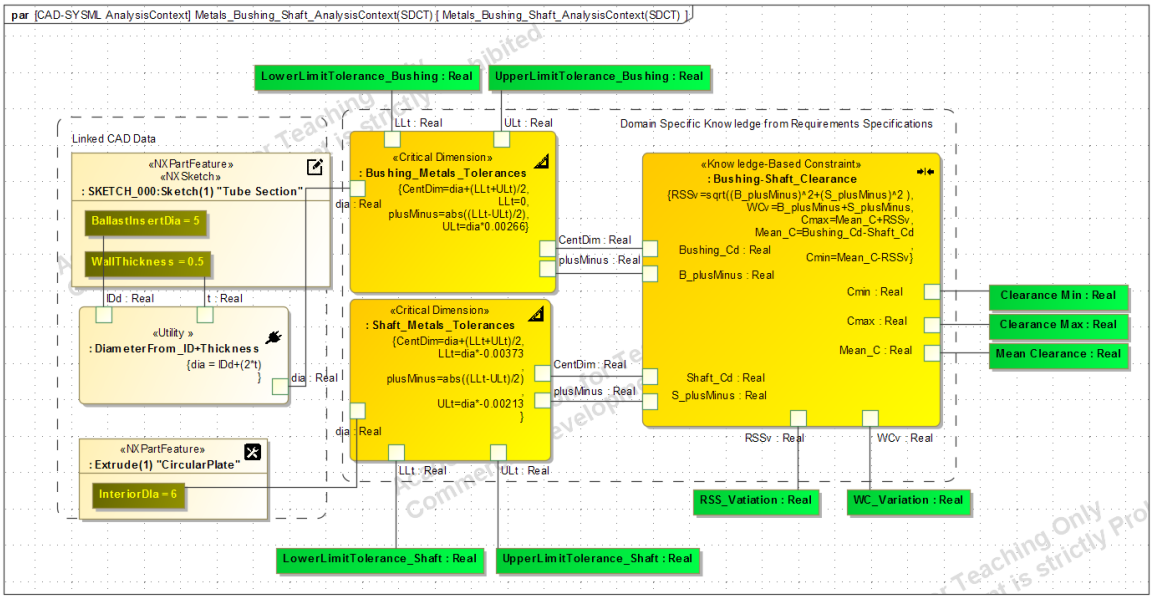


Figure 58: Analysis context in a parametric diagram that is ready for execution

After the components have been properly connected in an analysis context diagram, the system is ready to execute the model. When the model it is executed, the math console procedurally shows each performed calculation by following the analysis context internal organization (Figure 59).

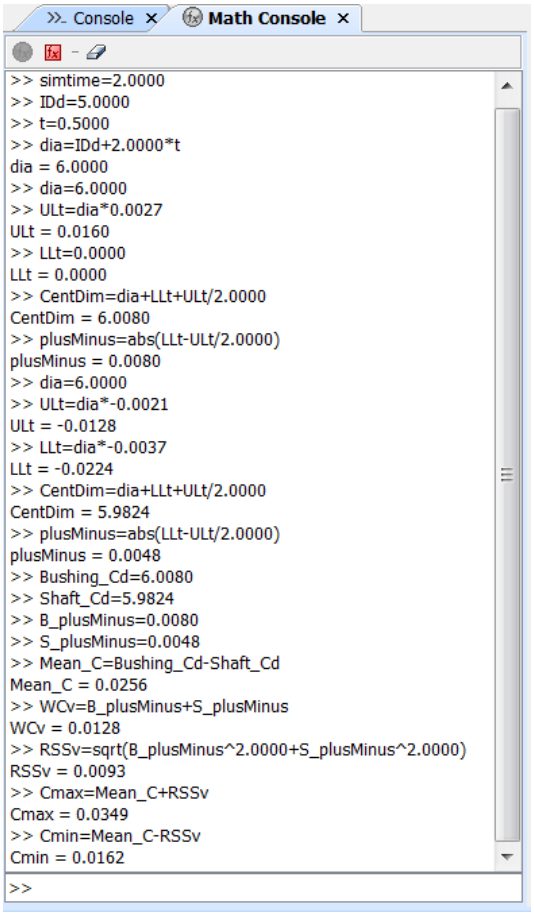


Figure 59: Math console of MagicDraw during parametric execution

Figure 60 shows the two stages of a parametric execution. At the left, the depicted block represents the same analysis context shown in Figure 57 and Figure 58. However, in this representation, the inner structure of the analysis context has been hidden. Despite this condition, all inner components of the analysis context are still visible and accessible in the block. Furthermore, as the <<CAD-SysML AnalysisContext>> is a sub-type of the <<block>> stereotype, it inherits the latter internal elements such as constraints, parts, and values, which are the required elements to perform the parametric evaluation. At the right side of Figure 60, the variables window shows the results of the parametric execution. In this window, constraints, parts, and values are also shown. However, the values placeholders of the analysis context have been instantiated with numeric values.

These values, which follow the NXProfile DSL shown in Figure 56, are target values and performance values. These values will be then stored in instances specification blocks that carry the stereotype of the specific analysis context (Figure 61);they will then be evaluated during the specification verification stage.

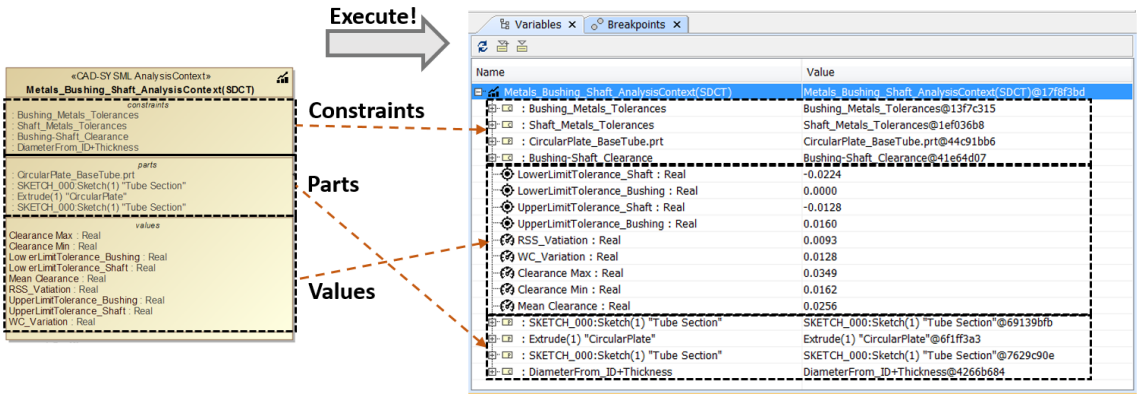


Figure 60: Execution and results of an analysis context

One of the main functionalities of this modeling framework is to perform quick trade-off analyses by parametrically changing the values that have been incorporated into the domain-specific evaluation. By doing so, designers can immediately prevent possible undesired tolerances interactions. For example, in Figure 61, when looking at important performance values such as Clearance Max and Clearance Min, we can quickly state that in Analysis 2 those parameters have negative values. This means, when both clearance limits have negative results, most likely there will be a conflicting assembly interaction in the form of a collision.

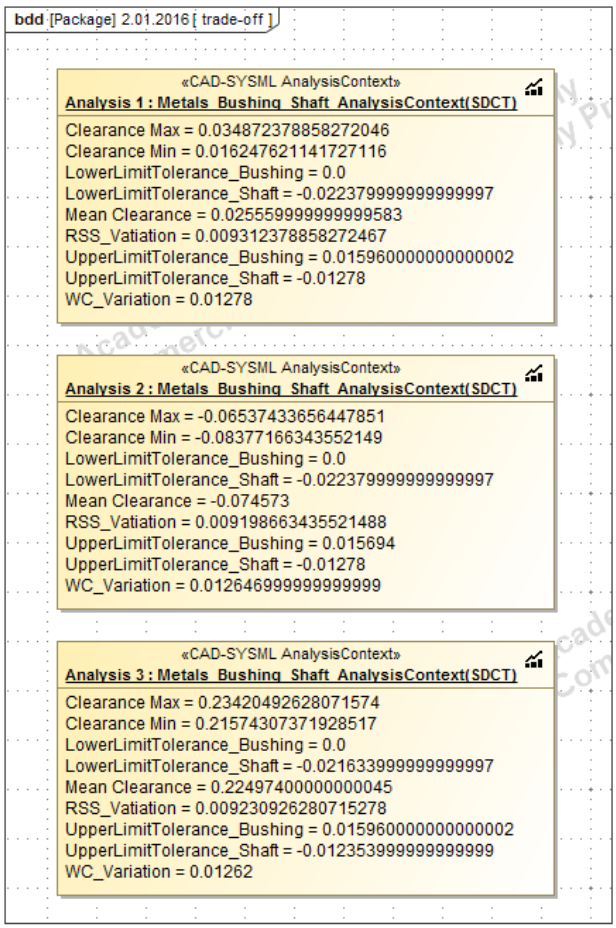


Figure 61: Instances specifications results

The following section of this document will take the results of the parametric execution and it will confirm that all manufacturing and tolerances specifications have been met during the specifications verification stage.

6.11. Specifications Verification

Figure provides a general description of the implementation regarding verification and validation. In this diagram, three different verification and validation stages are depicted at both SDCT and HCT hierarchical levels. These verification and validation stages are: manufacturing specification assessment, design specification assessment, and assembly design specifications verification.

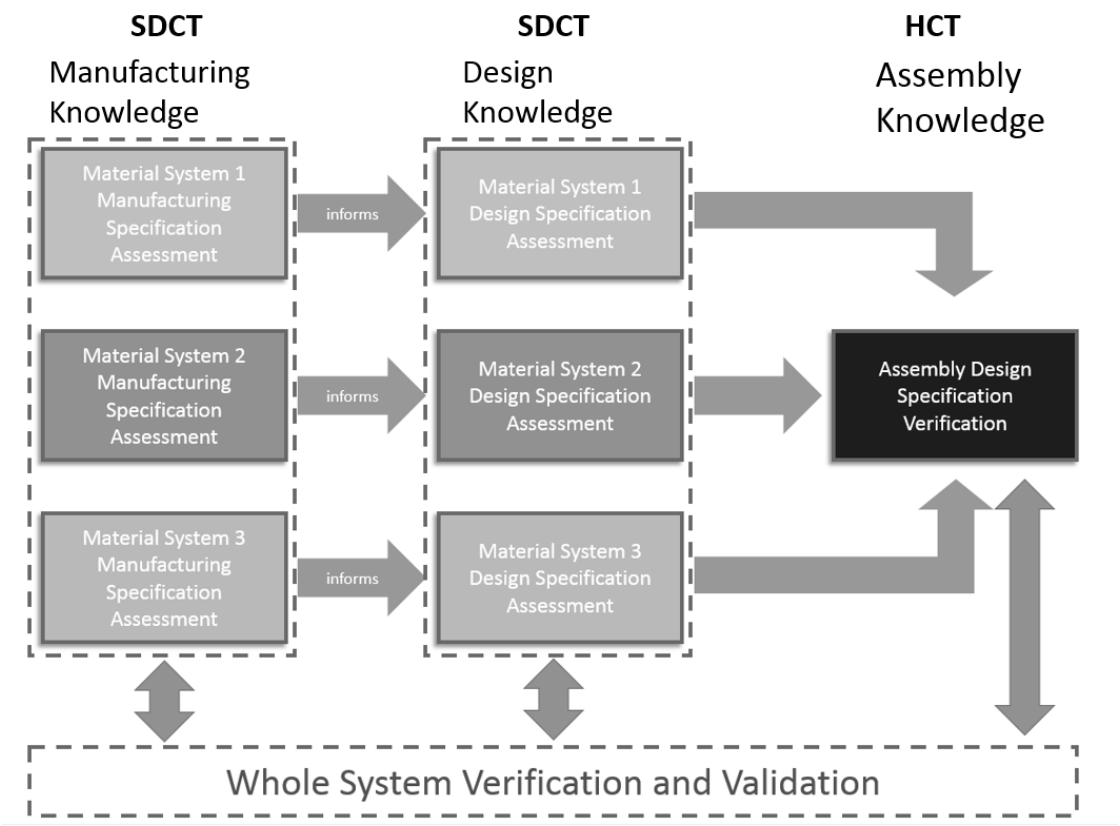


Figure 62: Overall process diagram for multi-material system assembly knowledge verification and validation

During the first stage, manufacturing specification assessment, each CAD single component (as NX Part) incorporated in the system model will be independently evaluated based on domain-specific knowledge of the material or process being identified for that component. This assessment, which comes from a manufacturing specification, will be documented by connecting linked CAD data to a mathematical constraint typed as <<Critical Dimension>>. This <<Critical Dimension>> element will then enforce that what is written in the <<Manufacturing Specification>> element is consistent with the embodiment (CAD values) of the feature that it is evaluating. For example, in Figure a simple element of the type <<Manufacturing Specification>> and an element typed as <<Critical Dimension>> are depicted. The <<Manufacturing Specification>> element has a very simple, but meaningful, written statement about the minimum distance

between a cutout and a bending feature in a SDCT sheet metal component. A failure to incorporate this piece on manufacturing know-how will most likely result in geometric deviations of the cutout feature due to the distortion that the bending operation will produce on the cutout feature. However, as this human readable statement cannot be understood for a machine[15], for this dissertation another machine readable stereotype has been created: <<Critical Dimension>>. As can be seen in Figure , both elements are connected by a verification type dependency (dashed line pointing from the critical dimension to the manufacturing specification). The dependency association has two meanings: first, it states that the <<Critical Dimension>> verifies the <<Manufacturing Specification>>; it also creates a link that defines a supplier/client relationship that can be queried at any time and from any part of the system model. For this implementation, custom icons have been established for all created stereotypes at the upper right corner of the element: manufacturing specifications are identified by a hammer and critical dimensions are identified by a 45-degree square.

[15] Although it could be a research area in the artificial intelligence domain, in this dissertation, text-based requirements and specifications will be considered as non-machine readable.

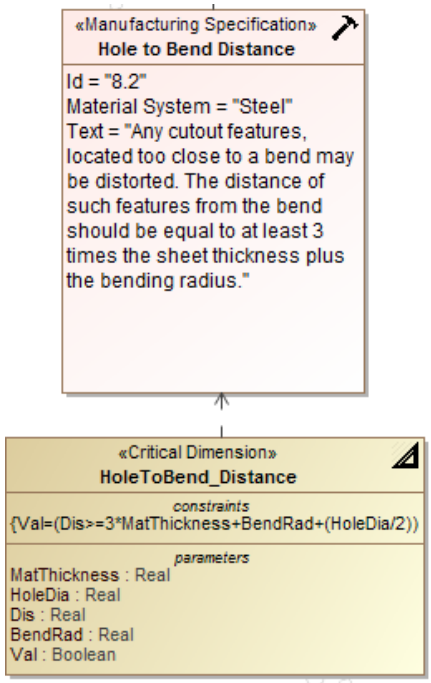


Figure 63: Text-based <<Manufacturing Specification>> that is enforced by a constraint typed as <<Critical Dimension>>

The second part of the specification verification state is called design specification assessment. This stage works in similar fashion to the manufacturing specification assessment procedure. One difference, as opposed to the previous procedure, is that in this stage the evaluation refers to the integration of multiple design features together. This procedure can be applied either to a single component or to an assembly within the same material system. However, this stage will not evaluate manufacturing variability but rather design decisions. For example, Figure shows a <<Design Specification>> called Bushing-Shaft Clearance Assessment. This specification is linked, by using a dependency association, to a <<Knowledge-Based Constraint>> named Bushing-Shaft_Clearance. In this case, the design specification is not assessing a possible geometric deviation that could occur during the manufacturing of either the bushing or shaft components. Instead, what the knowledge-based constraint does is to assess if the clearance between these two theoretically perfect components will

meet the requirements for a telescopic assembly. Thus, the specification change required for either the bushing or shaft diameter to be compliant with the telescopic requirement is a design decision and not a manufacturing consideration.

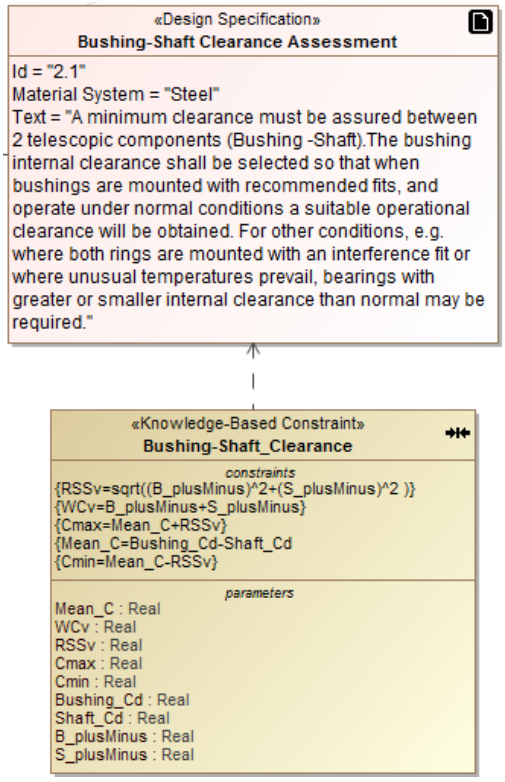


Figure 64: Text-based <<Design Specification>> that is enforced by a constraint typed as <<Knowledge-Based Constraint>>

The third verification procedure shown in Figure operates in the same way as the design specification assessment. The only difference in this stage is that the design knowledge required to execute the evaluation comes from different material systems. Therefore, this stage will operate in the HCT domain. The critical stage identified in this dissertation as HCT involves a system integration of dissimilar manufacturing know-how and material properties, which has been identified as the main source of assemblies mismatches and geometric variability during building erection.

Validation Context Execution:

The CAD-SysML Validation Context is executed in the same way as a CAD-SysML Analysis Context, as they both are specializations of the same SysML modeling element (<<block>>). Validation contexts are also created in block definition diagrams (bdd), which can be stored for reusability, and will be executed in parametric diagrams (par). Although validation and analysis contexts are at the same hierarchical level in the NXProfile, usually a validation context will contain one or more analysis contexts, and will confirm their numeric results by means of Boolean statements. The reason for this condition is that analysis contexts will apply the domain-specific knowledge of design and manufacturing, and their execution will create a set of target and performance values. However, an analysis context will not verify that those values actually meet the overall manufacturing specifications of the assembly. For this task, an artifact that performs knowledge validation has been created. Figure shows a validation context for the bushing-shaft example.

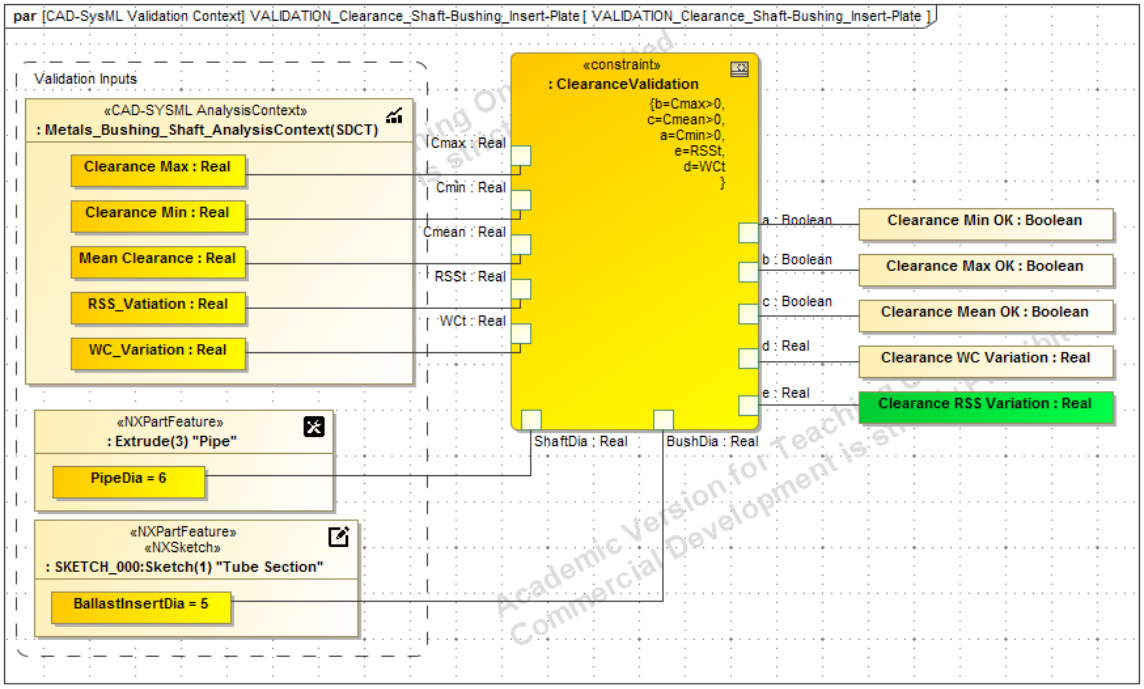


Figure 65: Validation context example

The validation context diagram, which is a specialization of a parametric diagram (par) contains five different types of elements: the validation constraint, typed as <<constraint>>; the analysis context being validated, typed as <<CAD-SysML Analysis Context>>; any complementary CAD feature required for validation, typed as <<NXPartFeature>>; any performance value, typed as <<Performance Value Property>>; any validation value, typed as <<Validation Value Property>>; and several binding connectors required to allocate values to and from the validation constraint. Functionally, the constraint block, based on domain-specific knowledge, will assess the values coming from the analysis context and will deliver a Boolean result (true or false) to notify the designer whether the manufacturing or design specifications have been met. The rationale of this custom validation procedure is to give quick and evident assessments without the need to refer to the instance value after calculations. This approach is intended to minimize the human interpretation of results in a complex

modeling environment such as BIM. However, performance values will be also provided for a deeper assessment. Figure represents a validation context (left) and the results of its evaluation (right). Parametric iterations based on knowledge allocation or direct value changes can be stored for trade-off analysis as shown in Figure .

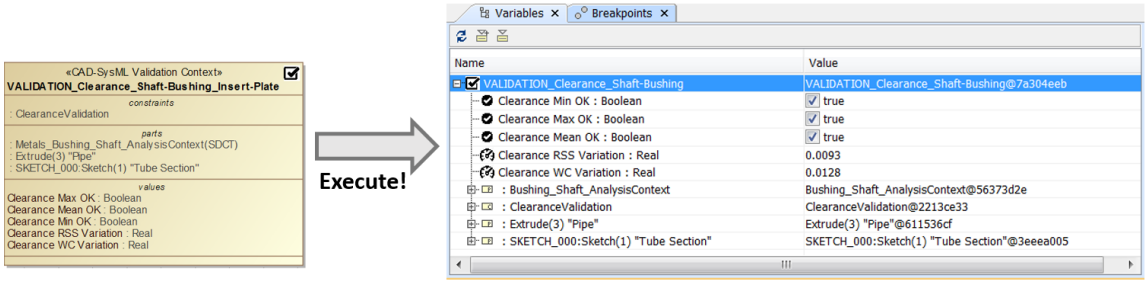


Figure 66: Execution and results of a validation context

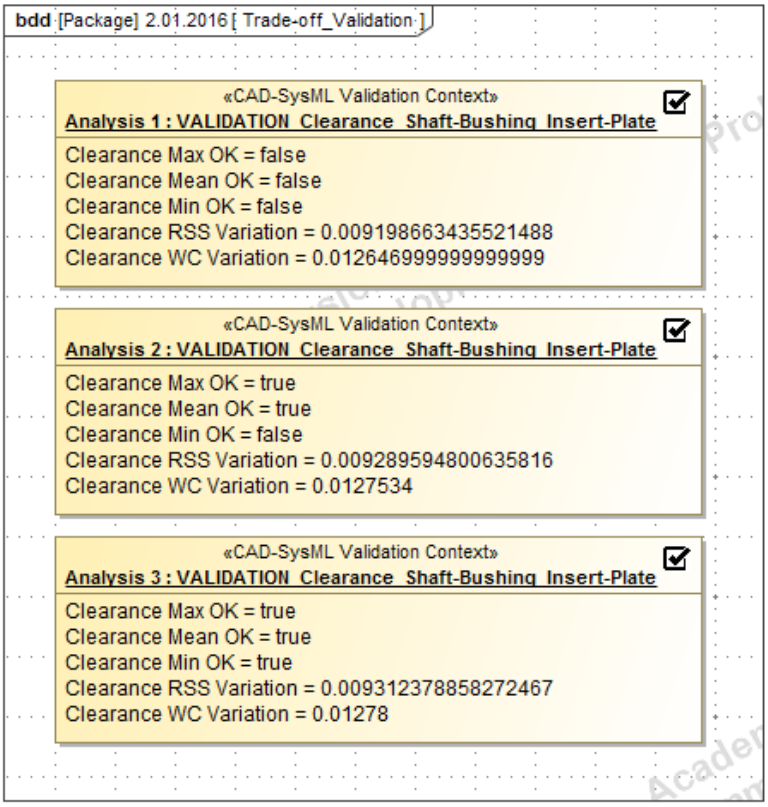


Figure 67: Trade-off analysis of different scenarios of a validation context using instances specifications

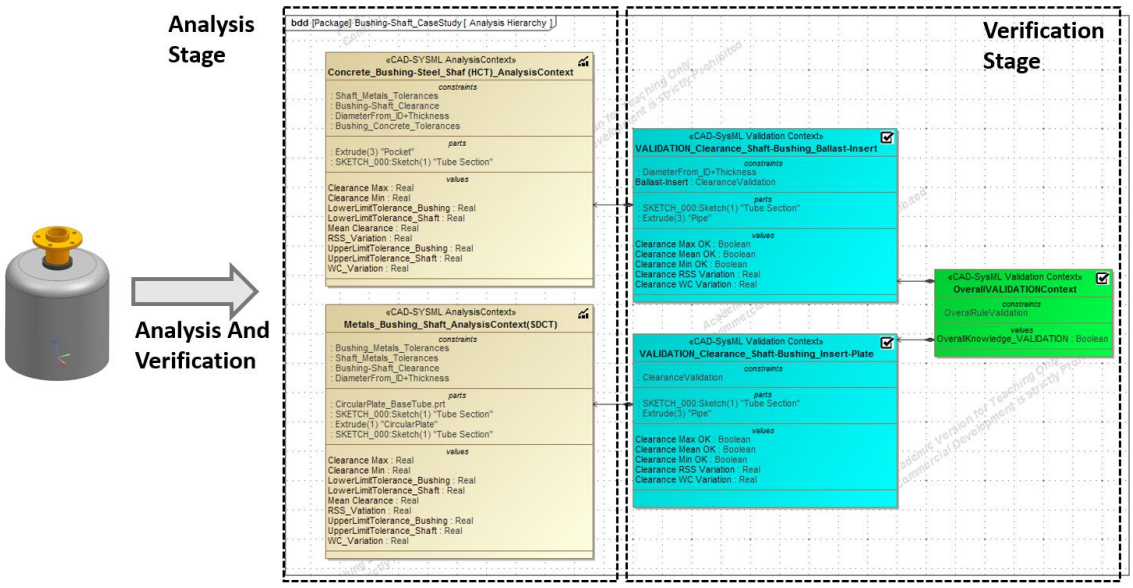


Figure 68: Overall specifications validation

Analysis and validation contexts can be nested parametrically in order to use results of previous parametric executions as inputs for new ones. In the diagram above (Figure), the analysis stage contains two analysis contexts for the two mating conditions being analyzed (metal to metal and concrete to metal). These analysis contexts are then fed into two validation contexts and finally, an overall validation context will evaluate the full manufacturing compliance of the CAD model. This feature greatly reduces the immediate complexity of the model by using the nested analysis contexts approach. Figure shows the interactive variables window of the parametric execution of the "OverallVALIDATIONContext" described in Figure . Here, the original unknown values seen in Table 2 have been calculated based on manufacturing knowledge, have been allocated directly to geometric features parameters, and have been validated through design specifications. At a high level, Figure summarizes the general approach of the implementation offered in this dissertation.

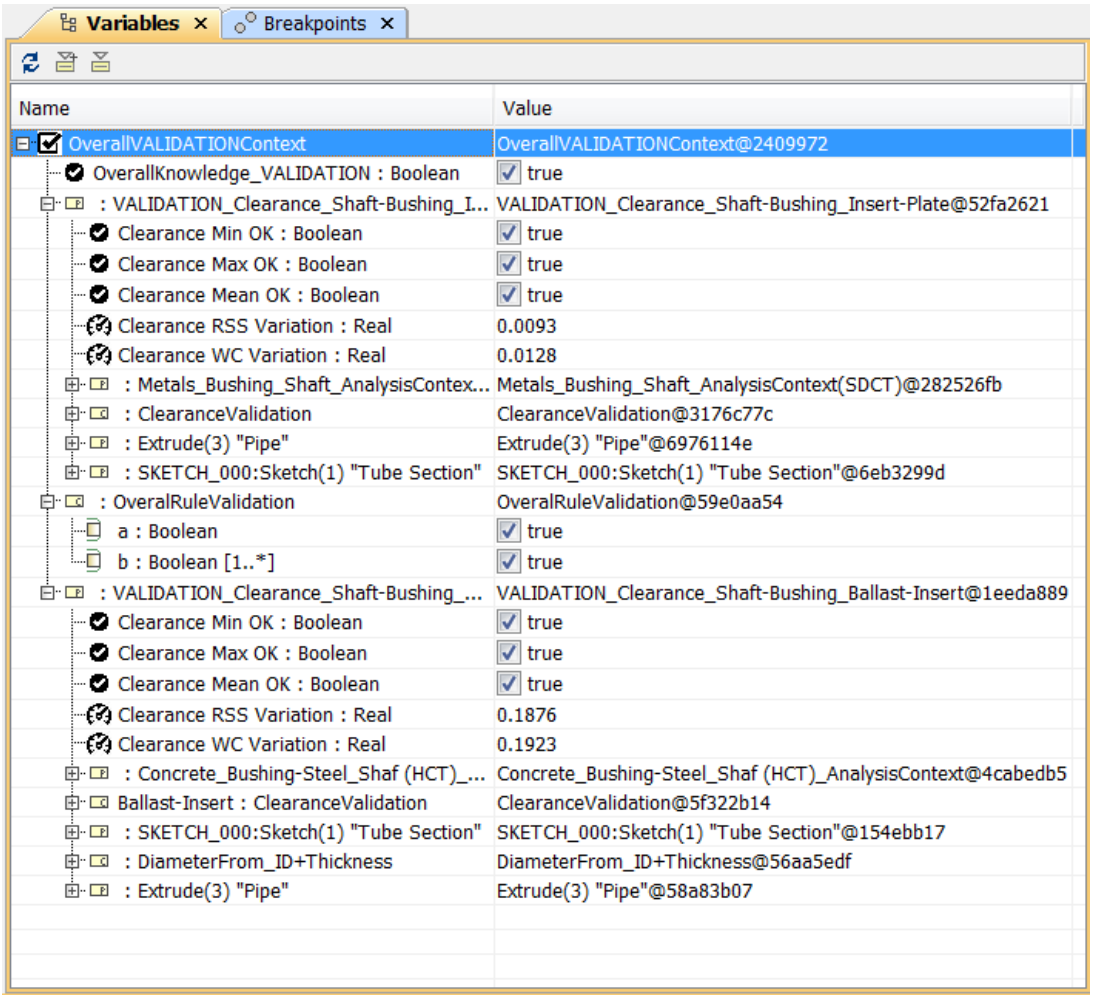
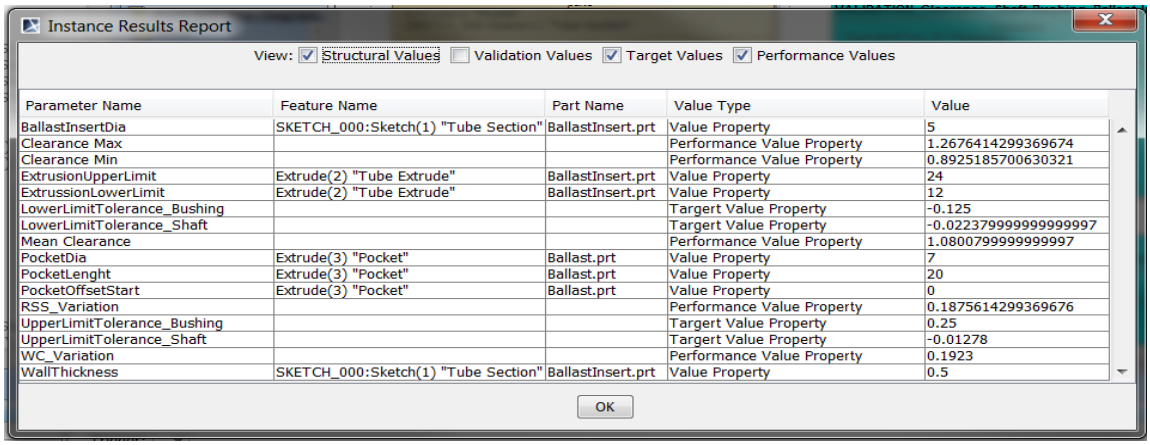


Figure 69: Overall validation context of an imported CAD model

The table shown in Figure contains the overall manufacturing and design specifications validation of an imported CAD model. Therefore, it contains all calculations and runtime values defined within the boundaries of the analysis. For each plus (+) sign in the table, a set of parts, constraints, and values will be expanded, making the model highly granulated and, sometimes, hard to navigate. In order to improve this issue, for the present implementation, a custom instances results report was created (Figure 62). This report summarizes the values properties that have been evaluated during the analysis. Also, using the same approach of the stereotypes filter (Figure 38), the instances results report offers the option of filtering the stereotypes shown in the table.

For example, in Figure 62, the validation values have been filtered to allow only numeric values to be displayed given that these values require a Boolean expression.



Parameter Name	Feature Name	Part Name	Value Type	Value
BallastInsertDia	SKETCH_000:Sketch(1) "Tube Section"	BallastInsert.prt	Value Property	5
Clearance Max			Performance Value Property	1.2676414299369674
Clearance Min			Performance Value Property	0.8925185700630321
ExtrusionUpperLimit	Extrude(2) "Tube Extrude"	BallastInsert.prt	Value Property	24
ExtrussionLowerLimit	Extrude(2) "Tube Extrude"	BallastInsert.prt	Value Property	12
LowerLimitTolerance_Bushing			Targert Value Property	-0.125
LowerLimitTolerance_Shaft			Targert Value Property	-0.022379999999999997
Mean Clearance			Performance Value Property	1.0800799999999997
PocketDia	Extrude(3) "Pocket"	Ballast.prt	Value Property	7
PocketLenght	Extrude(3) "Pocket"	Ballast.prt	Value Property	20
PocketOffsetStart	Extrude(3) "Pocket"	Ballast.prt	Value Property	0
RSS_Variation			Performance Value Property	0.1875614299369676
UpperLimitTolerance_Bushing			Targert Value Property	0.25
UpperLimitTolerance_Shaft			Targert Value Property	-0.01278
WC_Variation			Performance Value Property	0.1923
WallThickness	SKETCH_000:Sketch(1) "Tube Section"	BallastInsert.prt	Value Property	0.5

Figure 620: Instances results report

The <<Target Value Property>> Stereotype

One of the issues that have been addressed in this implementation is the calculation of many runtime values for the same <<value property>> during the execution of nested <<CAD-SysML Analysis Context>> elements. We will consider the following situation shown in Figure:

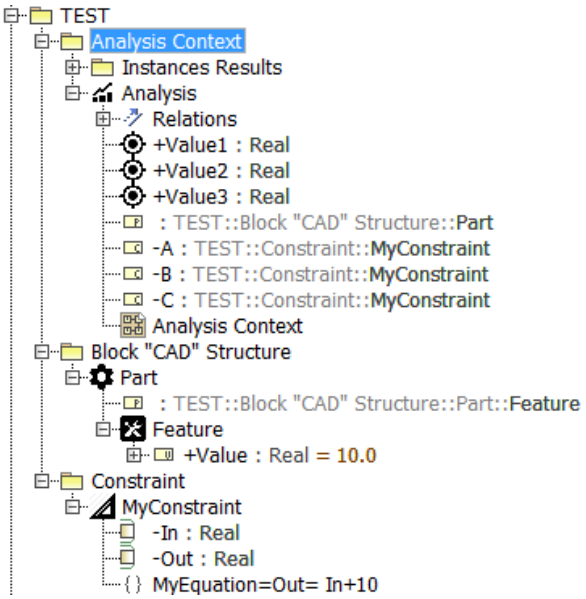


Figure71: Understanding the use of target value properties

In the capture, there is a <<NXPart>> called “Part” that contains a <<NXPartFeature>> called “Feature,” which contains a single parameter called “Value” with a default value of “10.” In the picture above there is also an Analysis Context “Analysis” that contains three parameters differentiated with a number (1,2,3). These numbers stand for different stages of analysis as seen in Figure:

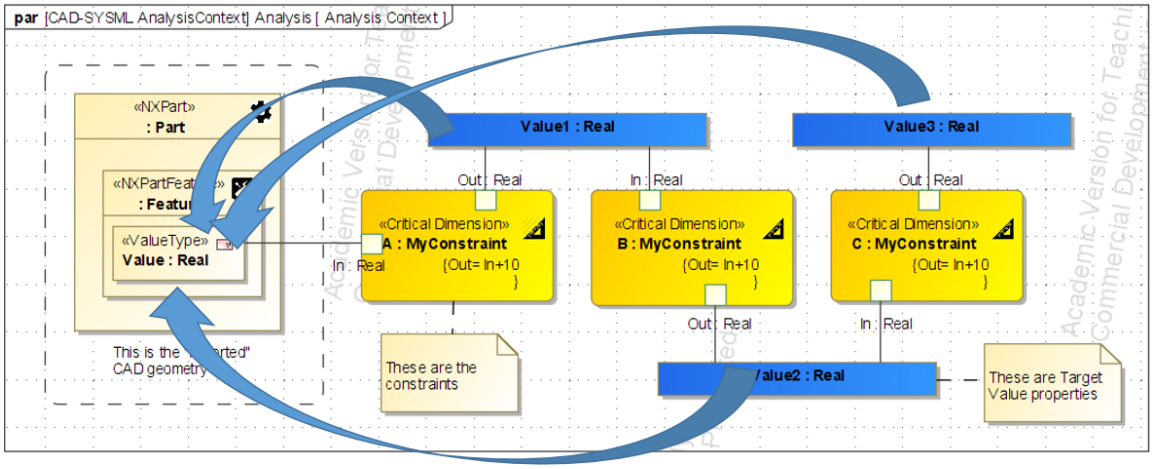


Figure 72: Target value property rationale

The picture above shows that “Value” is fed into “A” and becomes “Value1,” then “Value 1” is fed into “B” and becomes “Value2.” Finally, after going through the last <<Critical Dimension>> “C” (<<constraint>>), it becomes “Value3.” All of these runtime values (blue parameters) are required to maintain links to their initial classifier (“Value”) and to keep the internal consistency of the model. However, as the <<slot>> elements of instances specifications carry a link to their last classifier (called “defining feature” in Figure), after two constraints calculations, the link of parameters to their original classifier (Value) will be lost. To overcome this situation, in the present implementation a <<Target Value Property>> stereotype has been created. This element stereotype has a custom property called “Original Classifier” (shown in Figure 56) that carries the original <<value property>> object from where <<Target Value Properties>>

were made. Then, the slot of the runtime value has a “defining feature.” The application will query the defining feature, and request its “Original Classifier” object as shown in Figure.

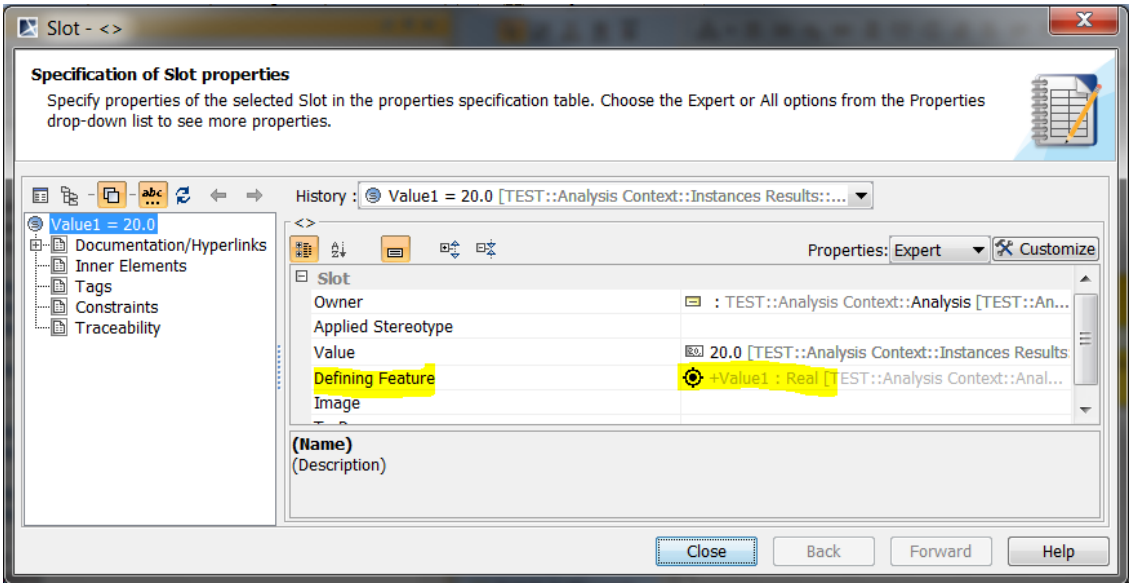


Figure 73: Specification of a slot that has a <<Target Value Property>> as its defining feature.

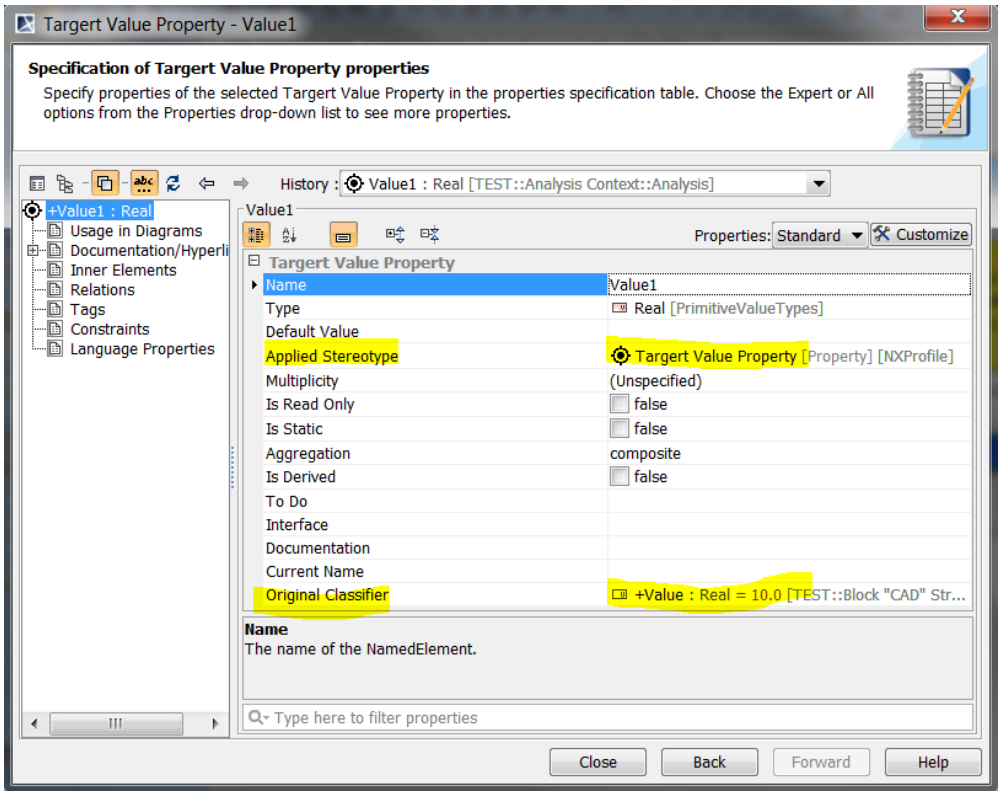


Figure 74: Target Value Property with custom property "Original Classifier"

CAD-SysML Consistency Approach:

This is a complex matter because both representations (SysML and CAD) are not mapped one-to-one on each side. On the one hand, the CAD representation is compliant with all the fundaments and rules of solid modeling, which ensure the correct depiction of three-dimensional objects in the Euclidian space. That is, faces, vertices, and features are geometrically and unequivocally specified. On the other hand, the system model representation of the CAD component could be a sub-set of the CAD entities. For example, if, while importing, the stereotype filter is used and we uncheck the << NX Coordinate System>> stereotype, the SysML representation will not include such CAD entities. This is one of the characteristics of this integration – a SysML representation helps to synthetize only what is important for a specific user in a specific context. This is what we call a “view” of the model. Thus, the consistency analysis must be unidirectional

– obtain the first element on the SysML side, then determine if the element is available on the CAD side. If not, it is a missing element and the mismatch must be added to the consistency report. If it is, the user must compare parameters and values (names, values). When differences are found, it is necessary to add those values to the consistency report. In this approach, from each CAD part file we extract a list of components (if this part is an assembly) and a list of features. Within the NX format, there is a directed relationship between features – any given feature may have any number of feature parents and any number of feature children. This structure is known as a Directed Acyclic Graph or DAG. When extracting the list of features from a CAD part, this graph structure is simplified into a tree as follows: perform a depth-first traversal of the feature graph marking each visited node and add a node to our tree if it has not been visited previously. A generic traversal routine has been implemented, which performs a simultaneous traversal of a (possibly empty) MagicDraw tree and an extracted CAD tree. At each step, the traversal routine maintains a pointer to a node in the MagicDraw tree and in the CAD tree; it then examines the list of child nodes for each, and from these invokes first a custom handler and then invokes itself recursively. The custom handler routine permits an action specific to a given task to be performed at each node in the tree. For example, during import the handler routine creates a new node and adds child nodes, while during a consistency check the handler routine simply compares the information between the nodes in the NX and MagicDraw trees and reports on inconsistencies. This routine will have a dialog box allowing the user to pick the correct information while scrolling through the inconsistency list (Figure). For each inconsistency, the user will have to

choose either the Siemens NX value, the SysML value, or do nothing (leave the difference unresolved until later on the process).

These are the steps implemented for such a capability. Some are User Actions (UA), some are Internal Actions (IA), and some are outcomes (O)):

1. UA: Right click on a SysML element with the stereotypes <<NXPart>> or <<NXAssembly>>
2. UA: Click on “Execute CAD-SysML Consistency Analysis”
3. O: Show the message box “Checking Consistency”
4. IA: Execute analysis following the use cases shown in Figure
    - If there is a SysML element and a CAD element, the system looks at the SysML element and compares its values to the CAD element. If values (names or numbers) are different, they are added to the consistency report.
    - If there is not a SysML element, but there is a CAD element, the system does nothing
    - If there is a SysML element, but there is not a CAD element, the system warns the user about the missing element by adding the inconsistency to the report.
5. O: The report is presented on the screen (Figure).

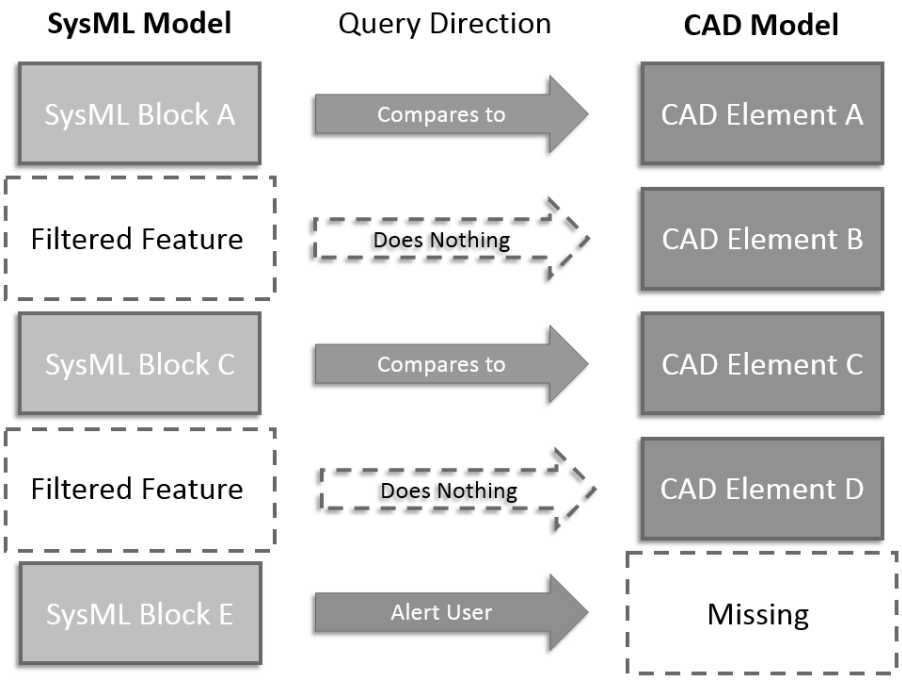


Figure 75: Use cases for (NX- SysML) external consistency analysis

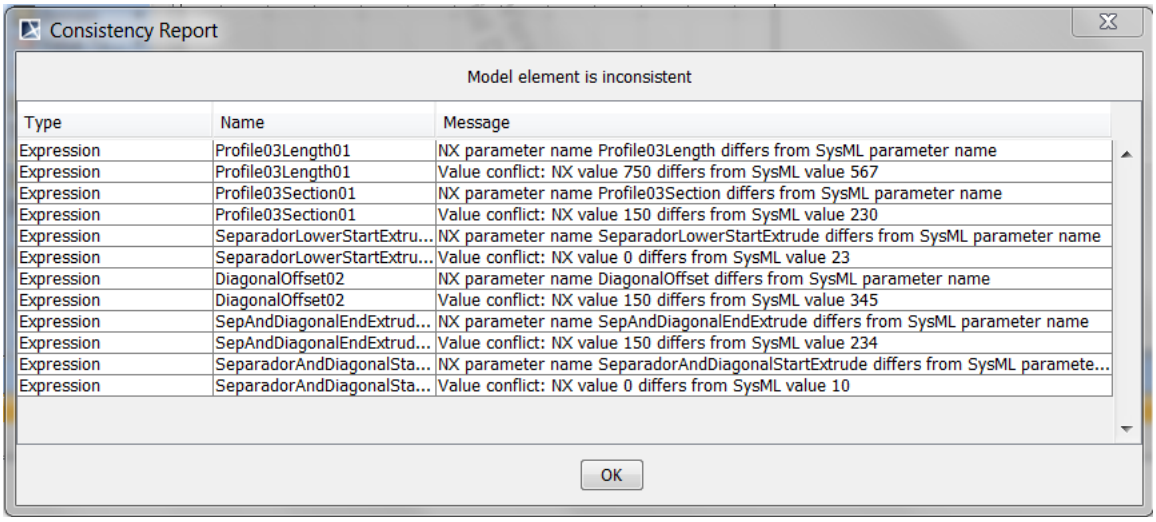


Figure 76: Consistency report example

Resolve CAD-SysML inconsistencies:

After inconsistencies are found by using the "Execute CAD - SysML Consistency Analysis," users can resolve inconsistences choosing between SysML and CAD data by means of a custom "Resolve Model" tool created in this implementation. There are two approaches built into the command: resolve all by selecting SysML or CAD, and resolve

one at a time by selecting SysML or CAD for each mismatch. This capability allows the user to recover a broken SysML model by restoring consistency with its CAD counterpart.

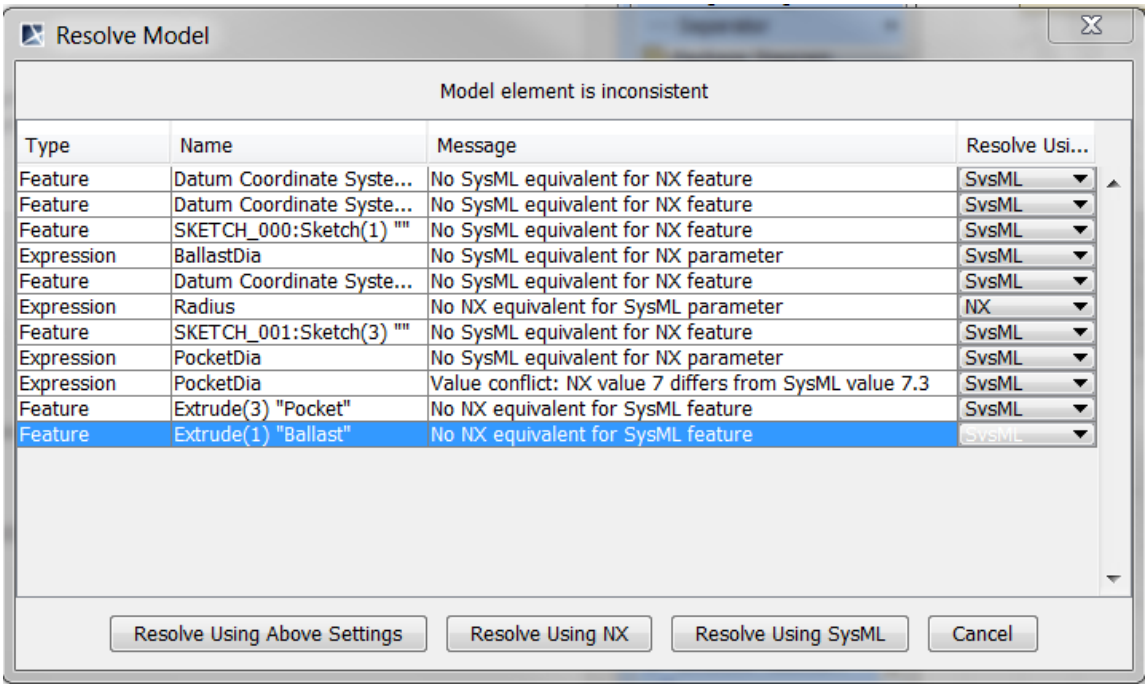


Figure 77: Resolve NX-SysML inconsistencies

6.12. Knowledge-Compliant Geometry Update

After the analysis and validations have been completed, the user will be able to commit those changes in the CAD model. This action will generate a manufacturing knowledge-compliant model that will include a complete assessment of manufacturing specification, design specifications, and tolerances calculations of a building assembly. Figure shows the main steps towards the knowledge-compliant geometry update from the instances specification perspective.

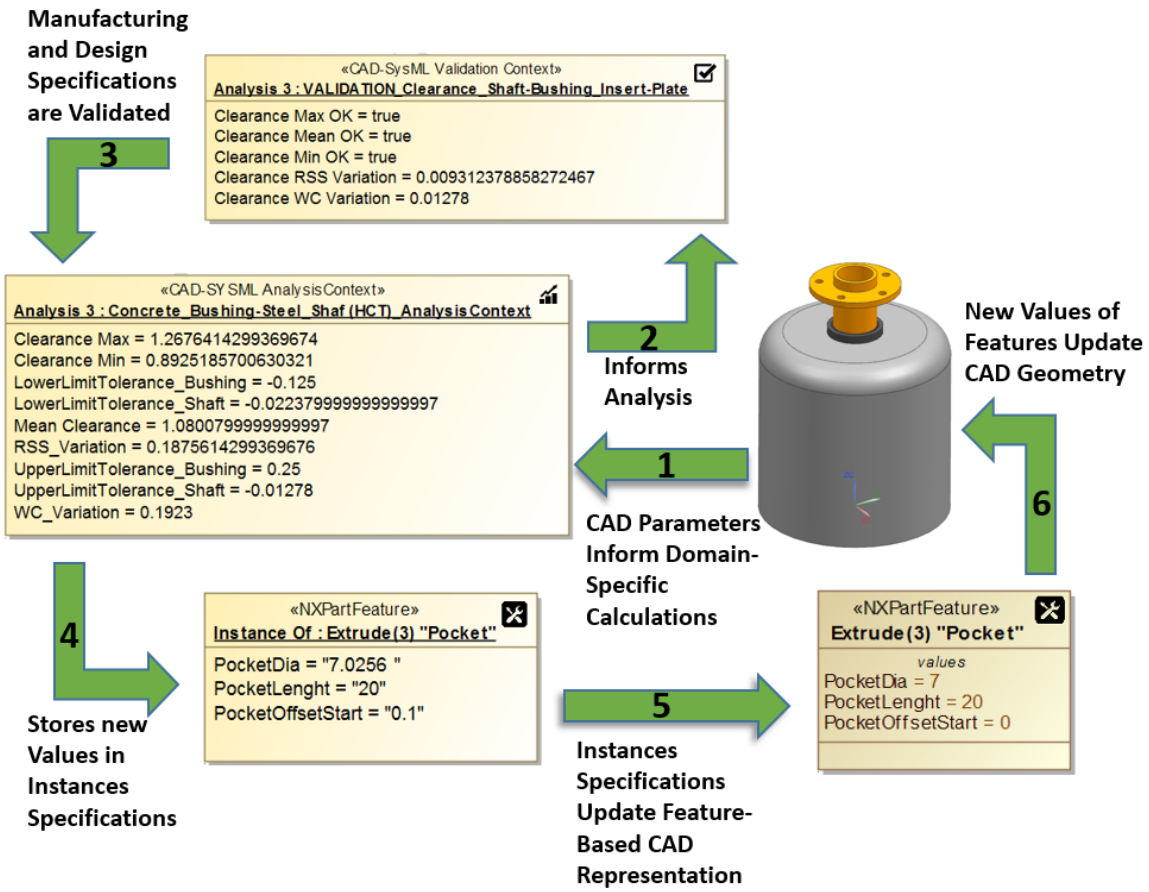


Figure 78: Knowledge-compliant geometry update

1.The features parameters of the imported CAD geometry will inform the calculation of knowledge-based material system-specific values. These values can be <<Target Value Property>>, to be re-allocated into the CAD parameters; or <<Performance Value Property>>, to be used as simultaneous feedback of specific feature-independent metrics such as tolerances or clearances values. The results of these calculations will be stored in <<Instances Specifications>> like the one showed in Figure

.

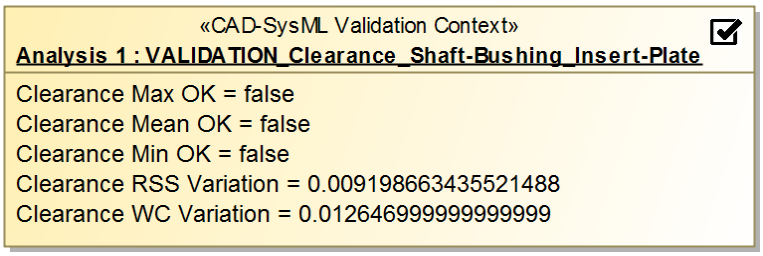


Figure 79: Instance specification element example

An instance specification is the manifestation of a <<block>> element. As seen in Figure , an instance specification will be created from a "classifier" element. Also, the instance specification will carry a value property for each parameter of its classifier. Value properties applied to instances specifications are of the type <<slot>>, and can be found in the "Owned Elements" field of the specifications box (Figure).

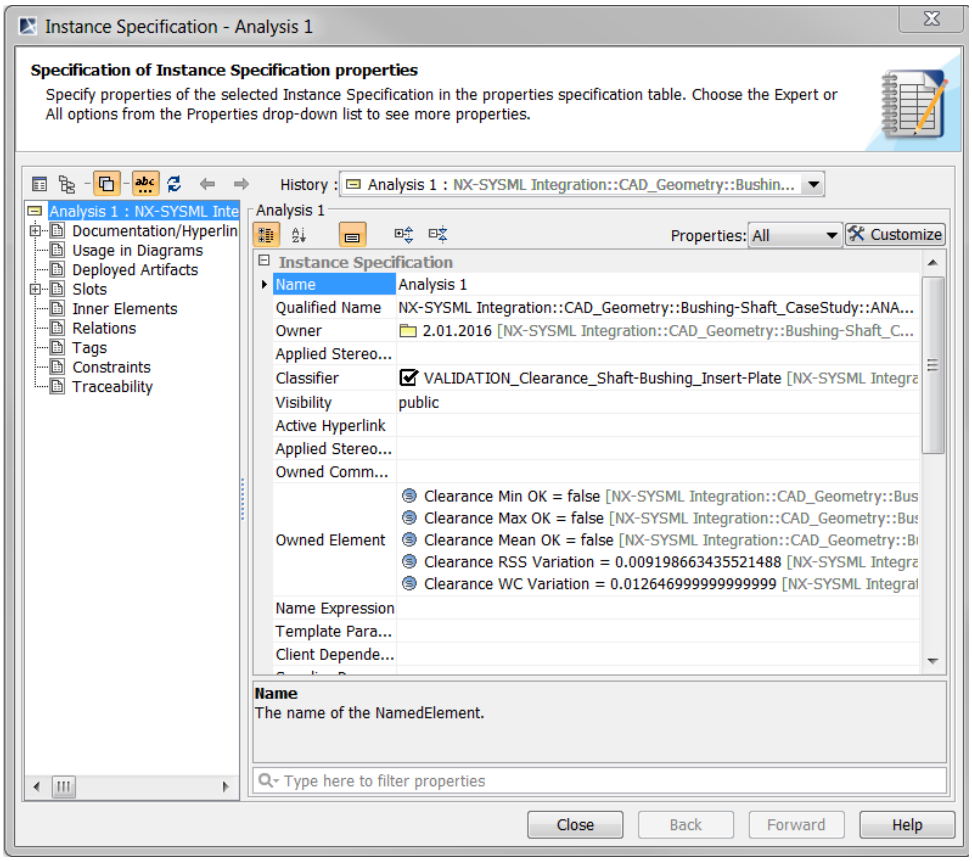


Figure 80: Details of instance specifications properties

2 and 3.<<Target Value Property>> and <<Performance Value Property>> elements will be fed into Analysis 3 <<CAD-SysML Validation Context>> to verify that the numeric values actually fulfill the intended requirements described in the manufacturing and design specifications.

4. After being verified, the <<Target Value Properties>> parameters (revised values from the CAD model) will be pushed out to <<Instances Specification>> elements for storage, trade-off analysis, and geometry updating. These instances specifications are storable copies of the original imported <<NXPartFeature>> elements. The only difference is that <<Instances Specification>> blocks have a composite naming convention, where the original feature name will be combined with the given name of the instance separated by a colon (:) symbol as shown in Figure .

5. The revised parameter values of the instances specification will update their <<block>> counterpart, which is the original imported <<NXPartFeature>>. This action will push the value of each <<slot>> of the instance specification to upgrade the default value of the original CAD feature. For example, in Figure the “PocketDia” feature value of the imported CAD feature (7) will be replaced by the calculated “PocketDia” slot value of the instance specification (7.0256). This action will be performed by using the custom artifact “Updated Block Value Property from Instance” (Figure).

6. All new default values of the <<NXPartFeature>> parameters will be pushed back to the NX CAD model for final geometry update.

Update Block Value Properties from Instances Results:

As previously explained in this section, the Update Block Value Properties from Instance capability offers the option of populating the block structure of the model with

the values obtained from parametric calculations. Possibly, there will be several instances that contain results from different parametric calculations. Those instances will be stored in a folder with the name of the target block of the parametric execution (or other name), under a folder “Executions Results.” Once the instances are stored, the user will be able to right click that PartName.prt instance and apply the command “Update Block Value Properties from Instance.” Then the application will traverse all instances blocks inside the folder, return their block classifier (root block), look up value properties, and populate the instance value property in the default value property of the blocks. As the instance folder will contain only a subset of the elements of the block structure, the application will not flag a missing component during this stage. The application will simply scan the instance folder, look for the root, and update the root value. To commit new value properties to the <<NXFeature>> parameters, the user will use the “Update Block Value Properties from Instance” command shown in Figure .

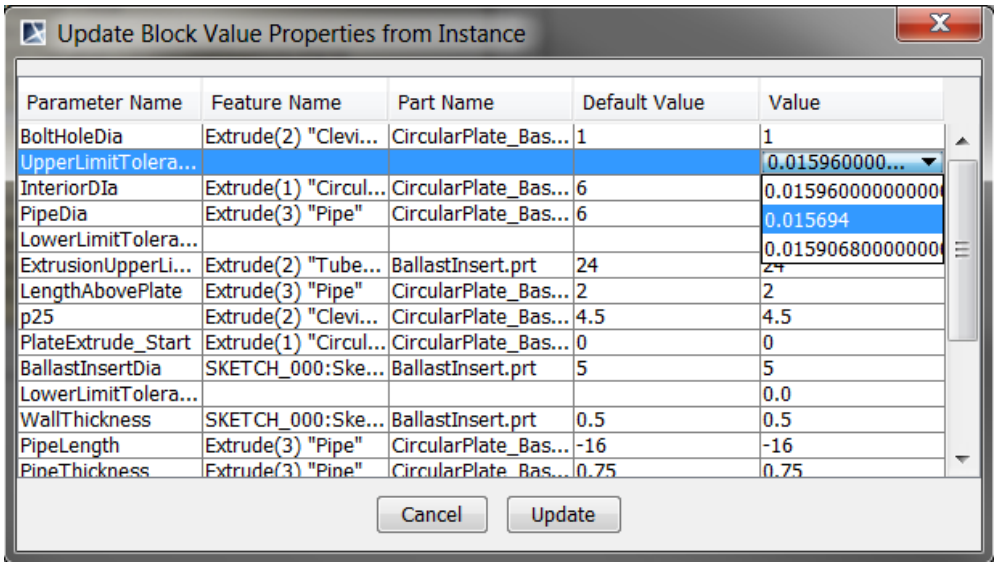


Figure 81: Update block value properties from instance table

Update CAD model from SysML model:

This action will populate the CAD model with new value properties of features (and potentially new names) from the SysML Model. Usually, this will happen after executing parametric calculations on the SysML side. All parameters typed as target values are sent back to the feature based SysML representation of the CAD model. Finally, the command "Update CAD from SysML Model" will create the "Knowledge Compliant Geometry."

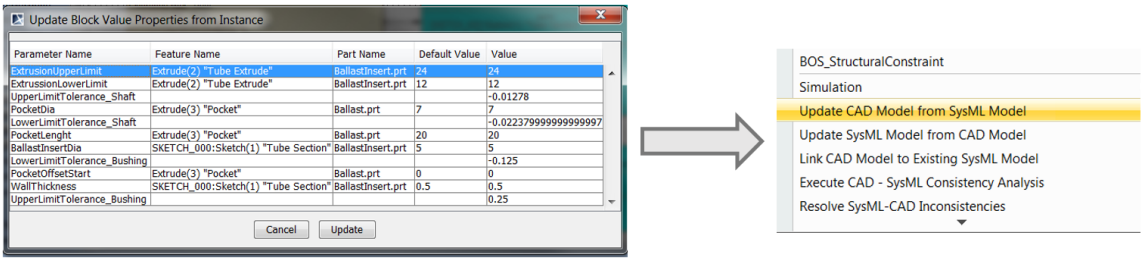


Figure 82: Update geometry procedure

Update SysML model from CAD model:

This action will populate a SysML model with new value properties of features (and potentially new names) from the CAD Model.

Although this dissertation only addresses knowledge-based matters, there is great potential for extensibility in several other areas of the building lifecycle. The next section will introduce a SDCT sheet metal case study based on a real project where the proposed approach has been tested for manufacturing.

CHAPTER 7: System Evaluation

7.1. Case study 2: Lower Chord Assembly, a QuadPod Solar Canopy System

QuadPod is a three-dimensional truss racking and mounting system for largescale photovoltaic (PV) power generation infrastructures. The system was developed as part of the Georgia Tech Research Institute's (GTRI) work as part of the United States Department of Energy's Sunshot program. In the third quarter of 2011, GTRI was awarded a BOS-X award, as part of Sunshot's broad initiative to revolutionize the solar industry. GTRI researchers developed radical new products that would allow solar to compete with other conventional energy sources by reducing projected labor costs and boosting installation efficiencies [148].

The QuadPod system is universal and is predicated on the principles of large-scale pre-assemblies and material reduction through the use of deep three-dimensional trusses. PV modules are aggregated into a structural mega-array in a pre-assembly facility or work area near the installation site, loaded onto lifting equipment, and deployed to the site as a complete prefabricated system, thus moving the vast majority of assembly activities into a central, controlled environment. The QuadPod system, as an architectural product, enables multi-functional spaces, including covered work spaces, shaded parking areas, and remote field hangars. The main premise is to be easily assembled on the ground by nonspecialized local technicians. It yields 80 to 90 percent more kilowatts per acre compared to conventional canopy systems, commanding the highest canopy-to-ground coverage ratio [148].

Figure 83: QuadPod Canopy system V1

A Version 1 QuadPod system has been successfully designed using galvanized sheet metal components and custom tooling and has been deployed in the field as a pilot project. However, the excessive use of hardware, some structural issues, and the lack of a scale manufacturing approach led to the development of QuadPod Version 2, where most of the components were optimized for lean manufacturing and easier field installation. The author of this dissertation has been appointed as lead designer for the optimization of all the components of the system. The main challenges of this endeavor have been the calculation of tolerances and clearances and the material reduction optimization through engineering design. The general scope of the optimization process was defined as follows:

- Standardization of lengths with respect to panel dimensions
- Ability to vary lengths of systems
- Reduction in part count and complexity
- Reduction of bolts and splices

This case study will test how applying the proper domain-specific knowledge to a set of different parts of a sheet metal assembly can reduce the likelihood of tolerances

and clearances mismatches. To set a realistic scenario, all the components of the studied assembly will be imported independently into the SysML environment. Despite the fact that the present case study includes Single Domain Construction Tolerances (SDCT), the simple restriction of importing the components independently emulates the circumstance where all assembly parts come from different sources, different vendors, or different applications, which was the case for the QuadPod project. Another restriction of the exercise is the need to design every part at its nominal value, and to follow the default settings of the feature-based capabilities of the CAD software. For example, bending radii of formed parts have been left as suggested by the CAD application. Thus, all components will perfectly fit in the CAD environment. In the end, after running design and manufacturing specifications for each independent, dimensionally-nominal component of the assembly, all components will be collected together in a non-linked assembly to assess the results of the analysis.

Restrictions of the exercise:

- Components will be modeled as nominal, which means clearances will be zero and fastener holes will be modeled as the nominal value of the bolt, among other nominal conditions.
- Components will be independently imported to the SysML platform. Mating conditions, fastening features, and their clearances will be calculated independent of geometry, based on design and manufacturing specifications stored in the SysML environment.

Parts to be tested:

- Inner lower chord

- Outer lower chord
- Lower chord stiffener
- Upper chord hat stiffener
- Upper chord splice
- Transversal welded plate

These six components will be assessed from a manufacturing standpoint and four will be also evaluated from a design specifications standpoint (assembly clearances). The four components to be evaluated for assembly clearances belong to a critical and repetitive node of the structure that showed tolerances issues in the previous design. Figure shows the general geometric situation of the assembly clearances exercise. Figure , Figure , and Figure show the three different kinds of construction knowledge addressed in this dissertation, which will be further decomposed in an additional analysis.

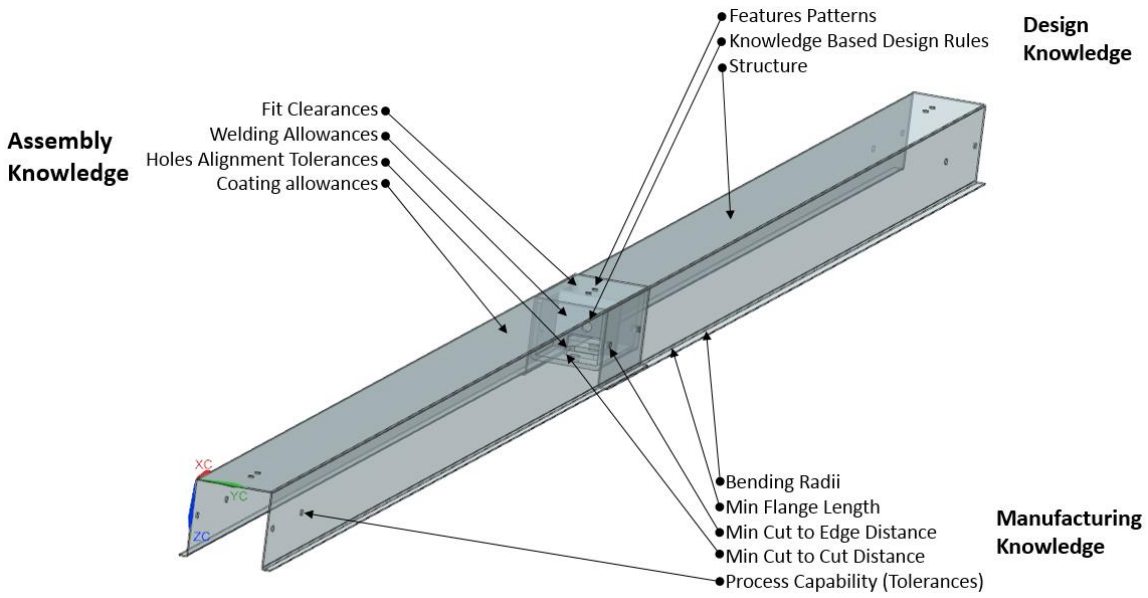


Figure 84: Different knowledge for a critical assembly design

The nominal depiction of the studied assembly is shown in Figure and Figure . It should be noted, especially from Figure , that the CAD assembly has been modeled with nominal fit, as assembly clearances cannot be identified among the components.

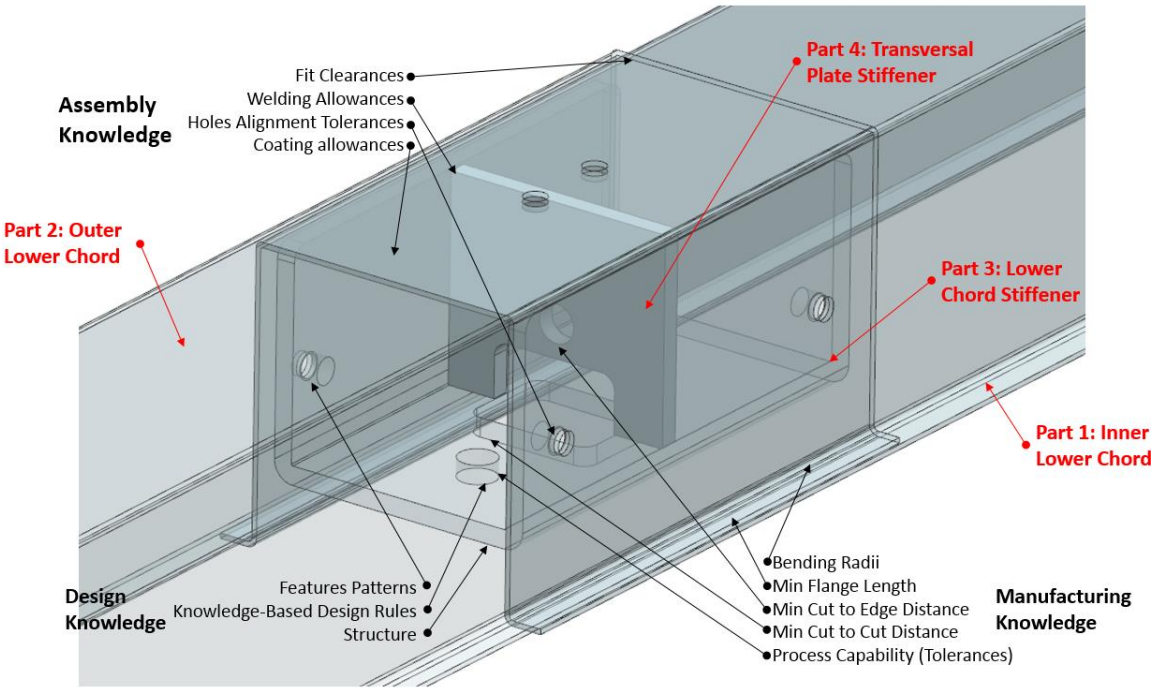


Figure 85: Knowledge integration for a critical assembly design

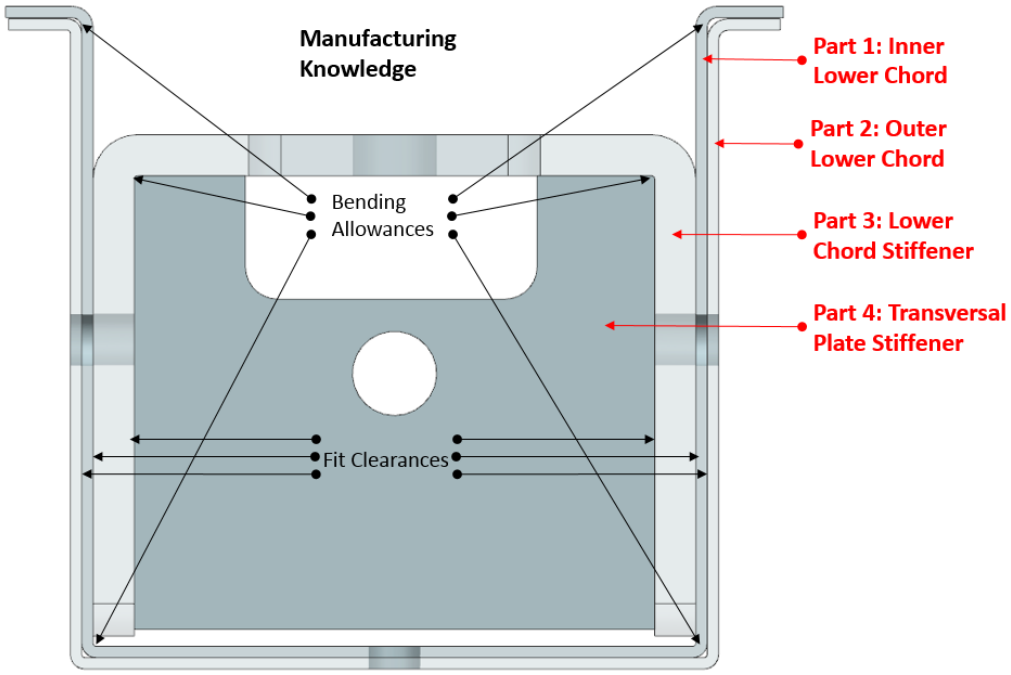


Figure 86: Nominal geometry for the studied assembly

7.1.1. Structural decomposition of the studied components

Structural decomposition of the six QuadPod components chosen for this case study are shown in Figures 87-92.

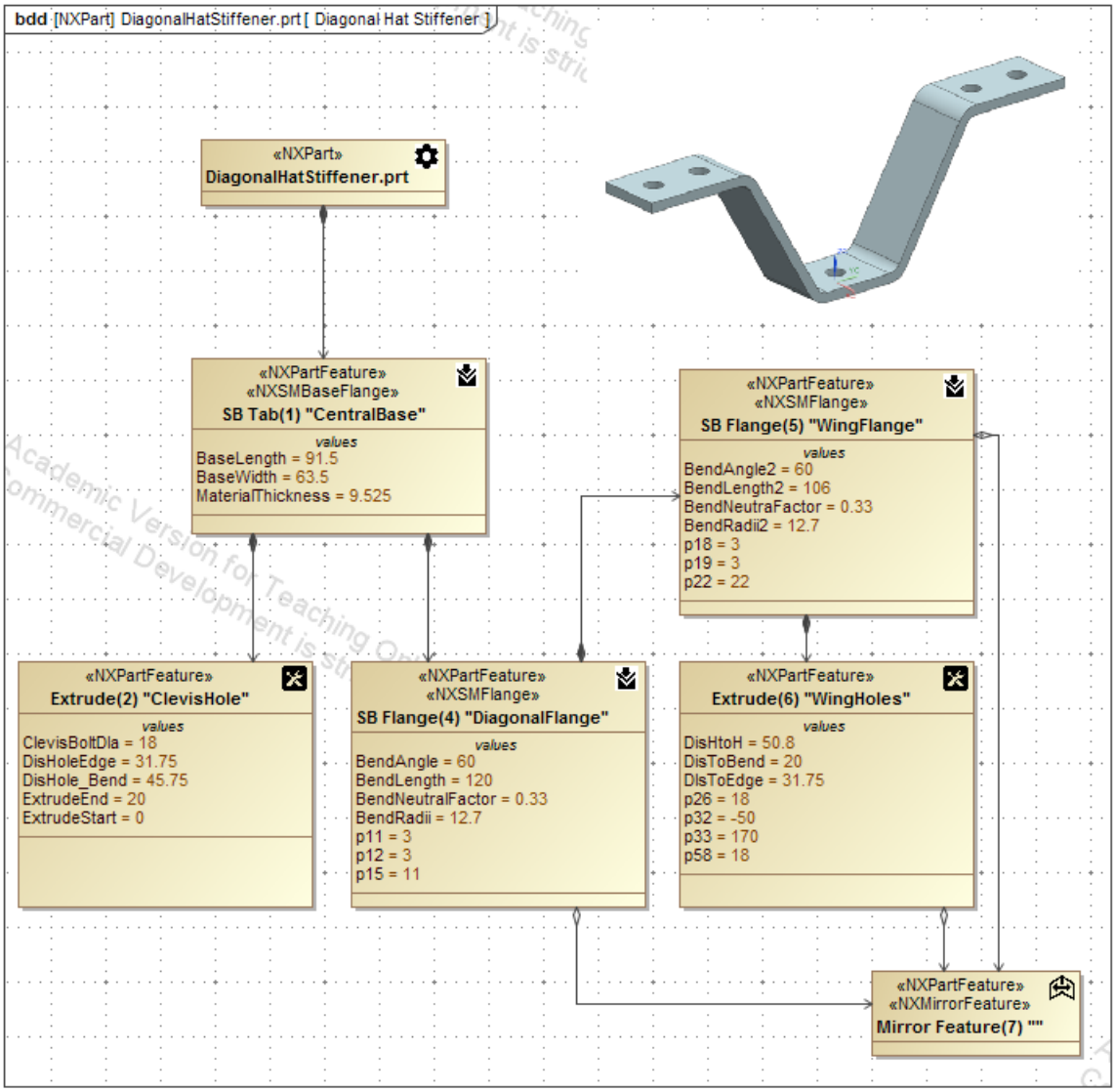


Figure 87: Diagonal Hat Stiffener: feature-based decomposition

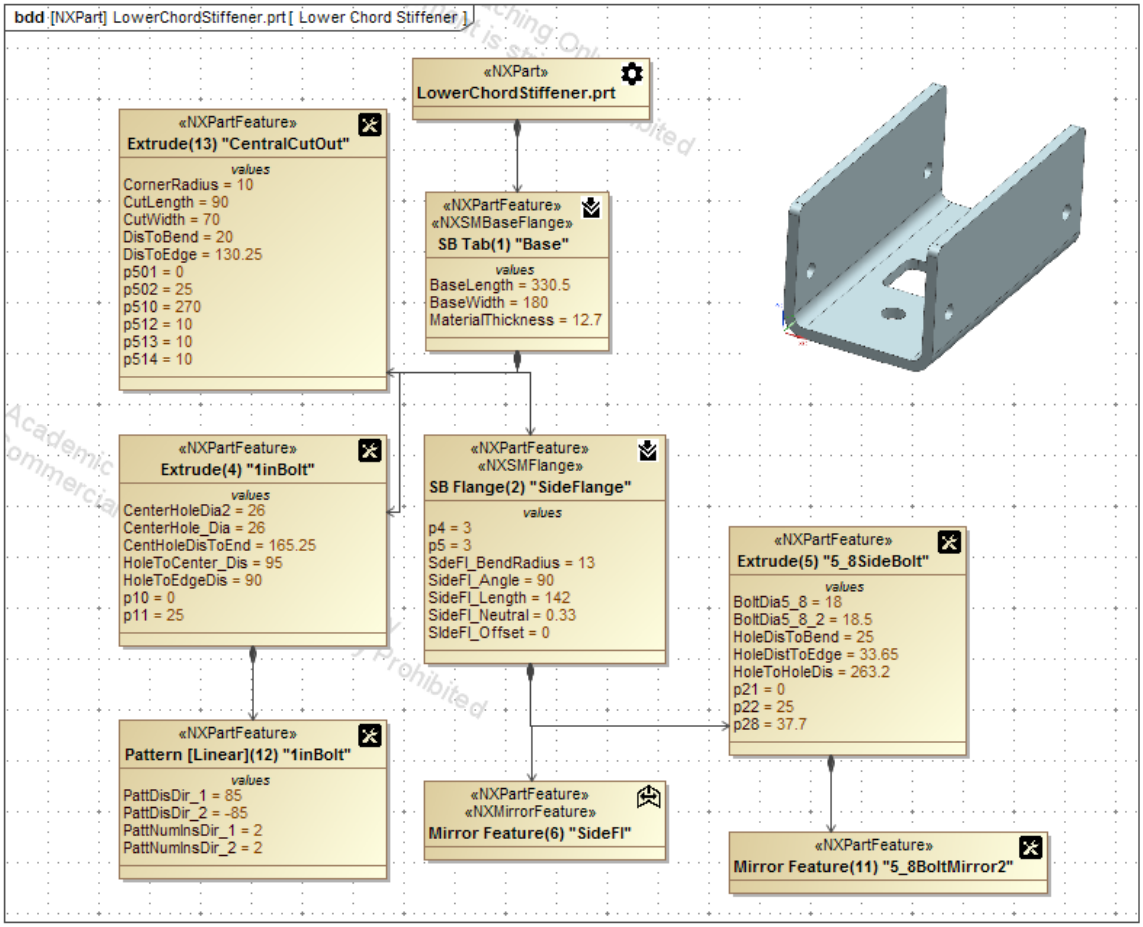


Figure 88: Lower Chord Stiffener: feature-based decomposition

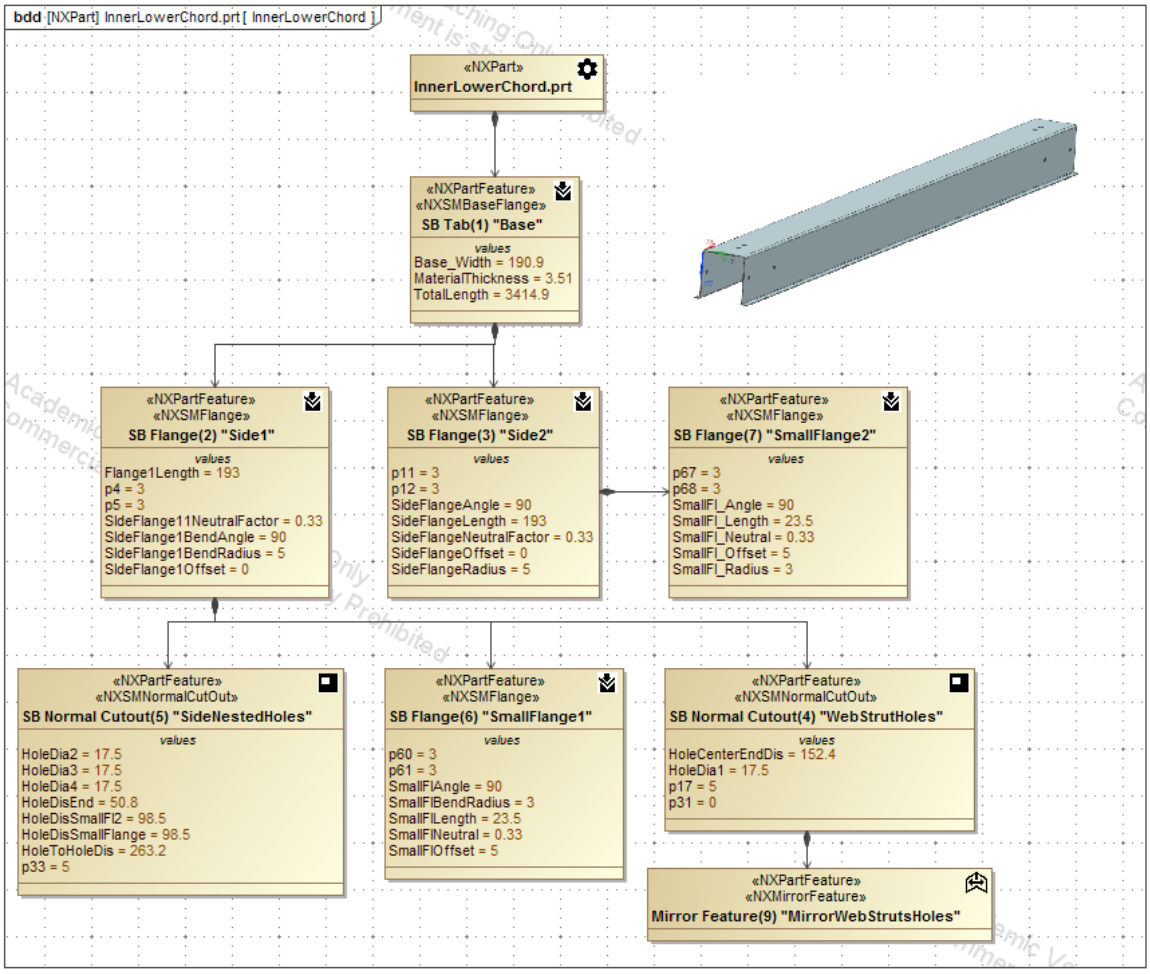


Figure 89: Inner Lower Chord: : feature-based decomposition

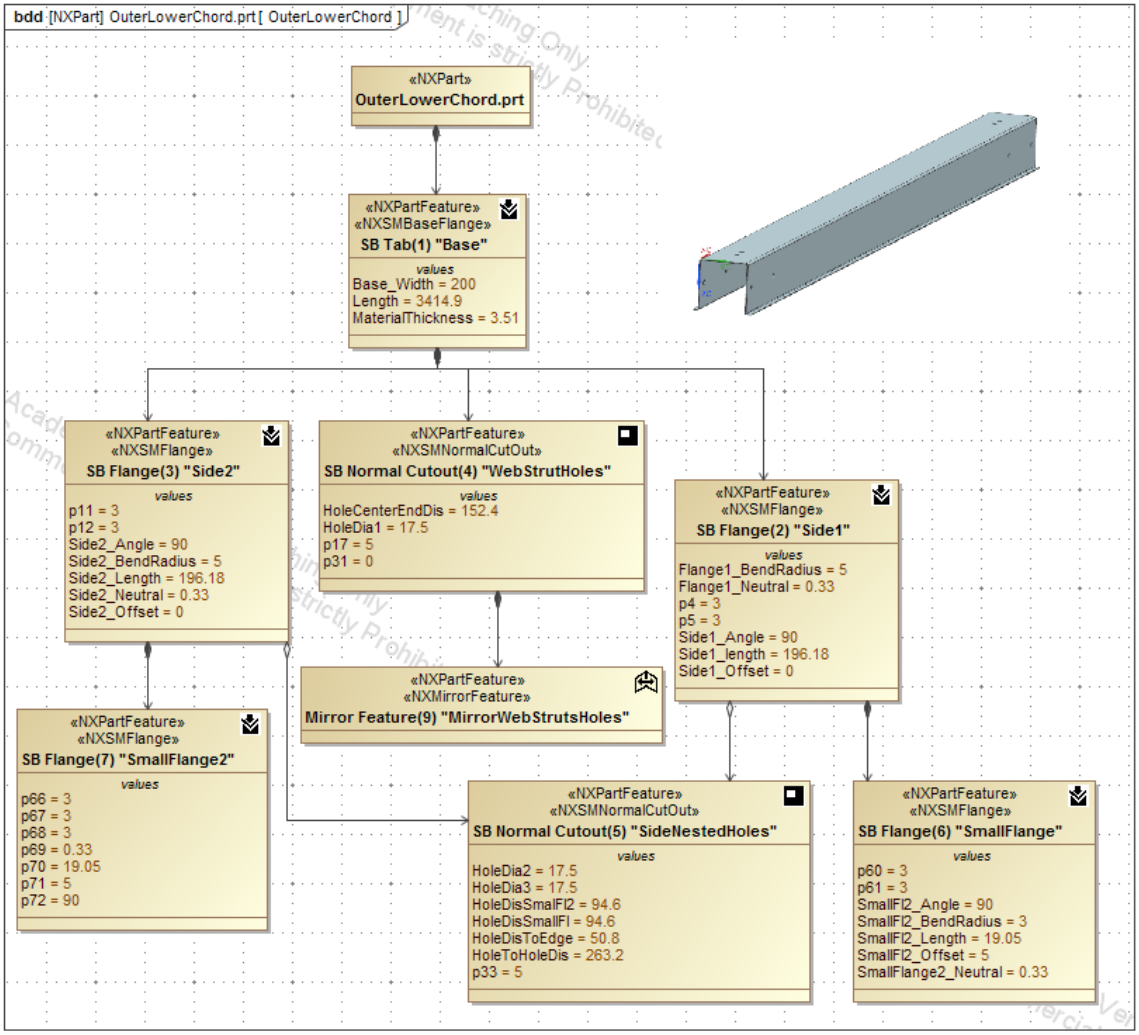


Figure 90: Outer Lower Chord: feature-based decomposition

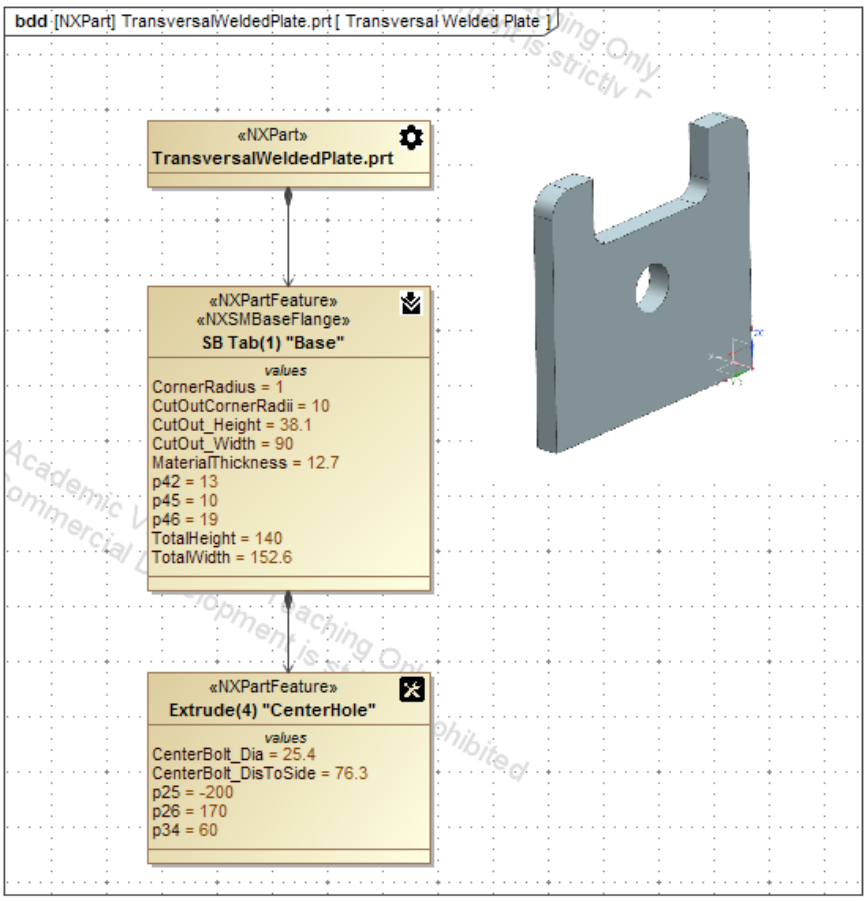


Figure 91: Transversal Welded Plate: feature-based decomposition

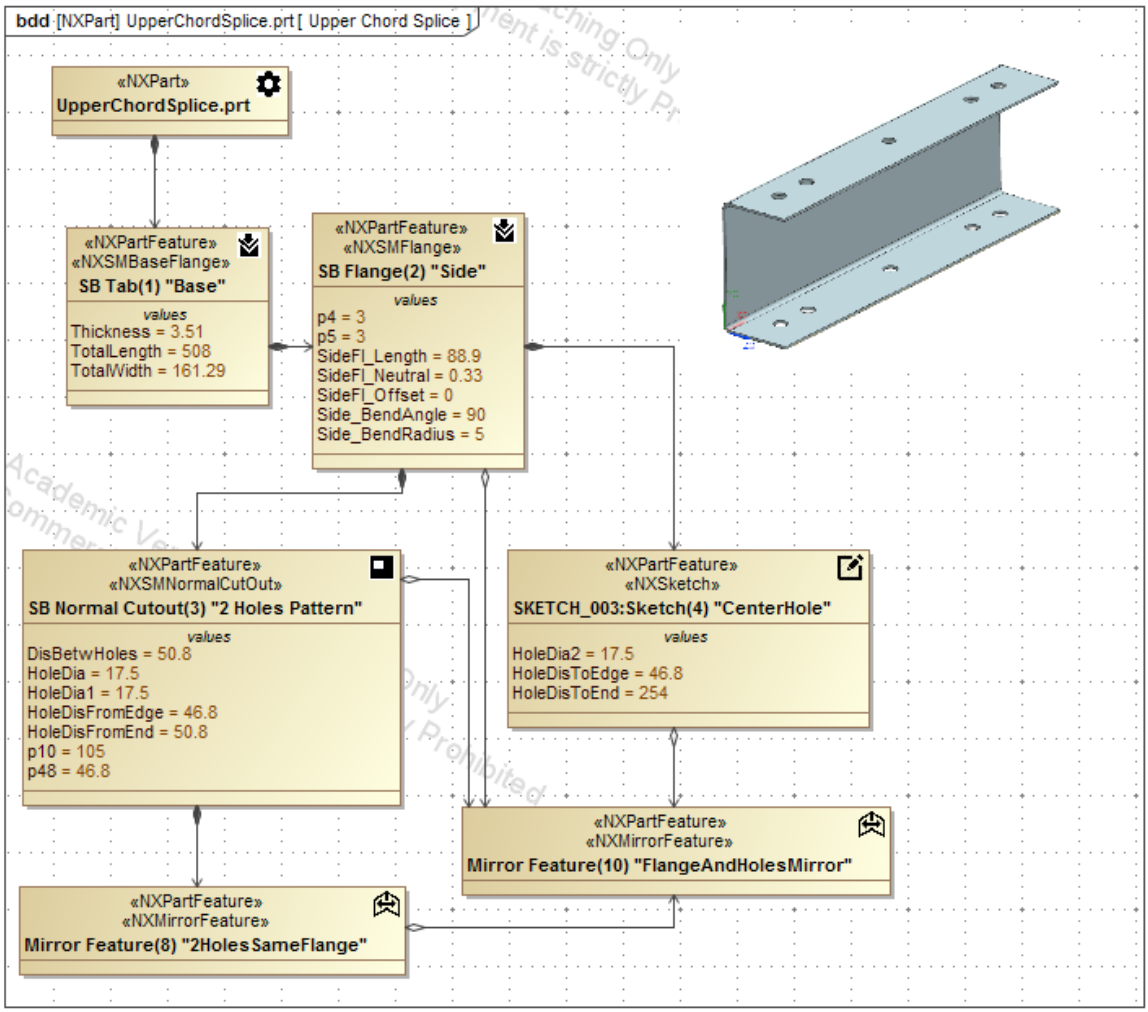


Figure 92: Upper Chord Splice: feature-based decomposition.

7.1.2. Knowledge Allocation of the Four Components of the Studied Assembly

The following pictures show the results of the knowledge allocation stage. In this section a picture with a Knowledge Allocation Matrix and a picture with the representation in context of the feature-linked manufacturing specifications and design specifications will be given for each of the four components of the assembly.

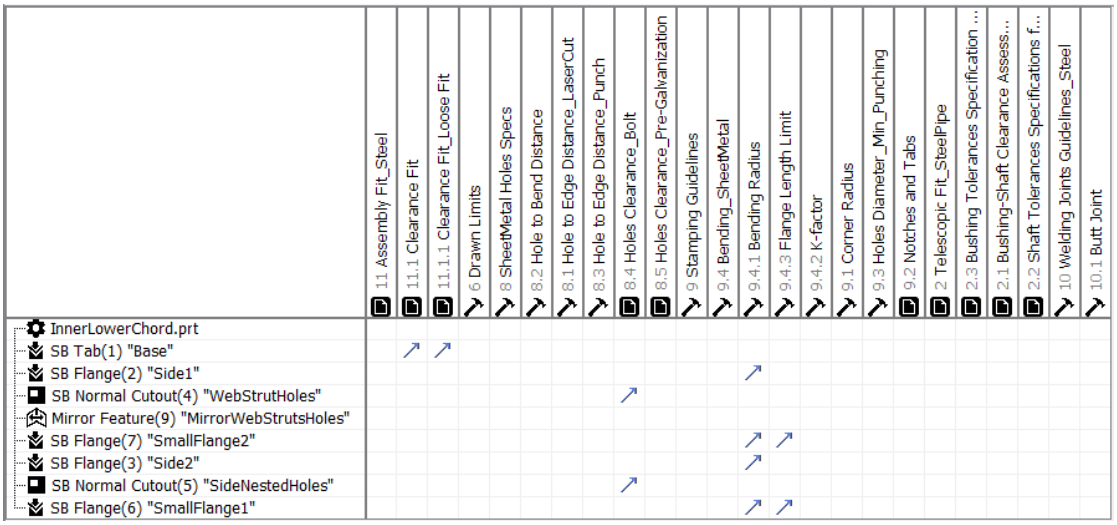


Figure 93: Inner Lower Chord Knowledge Allocation Matrix

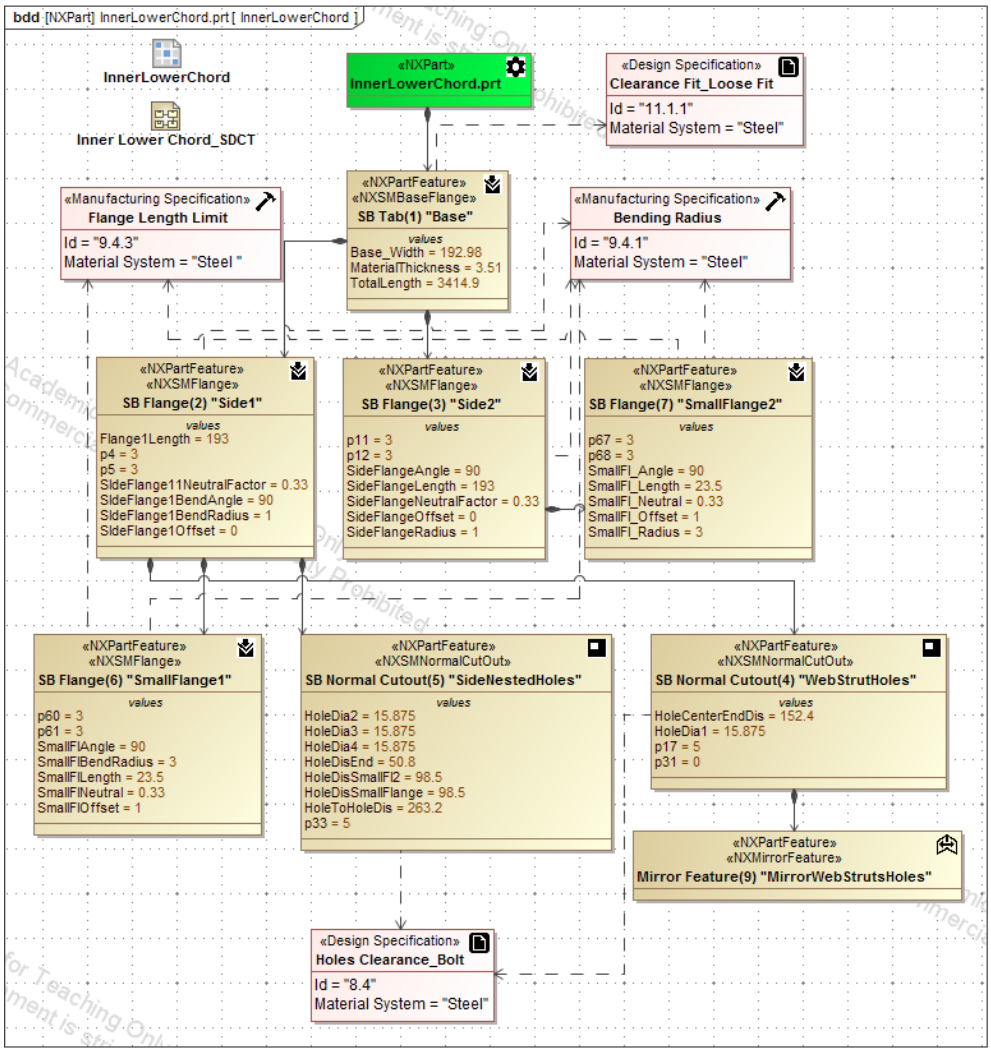


Figure 94: In-context Knowledge Allocation: Inner Lower Chord

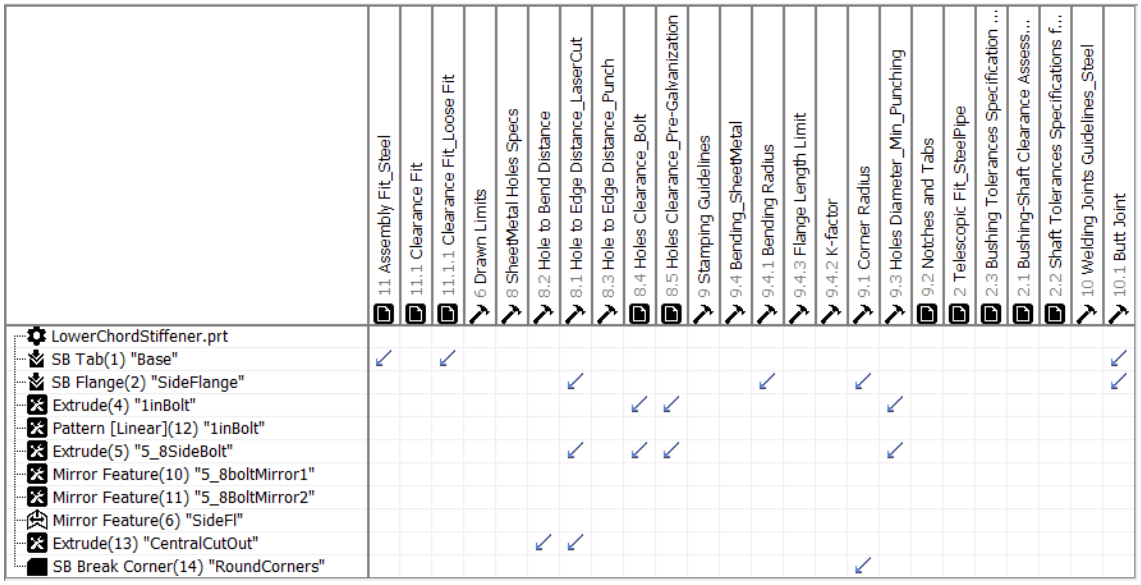


Figure 95: Lower Chord Stiffener Knowledge Allocation Matrix

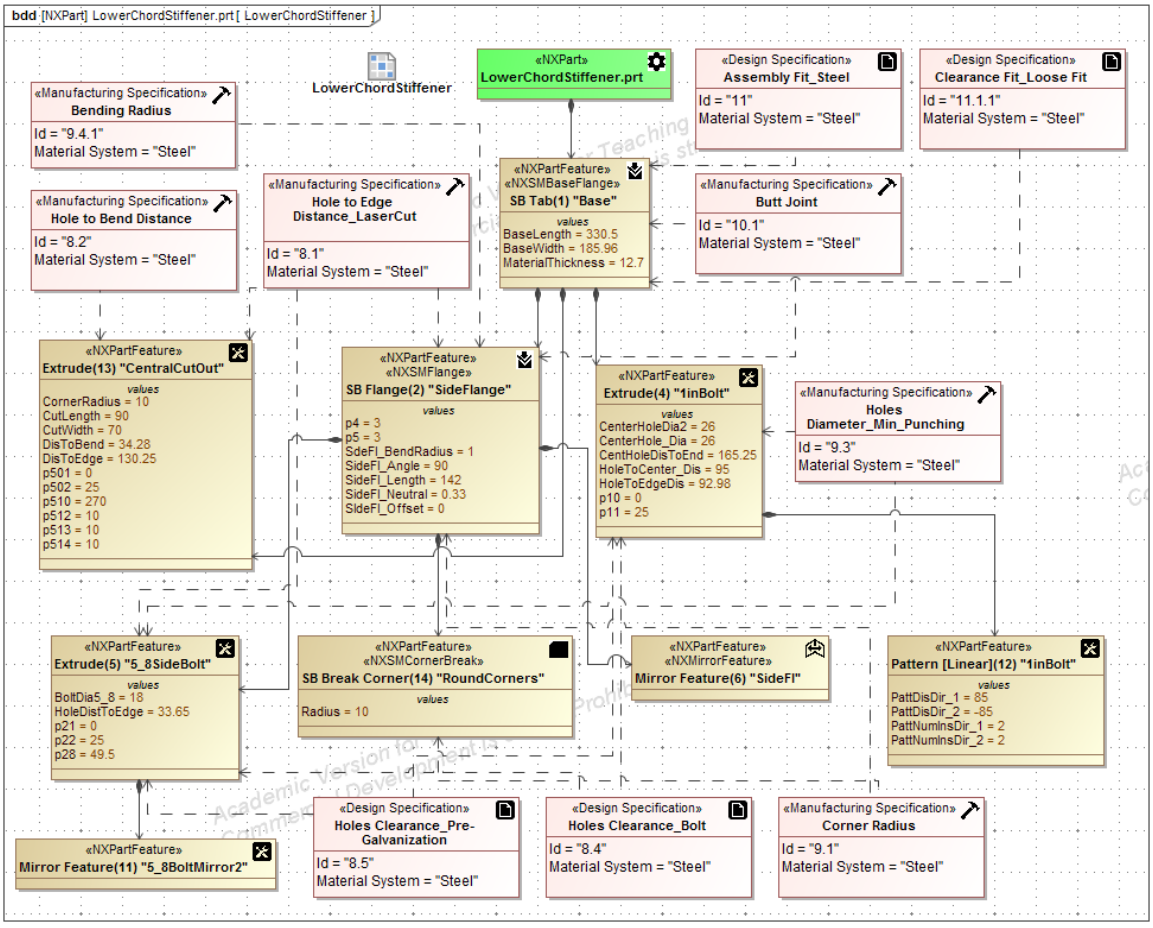


Figure 96: Lower Chord Stiffener with allocated manufacturing and design specifications

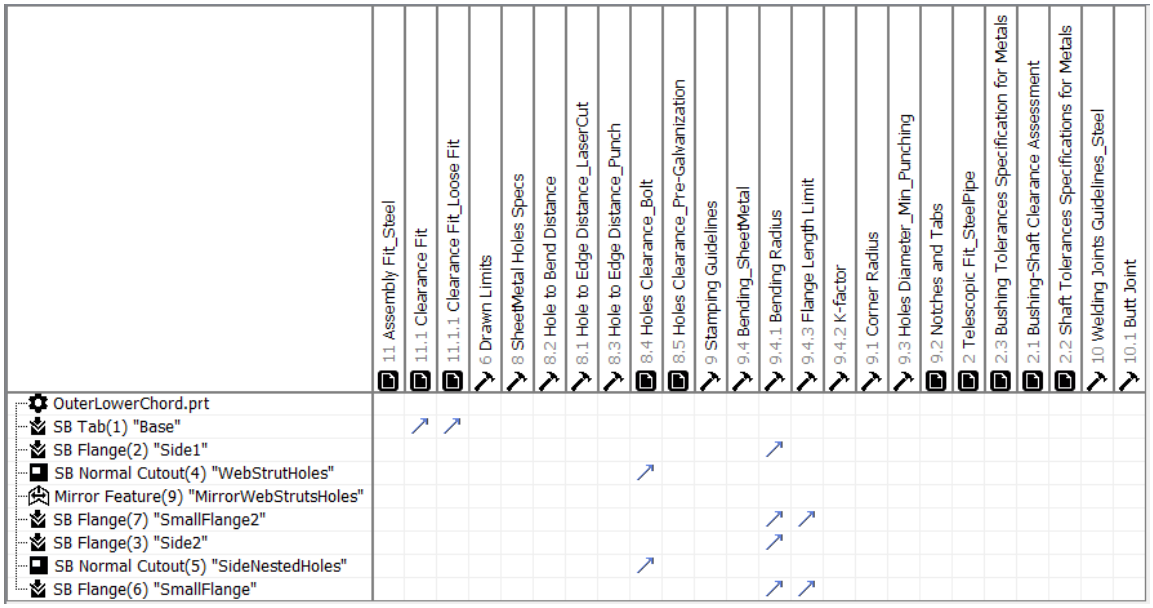


Figure 97: Outer Lower Chord Knowledge Allocation Matrix

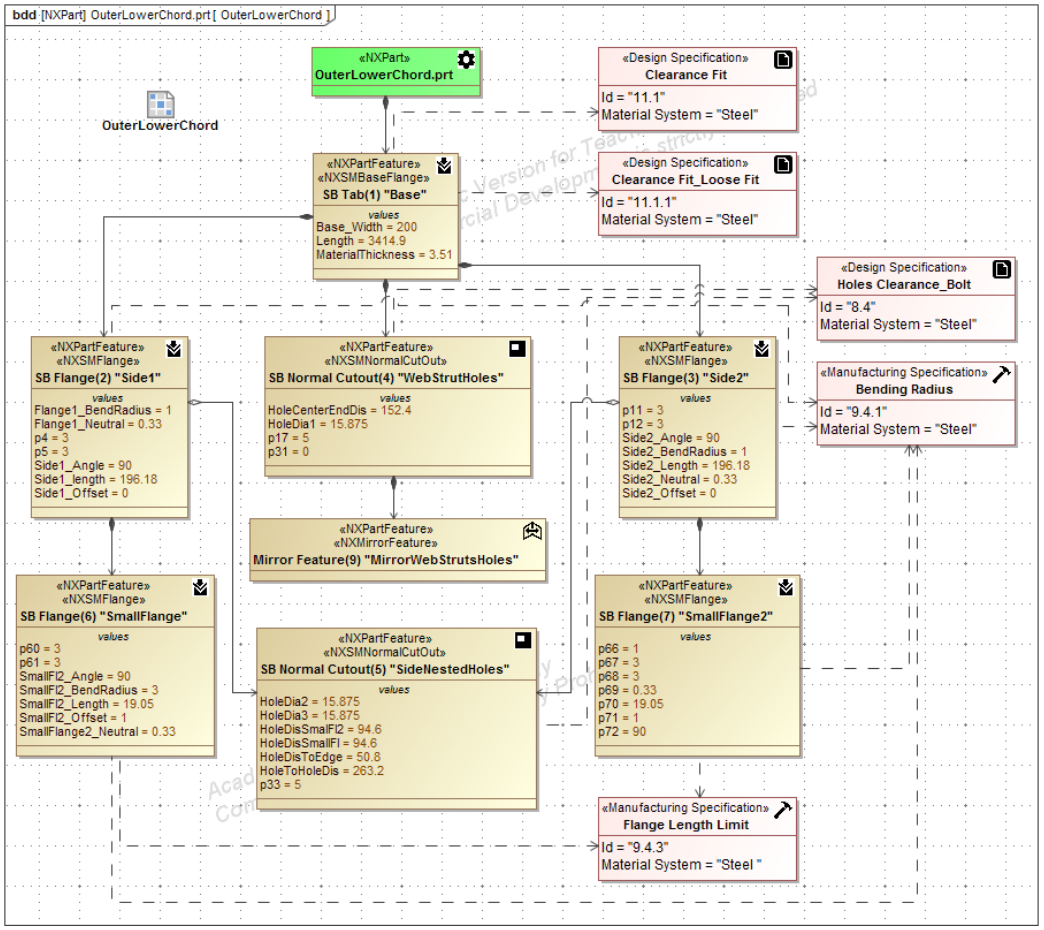


Figure 98: In-context Knowledge Allocation Outer Lower Chord

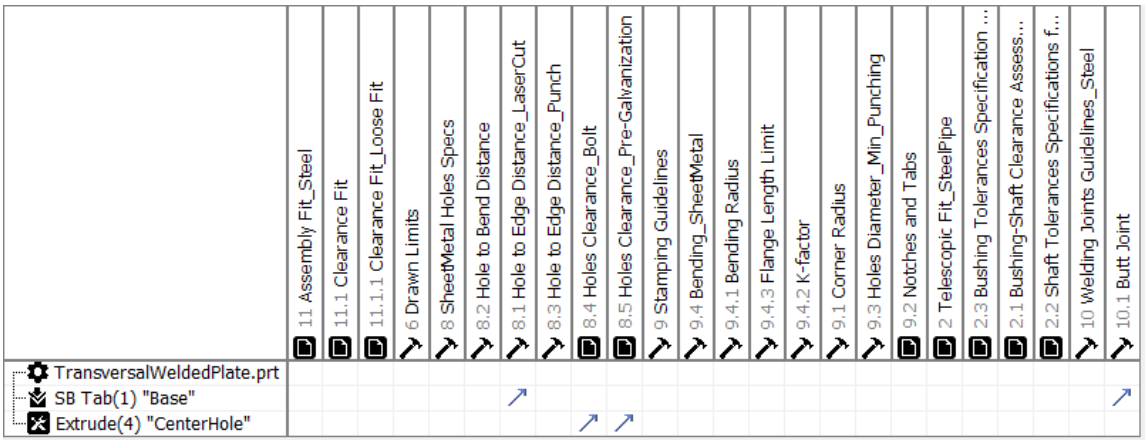


Figure 99: Transversal Welded Plate Knowledge Allocation

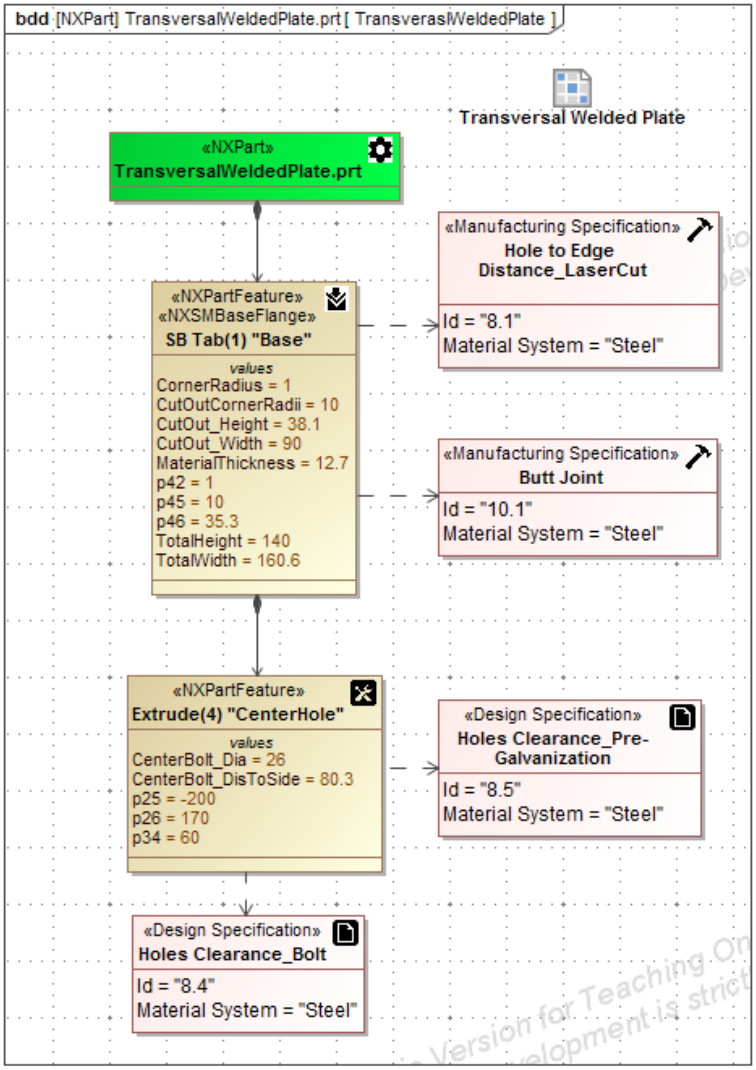


Figure 100: Transversal Welded Plate: with allocated manufacturing and design specifications

7.1.3. Parametric Executions of SDCT: Manufacturing and Design Specifications

The following figures show the specific analysis context of each of the four components of the QuadPod assembly being studied. For each component, the results of the parametric executions of the SDCT are also presented. For easy visual access to the components of the parametric diagram a simple color code is used. At the left, in a light yellow-brown color, are the imported CAD values and features needed for the parametric calculations. At the center, in a bright yellow color, the knowledge-based constraints and critical dimensions are connected to the CAD parameters through binding connectors specified as “real.” At the right, in a dark brown color, are the <<Target Value Property>> elements and in a blue color, the <<Validation Value Property>> elements. These components are also connected to the knowledge-based constraints by means of binding connectors, which are typed as real for numeric values and Boolean for validation values.

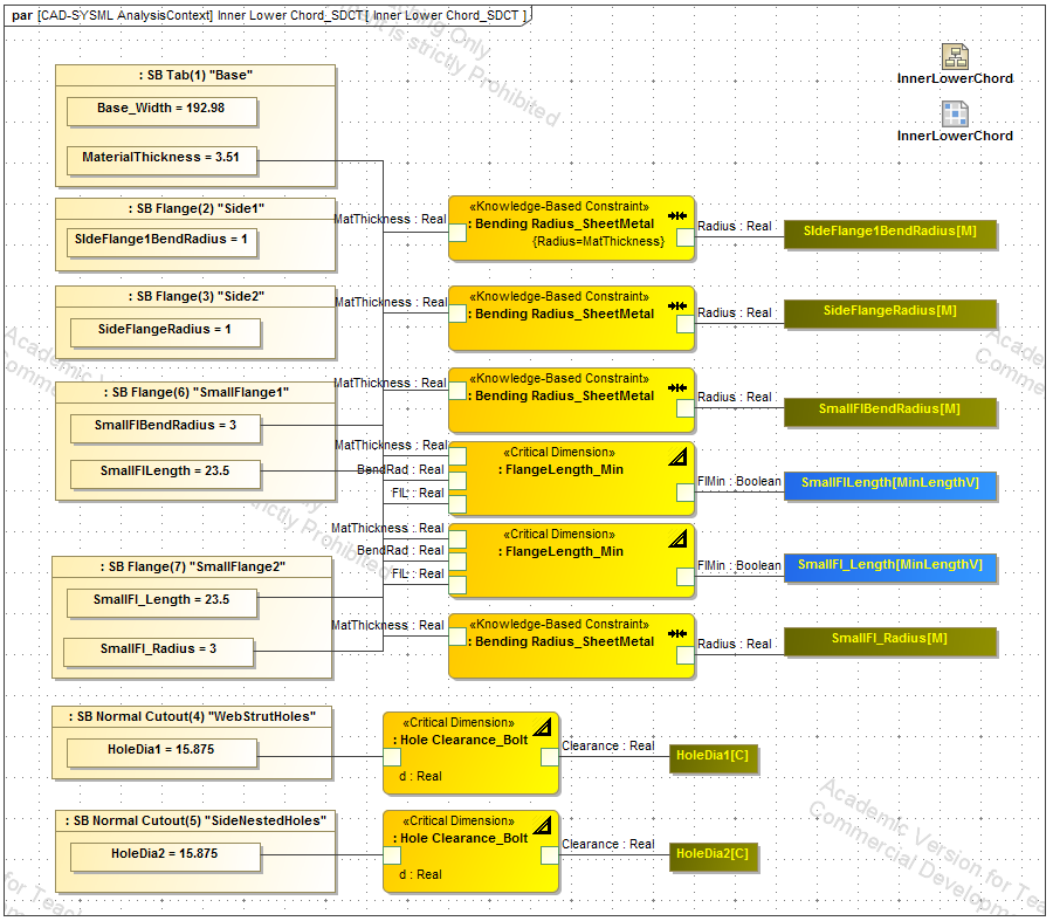


Figure 101: Parametric execution analysis context for SDCT Inner Lower Chord

For all the parametric execution results tables, custom icons have been created for easy visual access during results evaluation. Target values will later replace the default values imported from the CAD model for a final knowledge-based geometry update. At this point of the analysis, target values have been evaluated for knowledge and manufacturing compliance.

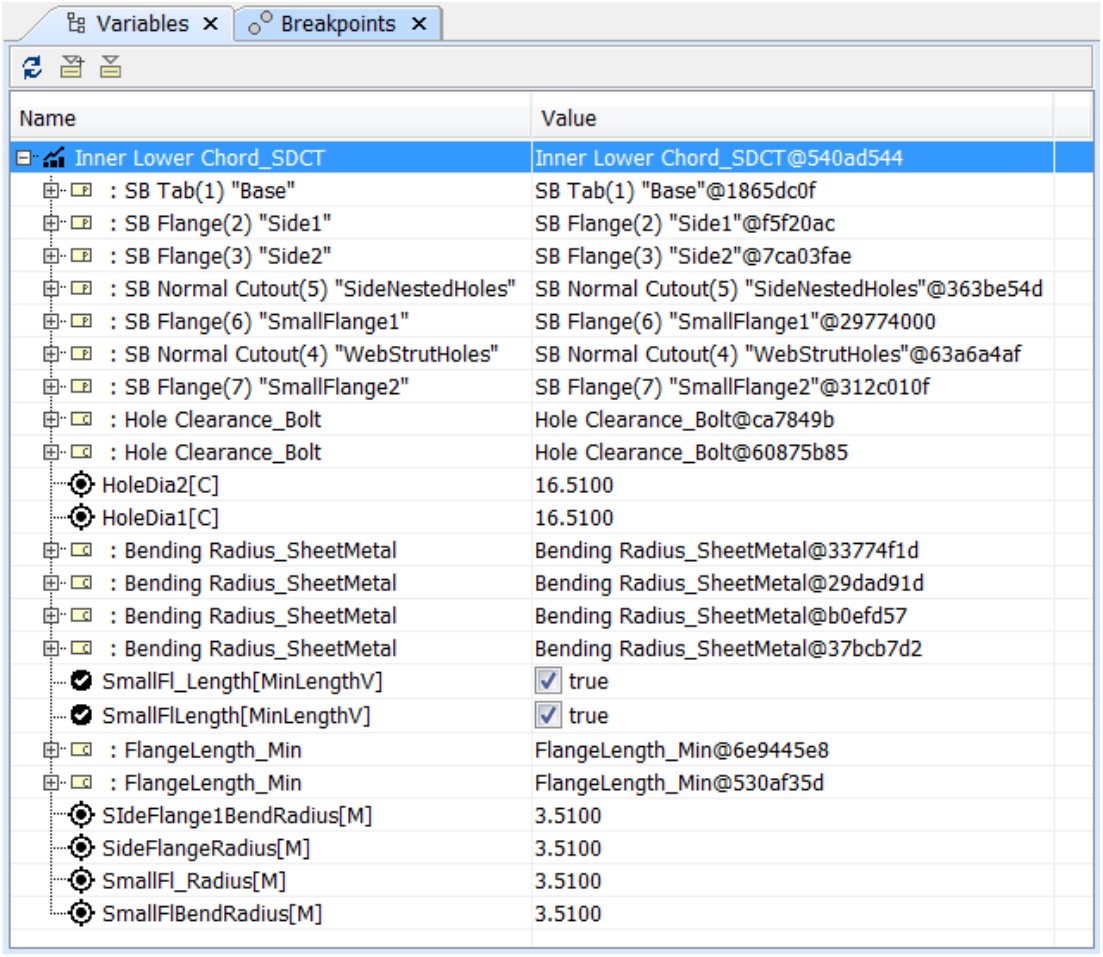


Figure 102: SDCT parametric execution results for Inner Lower Chord

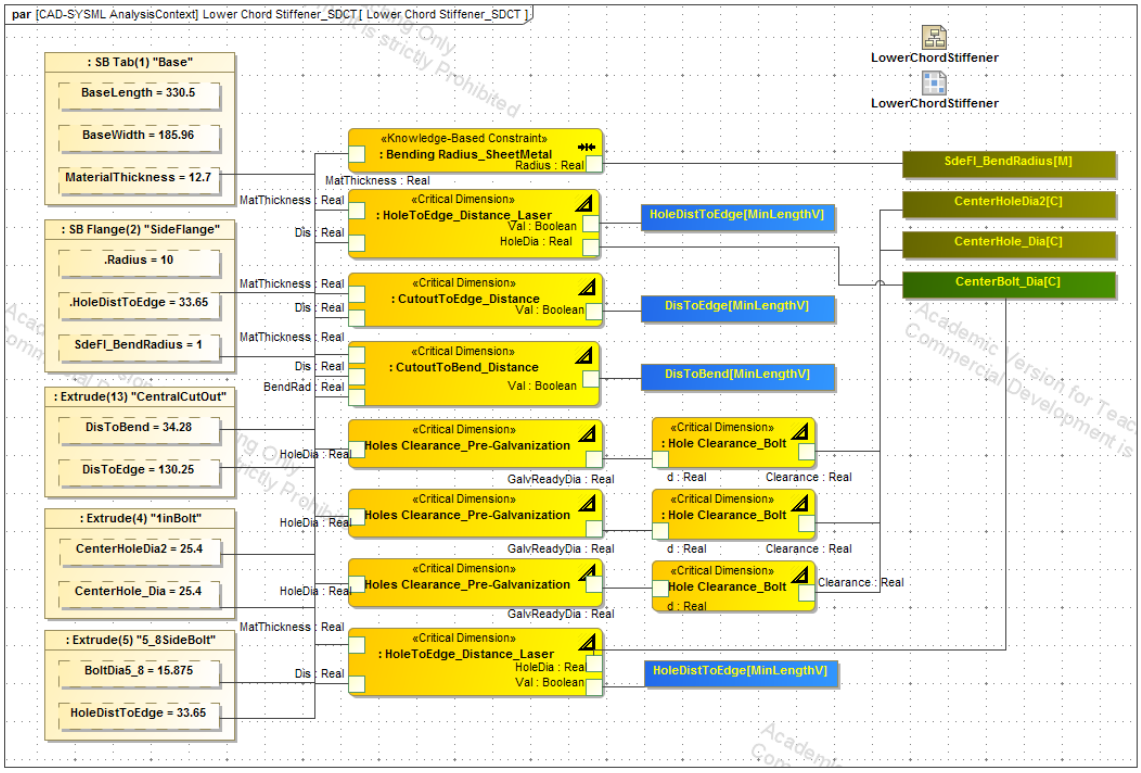


Figure 103: Parametric execution analysis context for SDCT Lower Chord Stiffener

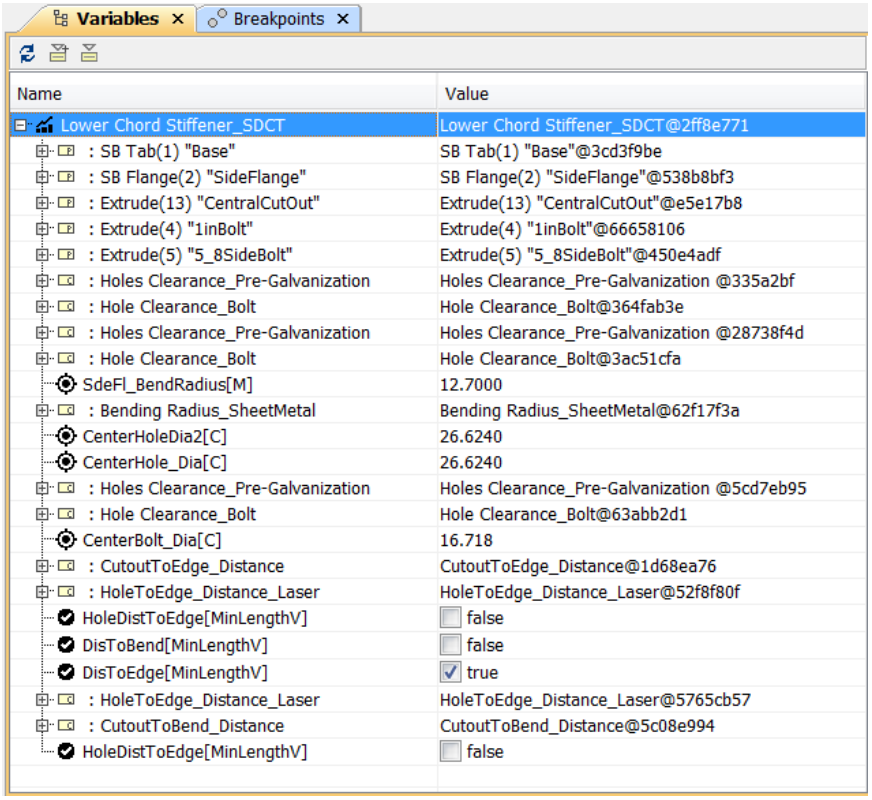


Figure 104: SDCT parametric execution results for Lower Chord Stiffener (failed)

Figure shows a parametric execution where some validation values have not been met. In this case, the user will check the names of the validation values that did not pass specification verifications (e.g. HoleDisToEdge) and search for the constraint with a similar name to verify what values are not in compliance with the specification.

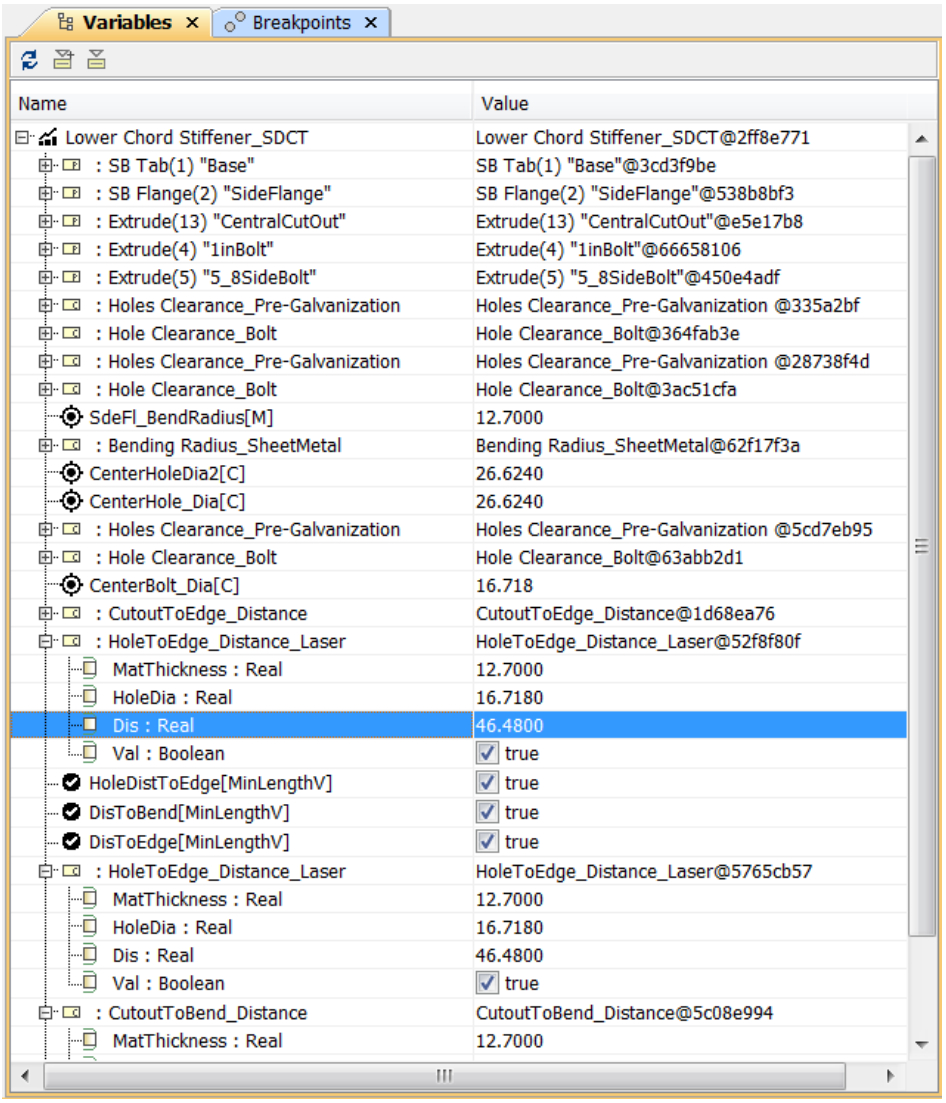


Figure 105: SDCT parametric execution results for Lower Chord Stiffener (passed)

These situations must be verified by the user, as some specifications changes could potentially create a conflict in the CAD model. For instance, a hole feature that is too close to the edge will not pass the HoleDisToEdge verification. However, if the

feature were moved to comply with that rule, it would potentially create an alignment mismatch with a feature that belongs to a different component of the assembly. In the Figure , the values have been reviewed and the part is now in compliance with all manufacturing rules.

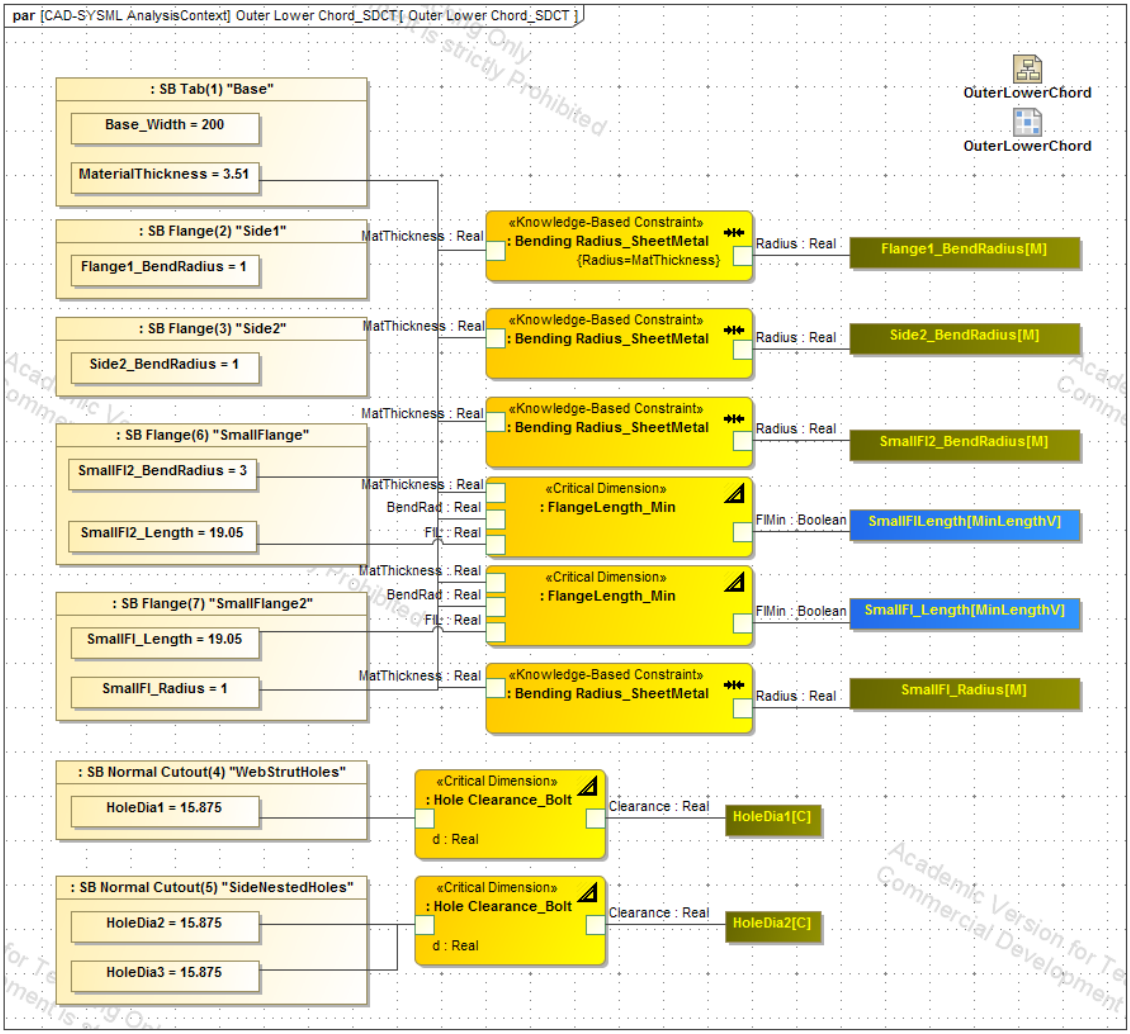


Figure 106: Parametric execution analysis context for SDCT Outer Lower Chord

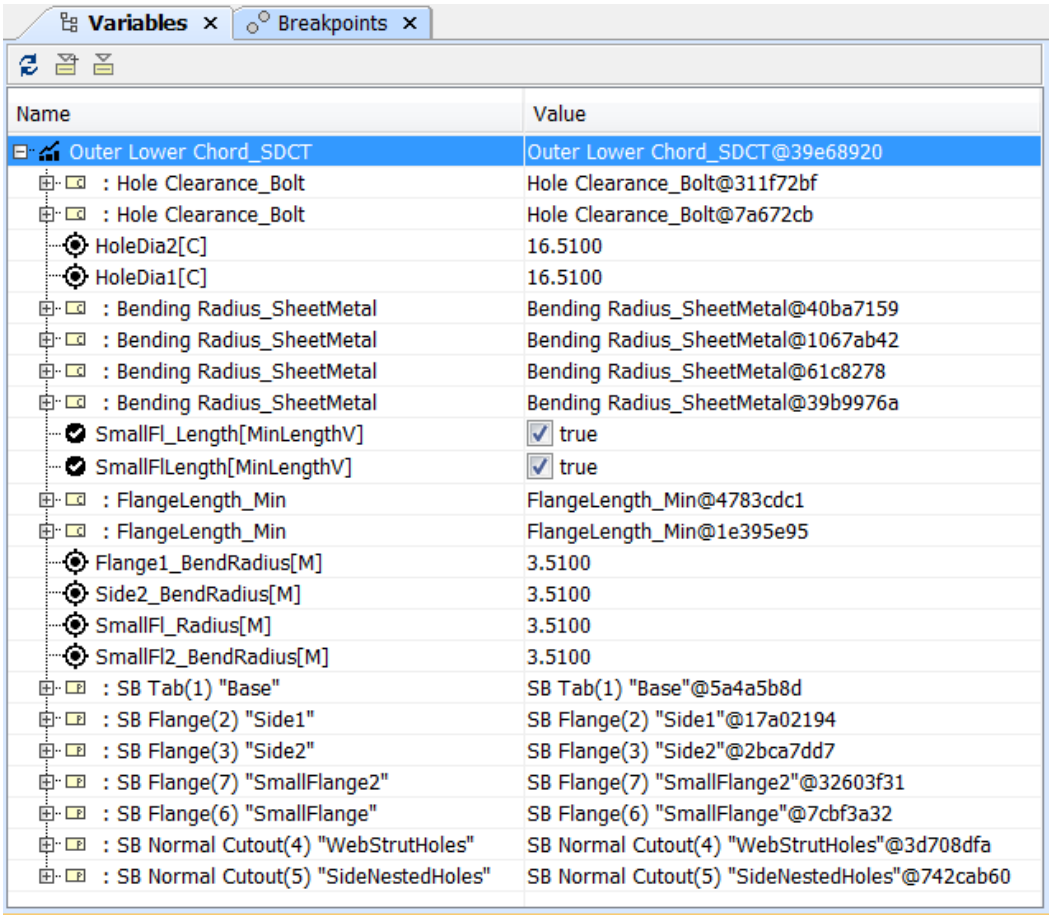


Figure 107: SDCT parametric execution results for Outer Lower Chord

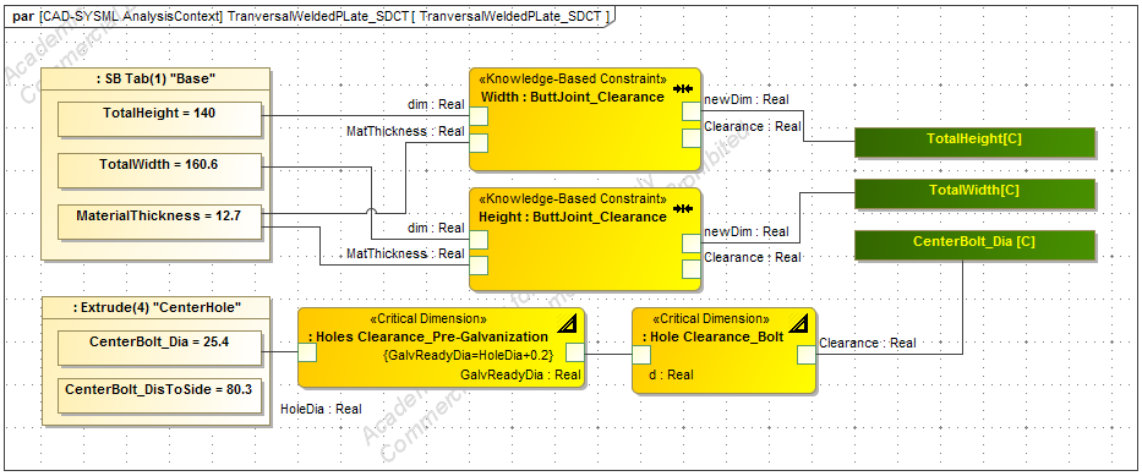


Figure 108: Parametric execution analysis context for SDCT Transversal Welded Plate

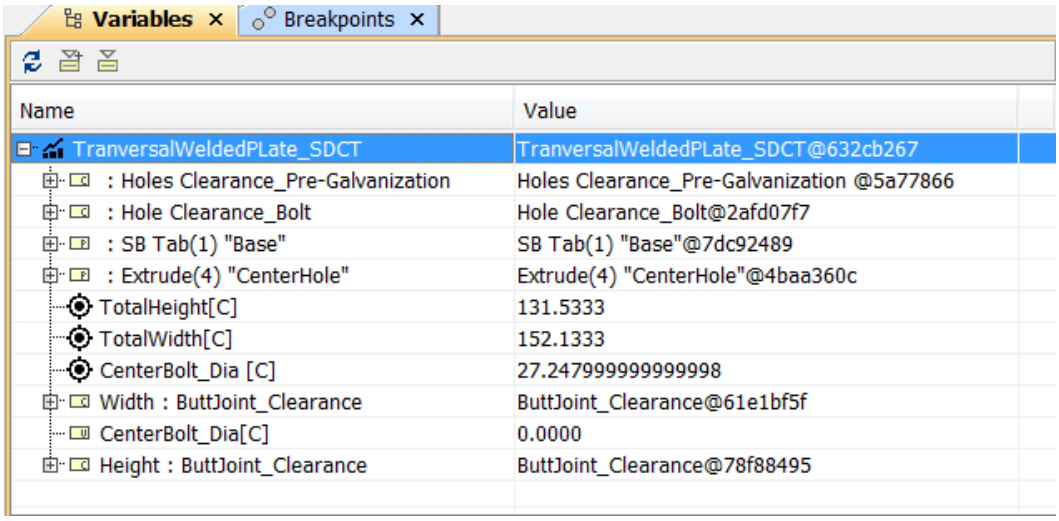


Figure 109: SDCT parametric execution results for Transversal Welded Plate (passed)

7.1.4. QuadPod Node Assembly HCT evaluation

Figure depicts a general description of the QuadPodNode_Assembly being studied. After the SDCT analyses have taken place, all components (NXPart) will be brought together to analyze clearances and assembly tolerances by means of an HCT evaluation.

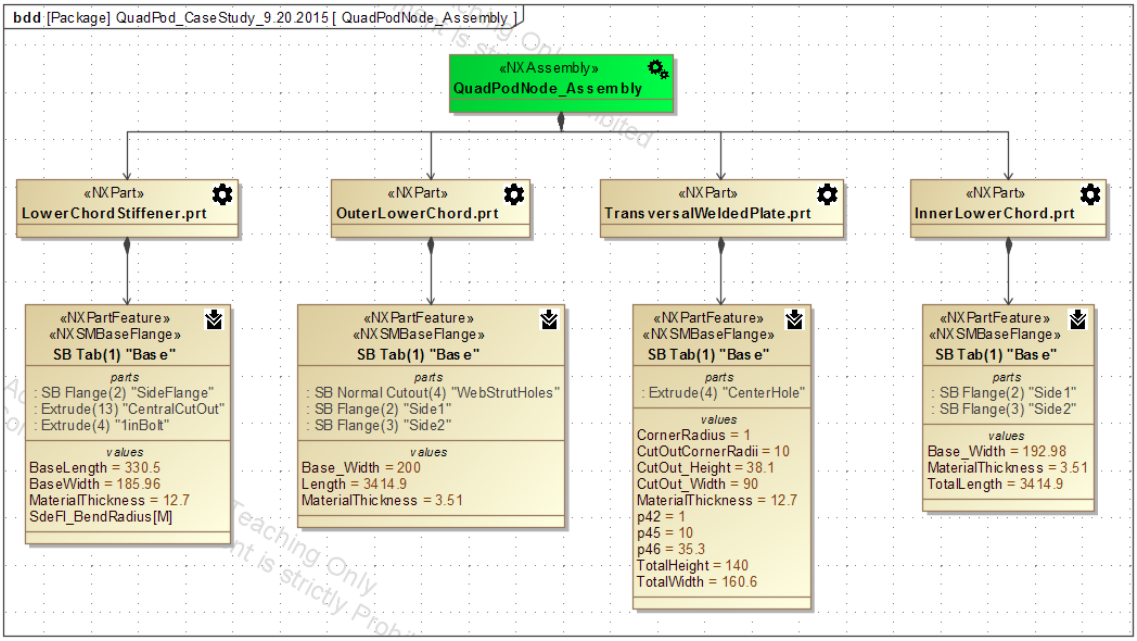


Figure 110: Feature-based decomposition: QuadPod: assembly level

As depicted at the far left in Figure , features and parameters from all four components of the assembly are included in this clearances and tolerances assessment of multi-nested parts. As seen at the far left side of the diagram, some parameters shown in a dark brown color are also inputs. These elements are the products of previous SDCT calculations.

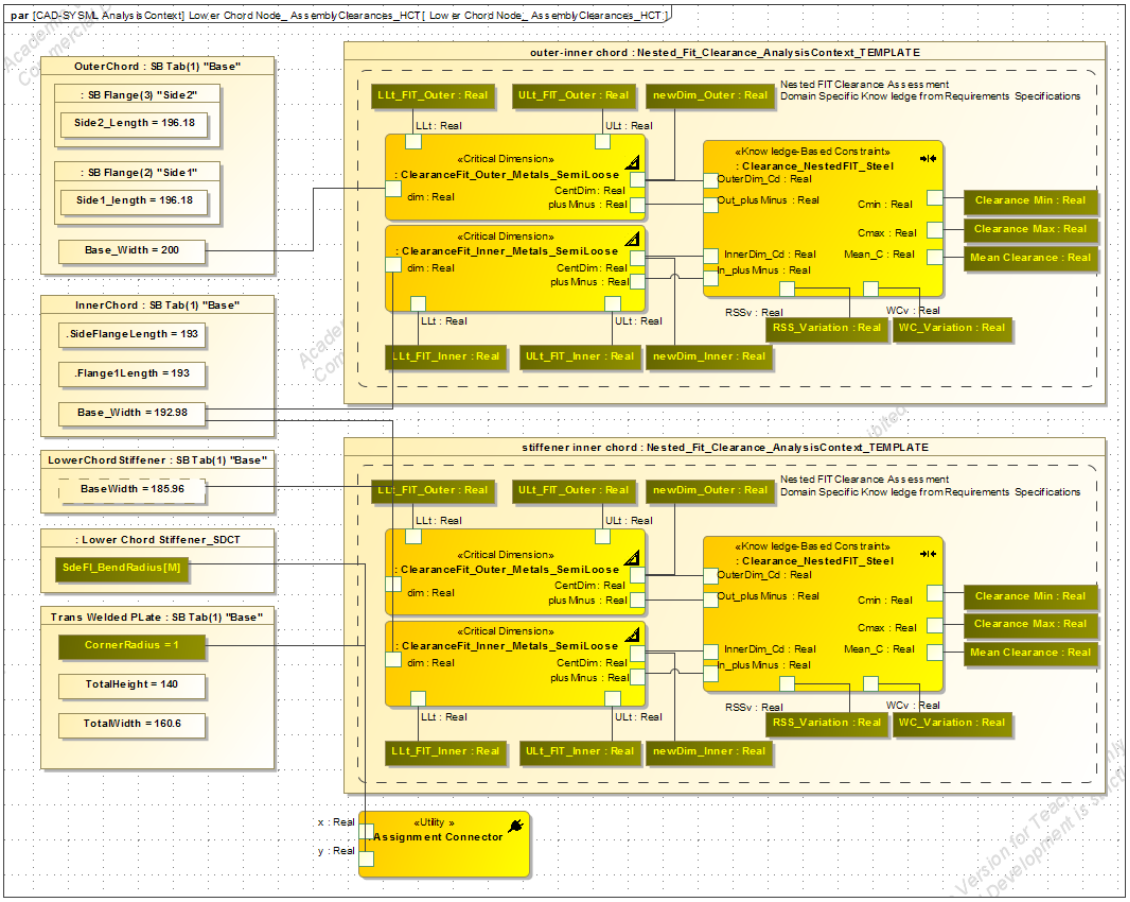


Figure 111: Parametric execution analysis context for assembly clearances: HCT for QuadPod assembly

This scenario exemplifies how the presented approach can accommodate nested analyses. It should also be noted that the "outer-inner chord: Nested_Fit_Clearance_AnalysisContext_TEMPLATE" has been framed as a single element with all its internal structure shown as a white box. This approach facilitates the

allocation of repetitive analyses and also enables an easy binding procedure by showing all internal ports of nested knowledge-based constraints.

Figure depicts the results of clearances and tolerances allocations for the entire assembly, although no target or validation values are visible. This situation occurs when all analyses contained in the parametric diagram are actually nested from other analyses or pasted as independent white boxes. However, in the red box in Figure , three nested analyses are shown. Accessing the useful data of these analyses is not a mandatory step. Alternatively, if the user follows the suggested process and applies the custom commands to commit the analyses results back to the CAD file, no human inspection will be required. Yet, if the user needs to inspect the evaluated parameters, double clicking in any of these nested analyses will display the hidden data.

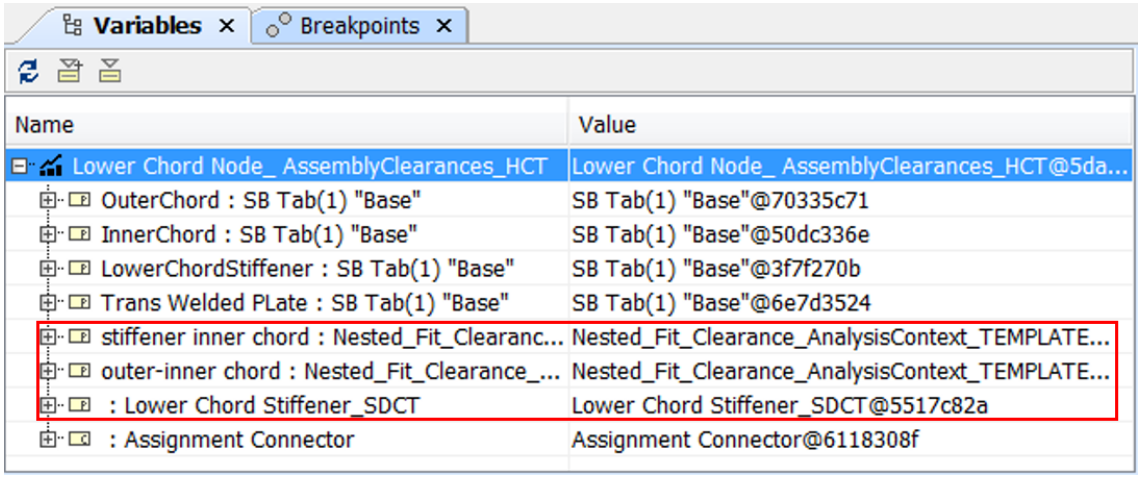


Figure 112: Lower Chord Node_AssemblyClearances_HCT general analysis results

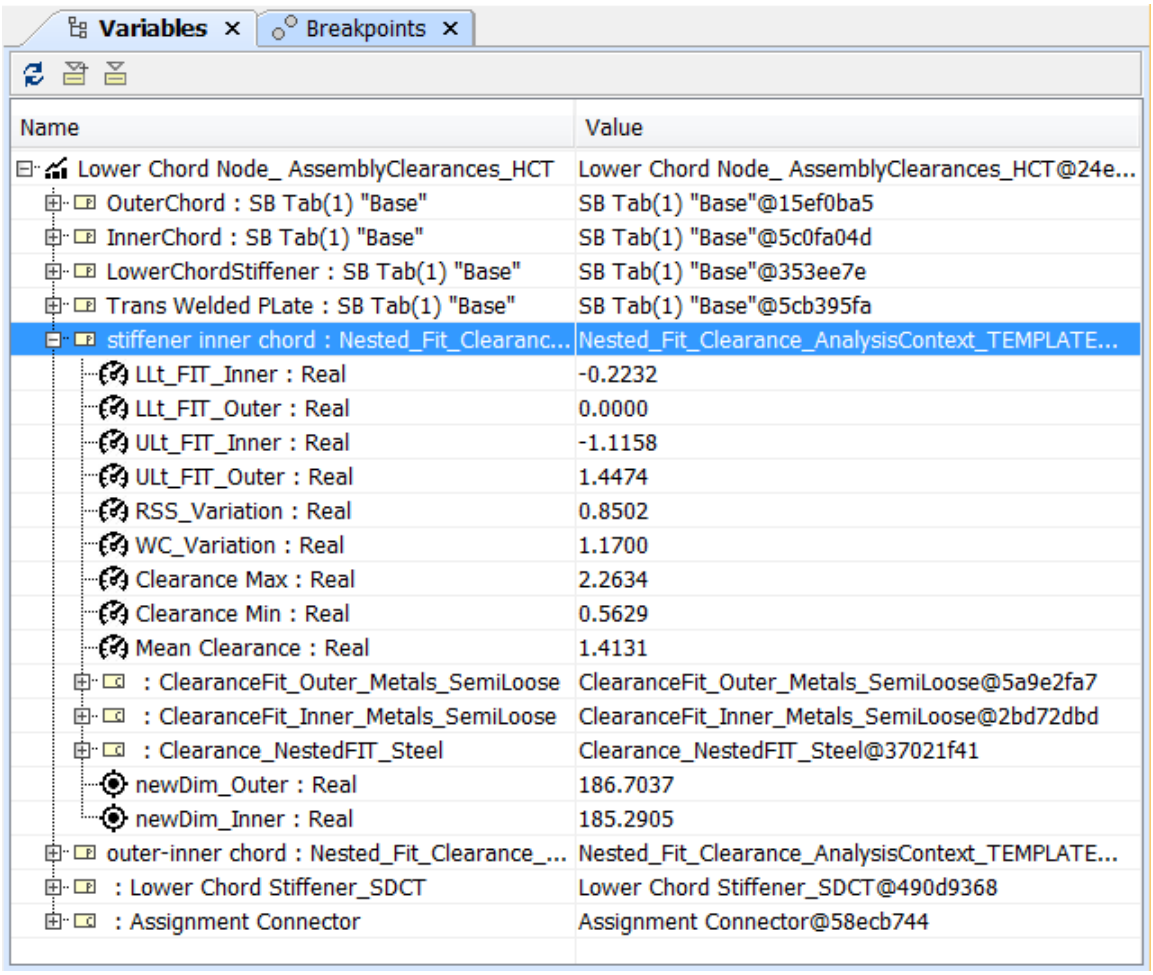


Figure 113: Expanded Inner Chord - Lower Chord Stiffener analysis results

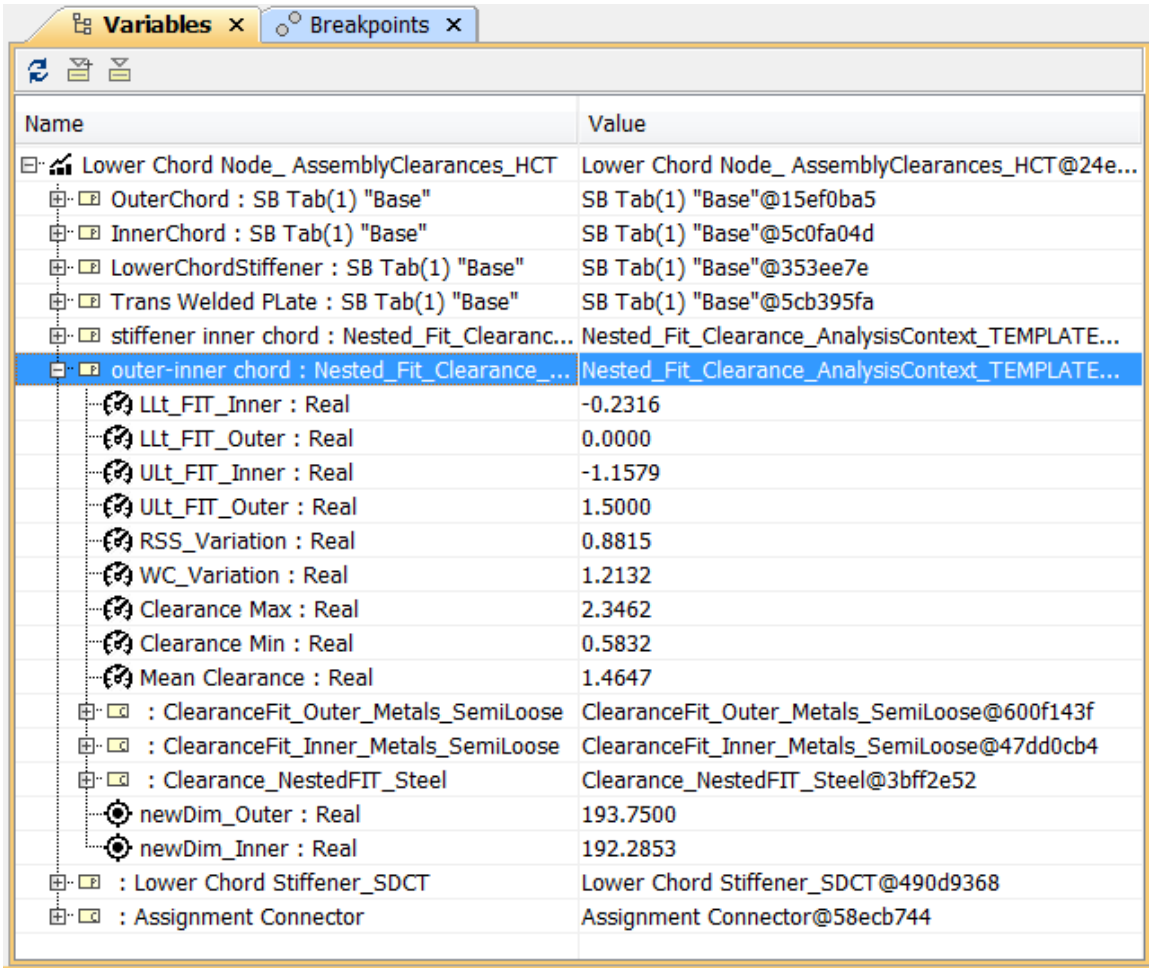


Figure 114: Expanded Inner Chord - Outer Chord analysis results

7.1.5. QuadPod Node Assembly: Feature-Based Components Update Based on Analyses Results

The following tables show the comparison of CAD parameters values before and after the knowledge-based manufacturing and design specifications analyses.

Table 4: Outer Lower Chord CAD update results

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	Base_Width	200	200	193.75+7 = 200.75
2	Flange1_BendRadius	1	1	3.51
3	HoleCenterEndDis	6*25.4	152.4	
4	HoleDia1	Hole_Dia	15.875	16.51
5	HoleDia2	Hole_Dia	15.875	16.51
6	HoleDia3	Hole_Dia	15.875	16.51
7	HoleDisSmalFl2	HoleDisSmallFl	94.6	
8	HoleDisSmallFl	94.6	94.6	
9	HoleDisToEdge	Hole_X_Offset	50.8	
10	HoleToHoleDis	263.2	263.2	
11	Hole_Dia	15.875	15.875	16.51
12	Hole_X_Offset	50.8	50.8	
13	Length	3317.638+101.6-4.33	3414.85	
14	MaterialThickness	3.51	3.51	
15	SM_Validation_MIN_Punch_Tool_Cle	5	5	
16	SM_Validation_MIN_WEB_LENGTH	5	5	
17	Sheet_Metal_Bend_Radius	3	3	3.51
18	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
19	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
20	Sheet_Metal_Material_Thickness	3	3	
21	Sheet_Metal_Neutral_Factor	0.33	0.33	
22	Sheet_Metal_Relief_Depth	3	3	
23	Sheet_Metal_Relief_Width	3	3	
24	[degrees]Side1_Angle	90	90	
25	Side1_Offset	0	0	
26	Side1_length	196.18	196.18	
27	[degrees]Side2_Angle	90	90	
28	Side2_BendRadius	1	1	3.51
29	Side2_Length	196.18	196.18	
30	Side2_Offset	0	0	
31	[degrees]SmallFl2_Angle	90	90	
32	SmallFl2_Length	19.05	19.05	
33	SmallFl2_BendRadius	1	1	3.51
34	[degrees]SmallFl_Angle	90	90	
35	SmallFl_Length	19.05	19.05	
36	SmallFl_Radius	1	1	3.51

Table 5: Lower Chord Stiffener CAD update results

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	BaseWidth	185.96	185.96	185.2905
2	BoltDia5_8 (CenterBolt_Dia)	15.875	15.875	16.718
3	CentHoleDisToEnd	165.25	165.25	
4	CenterHoleDia2	25.4	25.4	26.624
5	CenterHole_Dia	25.4	25.4	26.624
6	CornerRadius	10	10	
7	CutLength	90	90	
8	CutWidth	70	70	
9	DisToBend	34.28	34.28	
10	DisToEdge	130.25	130.25	
11	HoleDisToBend	49.5	49.5	
12	HoleToCenter_Dis	95	95	
13	HoleToEdgeDis	BaseWidth/2	92.98	
14	MaterialThickness	12.7	12.7	
15	PattDisDir_1	85	85	
16	PattDisDir_2	-85	-85	
17	PattNumInsDir_1	2	2	
18	PattNumInsDir_2	2	2	
19	Radius	10	10	
20	SIdeFl_Offset	0.0 // Used By ...	0	
21	SM_Validation_MIN_Punch_Tool_Cl	5	5	
22	SM_Validation_MIN_WEB_LENGTH	5	5	
23	SdeFl_BendRadius	1 // Used By ...	1	12.7
24	Sheet_Metal_Bend_Radius	3	3	
25	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
26	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
27	Sheet_Metal_Material_Thickness	3	3	
28	Sheet_Metal_Neutral_Factor	0.33	0.33	
29	Sheet_Metal_Relief_Depth	3	3	
30	Sheet_Metal_Relief_Width	3	3	
31	[degrees]SideFl_Angle	90.0 // Used By ...	90	
32	SideFl_Length	142	142	
33	SideFl_Neutral	Sheet_Metal_Neutral_Factor // Used By ...		

Table 6: Inner Lower Chord CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	Base_Width	192.98	192.98	186.7037 + 7 = 193.7037
2	Flange1Length	193	193	
3	HoleCenterEndDis	6*25.4	152.4	
4	HoleDia1	Hole_Dia	15.875	16.51
5	HoleDia2	Hole_Dia	15.875	16.51
6	HoleDia3	Hole_Dia	15.875	16.51
7	HoleDisEnd	Hole_X_Offset	50.8	
8	HoleDisSmallFl2	HoleDisSmallFlange	98.5	
9	HoleDisSmallFlange	98.5	98.5	
10	HoleToHoleDis	263.2	263.2	
11	Hole_Dia	15.875	15.875	16.51
12	Hole_X_Offset	50.8	50.8	
13	MaterialThickness	3.51	3.51	
14	[degrees]SIdeFlange1BendAngle	90	90	
15	SIdeFlange1BendRadius	1	1	3.51
16	SIdeFlange1Offset	0	0	
17	Sheet_Metal_Bend_Radius	3	3	
18	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
19	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
20	Sheet_Metal_Material_Thickness	3	3	
21	Sheet_Metal_Neutral_Factor	0.33	0.33	
22	Sheet_Metal_Relief_Depth	3	3	
23	Sheet_Metal_Relief_Width	3	3	
24	[degrees]SideFlangeAngle	90	90	
25	SideFlangeLength	193	193	
26	SideFlangeOffset	0	0	
27	SideFlangeRadius	1	1	3.51
28	[degrees]SmallFlAngle	90	90	
29	SmallFlLength	23.5	23.5	
30	SmallFlOffset	1	1	
31	SmallFlBendingRadius	1	1	3.51
32	SmallFl_Length	23.5	23.5	
33	SmallFl_Radius	1	1	3.51
34	TotalLength	3317.638+101.6-4.338	3414.85	

Table 7: Transversal Welded Plate CAD update results

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	CenterBolt_DisToSide	80.3	80.3	
2	CornerRadius	1	1	16.7
3	CutOutCornerRadii	10	10	
4	CutOut_Height	25.4*1.5	38.1	
5	CutOut_Width	90	90	
6	MaterialThickness	12.7	12.7	
7	Sheet_Metal_Bend_Radius	3	3	
8	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
9	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
10	Sheet_Metal_Neutral_Factor	0.33	0.33	
11	Sheet_Metal_Relief_Depth	3	3	
12	Sheet_Metal_Relief_Width	3	3	
13	TotalHeight	140	140	131.5333
14	TotalWidth	160.6	160.6	152.1333
15	CenterBolt_Dia	25.4	25.4	27.248

Figure shows the before and after knowledge-based manufacturing and design specifications allocations. Before analyses, all geometry is modeled with nominal dimensions such as sizes for bolts and thicknesses and nominal fits for assemblies. The following list offers a summary of the material-specific design and manufacturing constraints automatically applied to the CAD geometry for this case study:

- SDCT hole clearances for all bolts sizes were applied
- SDCT coating hole clearances for all hot-dip galvanized components were applied
- SDCT bending radii were calculated based on material thickness and geometry context
- SDCT contours of nested shapes were calculated to avoid clashes
- SDCT cut-to-edge distances for cutout features were validated
- SDCT cut-to-bend distances for cutout features were validated
- HCT clearances were calculated and applied for all nested components
- HCT tolerances of clearances based on RSS, WC, were calculated

- HCT plus/minus, mean, and centered tolerances were calculated for all assembly clearances

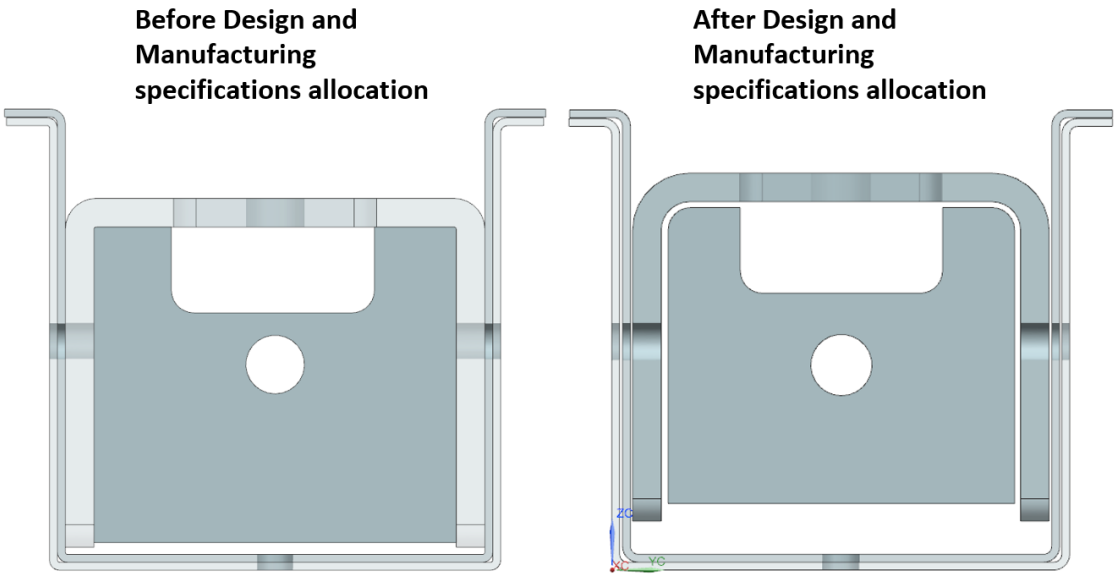


Figure 115: Before and after design and manufacturing knowledge analysis and allocation

Figure 116: Studied QuadPod assembly after fabrication and erection

Figure 117: Final result of QuadPod structure that includes the studied parts and assembly

7.2. Case study 3: Multi-material Assembly: Steel frame, Pre-Cast, Cast-in Place, PVC Window

This case study is intended to demonstrate the manufacturing compliance functionality of the developed application in a multi-material assembly. The same case study was previously conducted through a different tolerances analysis method developed by the author. In that study, the main objective of the analysis was to create "virtual, as-built" geometry of assemblies to assess possible scenarios of dimensional variability. To reach that outcome, the previous approach used SolidWorks design tables to specify ranges of variability of critical dimensions based on information about tolerances and clearances. The design-table-based system (Figure) produced five areas of possible variation – nominal dimensions, worst case and best case scenarios based on tolerances specification, and the RSS max and RSS procedures). In addition, for each of the assembly dimensions identified as "critical," an independent random case (n) was applied.

Study Name	Top Track-Header	
Analysis Parameters		
Orientation tolerances	ON	
Normal to origin feature	NO	
Float fasteners and pins	OFF	
TolAnalyst Measurement	(in)	
Nominal Value	31.77	case 1
Minimum Value	31.715	case 2
Maximum Value	31.825	case 3
RSS Minimum	31.716	case 4
RSS Maximum	31.825	case 5

Figure 118: Possible scenarios of variability based on standard tolerances calculations

In the "virtual, as-built" experiment, when CAD geometry was modified by applying tolerances directly as dimensional constraints, some parts kept the "coincidental mate" condition, but parts that changed at least one dimensional attribute did not keep the same level of assembly consistency. This situation generated breaks in the assembly tree definition and, as a result, the entire topology of the assembly failed. In this case, simulations broke the model because certain parametric as-built-like modifications created unfixable topological inconsistencies in the solid model. For example, conflicting mating conditions for the same components or paradoxical operations broke sketches of some features. Another issue with this past experiment was the complexity of the assembly. Most tools and methods for tolerances allocation work very well on single components. However, when numerous groups of object and features are analyzed, the number and complexity of parameters and geometric constraints grows exponentially. Furthermore, the numbers of parameters that a simple assembly can reach and keep consistent is a persistent challenge of the solid modeling domain. [38] explains how a very small detail of precast concrete can yield numerous different parameters and relations (Figure). To keep consistency in this kind of assembly, those parameters and their internal relations must be defined by domain experts. Overlooking this restriction

generates constant failures of the assemblies when one parameter is updated without considering all the domain-specific implications. The semantic validity of a model can only be judged by a domain expert. Incorrect (or "absurd") design situations are obvious to a human viewer, but are amorphous and thus very difficult to identify algorithmically [38]. To overcome the issues learned from the previous approach, for the new methodology, we have created manufacturing-compliant nominal geometry, of heterogeneous material assemblies, based on material-specific knowledge. This approach is expected to provide domain-specific semantics that will keep the unambiguous CAD representation in compliance with manufacturing rules and design specifications.

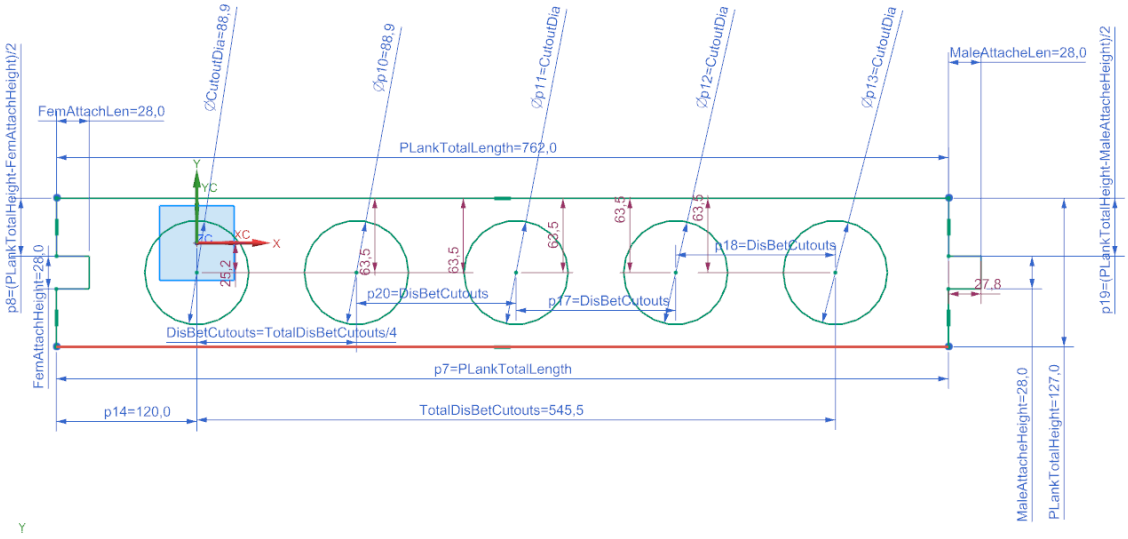


Figure 119: It's critical that sketches are fully constrained to maintain the integrity of the model when applying parametric modifications within SysML

A proper balance among constraints is necessary in order to get a fully-defined object. If the constraints of a solid model are not enough to define an object, we call it under defined, and if the object has more constraints that it needs, the object is over defined. Both under and over defined objects can lead to semantic contradictions and modeling inconsistencies. The present implementation aims to control the constraints

balance by only evaluating rules coming from material-specific knowledge into well-defined, fully-constrained, solid models. This means, there will not be topological modifications started outside a fully-defined CAD model.

Parametric model of the assembly

The first activity of this exercise was the development of a parametric model for the assembly. For each element of the system, all geometric features that must be described for a complete model will be created in the CAD application and then imported into the SysML environment for analysis.

Restrictions and assumptions for the exercise:

- Components will be modeled as nominal, which means clearances will be zero and lengths will be at their ideal value, for all material systems.
- Only manufacturing, tolerances, and clearances will be assessed. Although other parameters or behaviors such as gravity can be incorporated in this approach, they will not be part of the scope of this exercise.
- Off-the-shelf standard dimensions for all components will not be considered. Rather, all components will be understood as custom made for the specific assembly. This condition will ensure total geometric freedom for features updates for each material system to their ideal.
- Components will be independently imported to the SysML platform. Mating conditions, fastening features, and their clearances will be calculated independent of geometry, based on design and manufacturing specifications stored in the SysML environment.

- Based on the current practice of light-gauge steel framing, which allows a case-by-case field fastening, this exercise will not consider hole locations.

Parts to be tested and their material domain:

- BottomTrack: Sheet metal SDCT
- TopTrack: Sheet metal SDCT
- Headers : Sheet metal SDCT
- Planks: Pre-cast concrete SDCT
- ShortStudBottom: Sheet metal SDCT
- ShortStudTop: Sheet metal SDCT
- ConcreteSlab: Cast in place SDCT
- Stud: Sheet metal SDCT
- StudJamb: Sheet metal SDCT
- StudShortHeader: Sheet metal SDCT
- Window: PVC windows SDCT

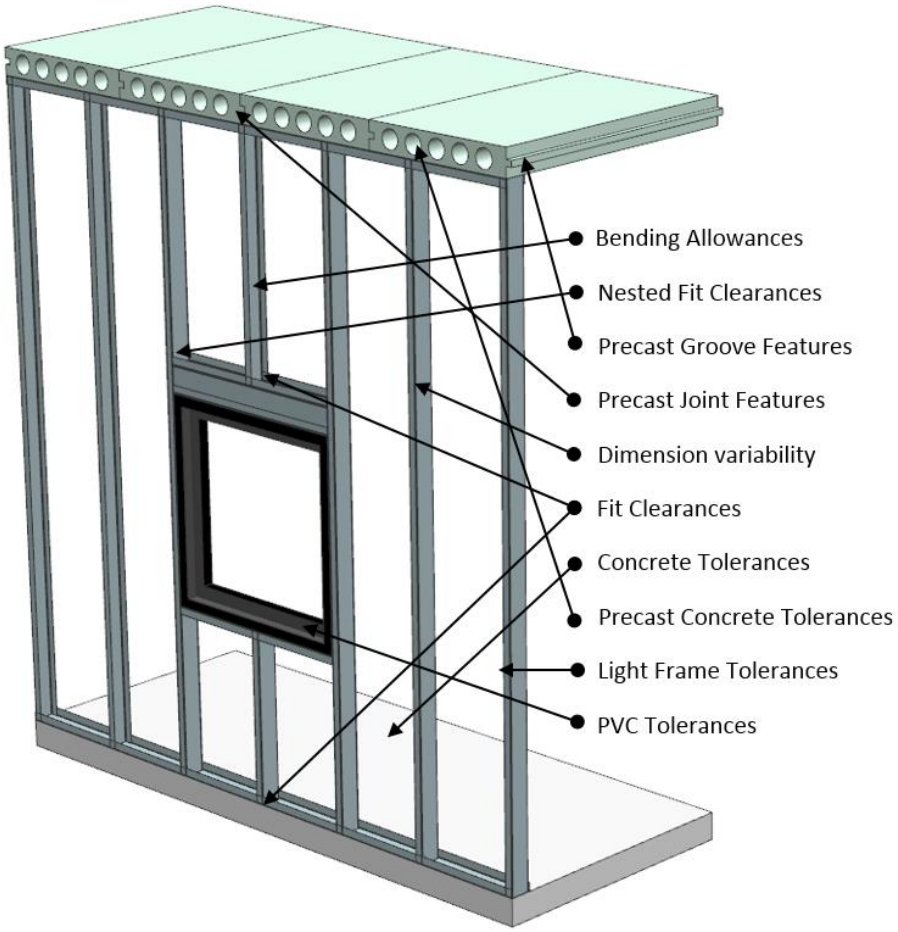


Figure 120: Wall Assembly with context-specific issues about material systems: tolerances: and clearances to be identified during the case study

7.2.1. Material Systems

Light-gauge framing:

The recommended tolerances from several standard and know-how sources state that plumbness and the level of studs must be within 1/960 of the span, or 1/8 inch (3mm) per 10 feet. ASTM C840 [115] requires that the attachment surface of any member shall not vary more than 1/8 inch from the plane of the faces or adjacent framing members. The Gypsum Association also states that adjacent fastening surfaces of framing or furring should not vary more than 1/8 inch. Previous specification guides from the Metal Lath/Steel Framing Association (ML/SFA) also recommended the same tolerances as

ASTM C1007. The 1/8 inch (3mm) per 10 feet tolerance is consistent with the substrate requirements for other finish materials, such as some types of ceramic. ASTM C754 requires that spacing of studs and other framing members vary no more than 1/8 inch from the required spacing and that the cumulative error does not exceed 1/8 inch (3mm). This is to ensure that the edge of a piece of gypsum board has sufficient bearing on half of a stud for fastening.

Concrete slab tolerances:

SDCT for concrete are applied to physical dimensions such as thickness, length, width, squareness, and location and size of openings. They are determined by economical and practical production considerations, and functional and appearance requirements. For this classification we identify two main kinds of concrete surfaces. First, formed surface is a surface requiring formwork to provide shape and texture/finish to the concrete. Second, unformed surface is a surface that does not require formwork to provide either shape or finish to the surface, for example, the top surface of slabs or pavements. These surfaces generally have to meet two independent tolerances the "flatness" of the surface and variation from the designed elevation called "levelness". Flatness is the deviation of the surface from a straight line joining two points on the surface. Levelness (height tolerance) is the permitted vertical variation of the surface from a fixed external reference point or datum. Level alignment tolerances of the top surface of the slab are important because it is in this surface where the bottom track profile that will support the metal structure will be assembled. From Standards, over the entire surface of the slab, all points must fall within an envelope of ¾ inch (19mm) above or below the ideal (nominal) plane. Flatness is also relevant to ensure the correct fix of the bottom track of the metal framing.

From Standards, fatness of a slab that will require a proper level of flatness to fix a track profile without creating gaps must fall within ¼ inch (6mm) and ½ inch (13mm).

Hollow-core concrete plank:

Allowable dimensional tolerance for the hollow-core concrete blocks based on ASTM standards is a general 1/8 inch (3mm) from the actual dimension . This includes width, height, and length. However, in practice the units are manufactured to a 1/16 inch (1.6 mm) tolerance. For non-load-bearing concrete blocks, the face shell thickness cannot be less than ½ inch (13mm). For concrete building bricks, the face shell thickness tolerance is 1/8 inch (3mm). In addition, the total variation in finished face dimensions of prefaced unit cannot exceed 1/16 inch between the largest and the smallest unit in any lot of each size. The distortion of the plane and edges of the face or prefaced unit from the corresponding plane and edges of the concrete unit cannot exceed 1/16 inch (1.6mm).

PVC Windows:

This section includes the standards and dimensions for PVC windows that are manufactured according to the Windows Institute. It is very often seen that windows and doors present issues with installations and operations due to tolerances problems. Some of the specifications for tolerances are the results of tests using mechanical equipment such as ventilators to check gaps between framing parts. For residential units it should not be possible to insert a feeler gauge 0.031 inch thick between the inside contacts or freely insert a 0.020 inch feeler gauge between more than 40 percent of the contacts. For the framing sections, these must be constructed so that the glass in each window will lie in the same plane within a tolerance of ¼ inch. For the frame members I applied a deflection tolerance that is not bigger than 1/175 of the span of the member. Outside

frame members must be designed to lap masonry at least ½ inch. This last condition has not been applied to the actual model.

For this experiment, the critical matter is to know how the assembly analysis determines if all the parts will fit together. In particular, we are concerned with determining if an assembly is an interchangeable assembly. An assembly is an interchangeable assembly if none of the constituent parts interfere with each other in their assembled positions for any possible set of parts that are manufactured to within specified tolerances (Figure). For instance, Figure shows that before the analyses, several components are interfering with each other, which means it is not an interchangeable assembly due to assembly tolerances and clearances that have not been well specified. From the interferences presented in Figure , we are interested in those labeled as "(Hard)" as they refer to physical clashes. The interferences labeled as "(Touching)" are not critical as they could exist based on components aggregation strategies during assembly.

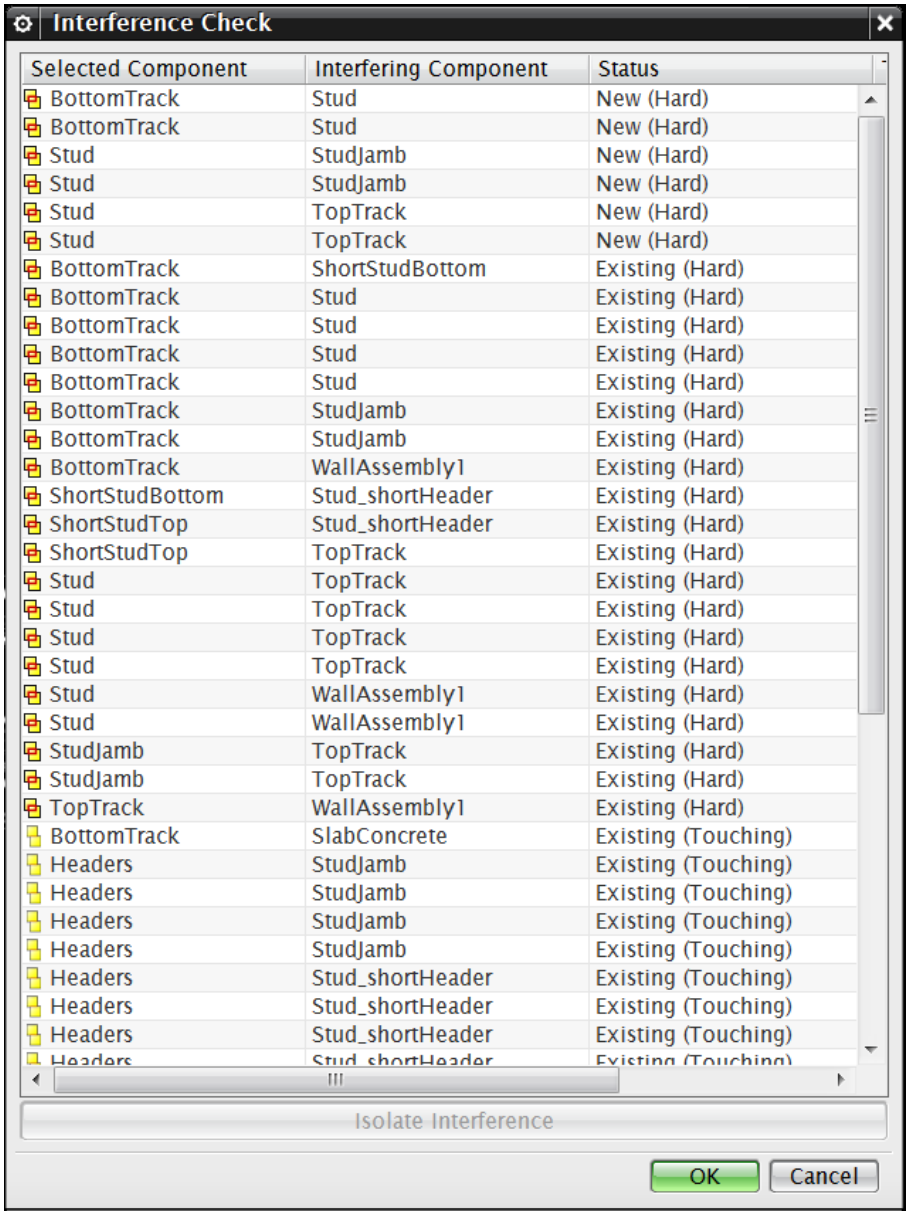


Figure 121: Interference check before analyses

7.2.2. Features Decomposition

This section presents the feature-based structural decomposition of the eleven wall components chosen for this case study. In SysML, diagrams are built from imported CAD models stored in the containment tree of MagicDraw. Figure shows a fully imported CAD model in the MagicDraw containment tree (browser). Thus, the creation of diagrams based on models imported in the browser offers another option for filtering

the features tree without interfering with the underlying topological structure of the model (Figure).

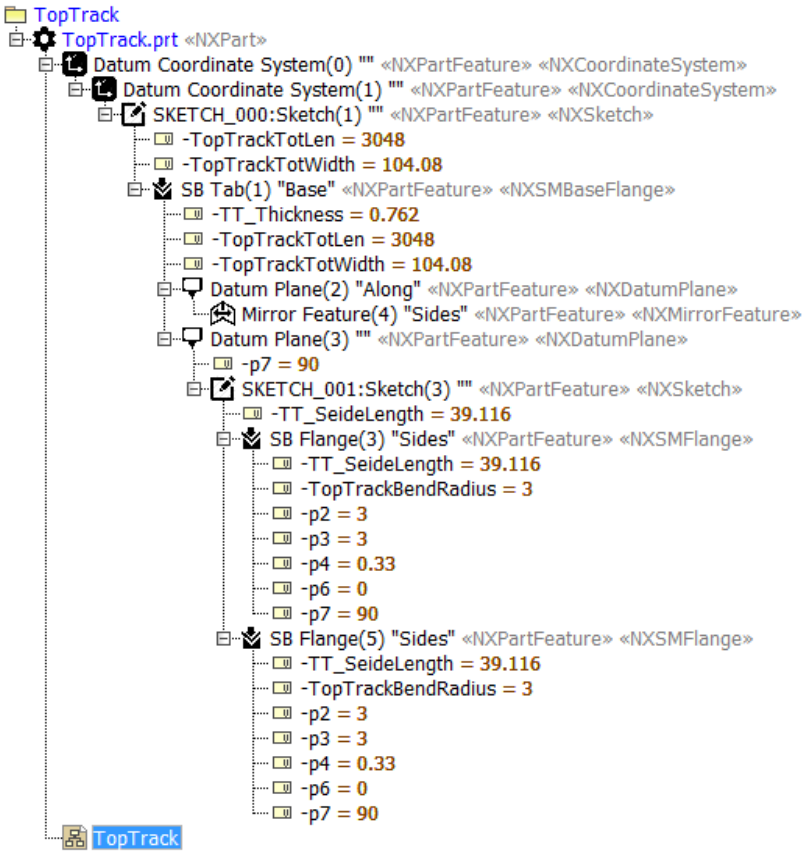


Figure 122: Full import of a sheet metal component

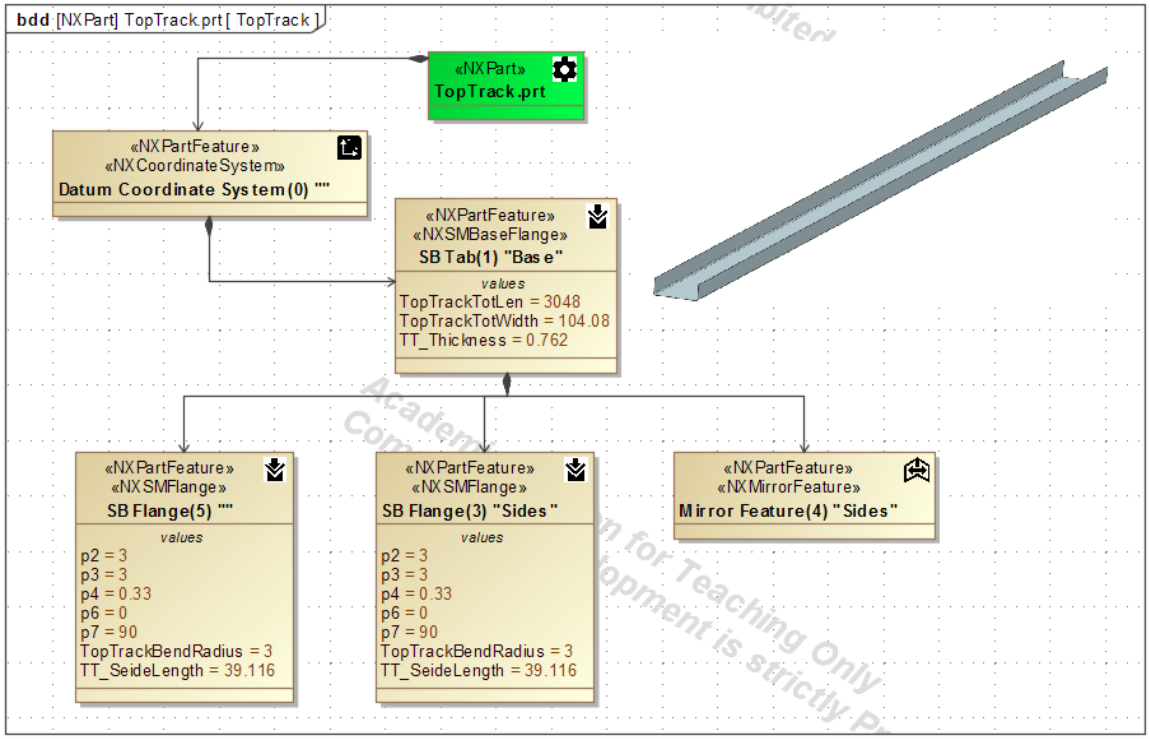


Figure 123: TopTrack: feature-based decomposition

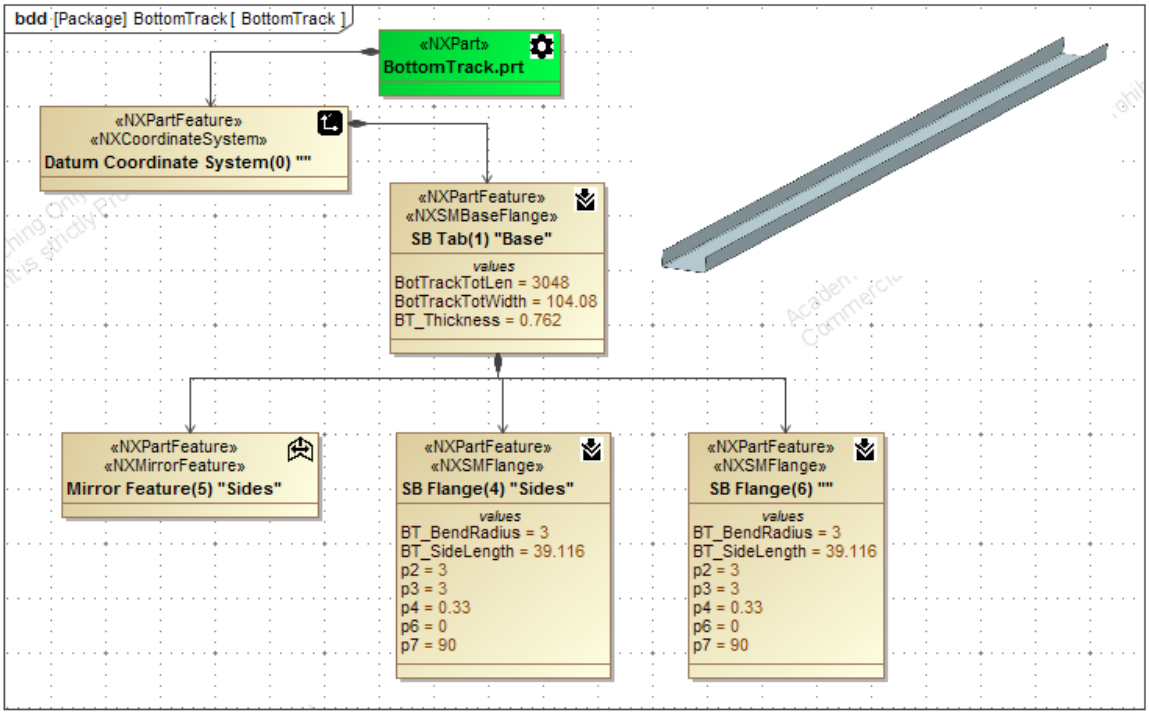


Figure 124: Bottom Track: feature-based decomposition

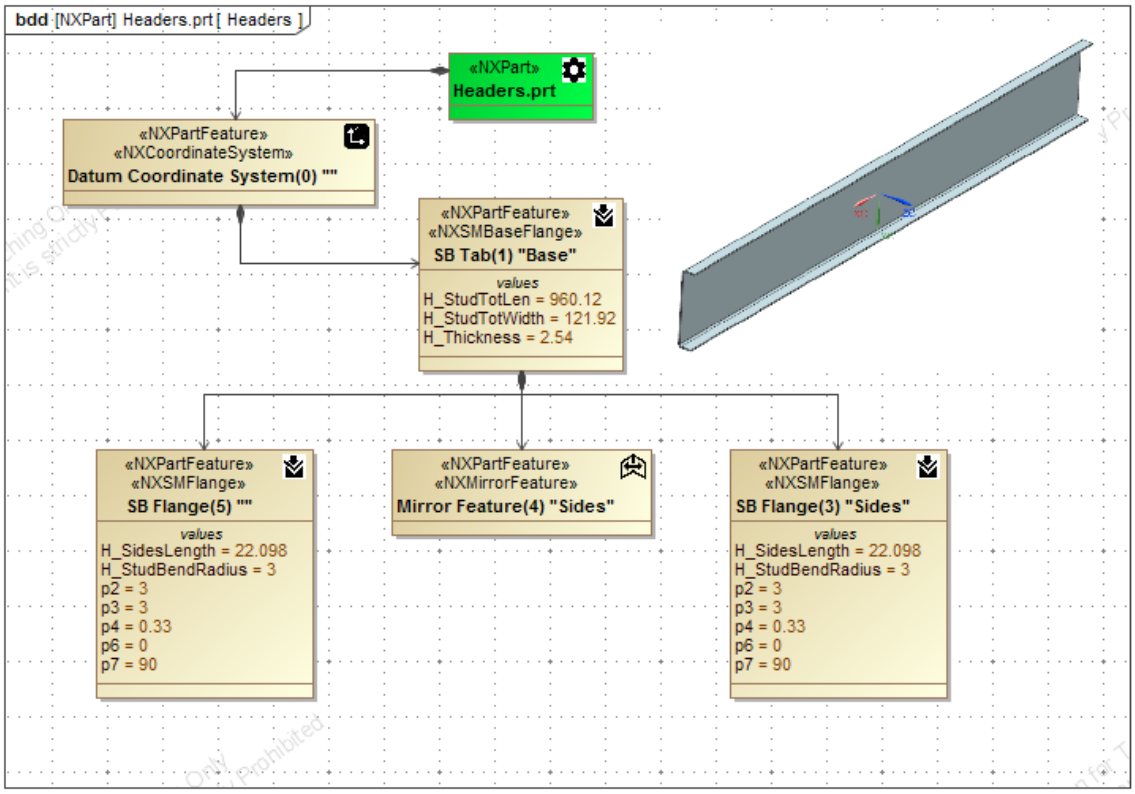


Figure 125: Headers: feature-based decomposition

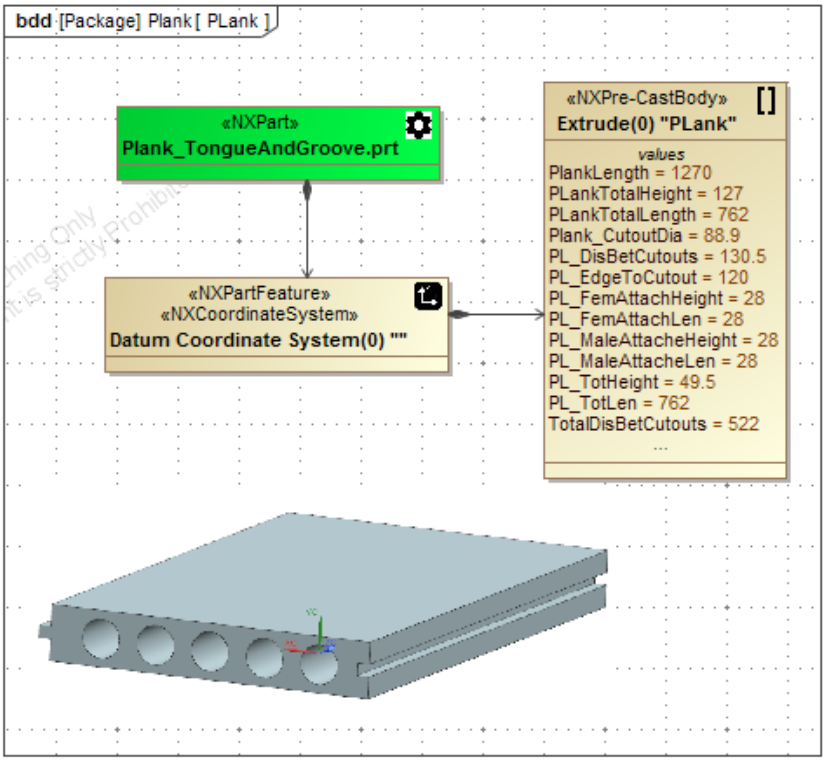


Figure 126: Pre-cast Plank: feature-based decomposition

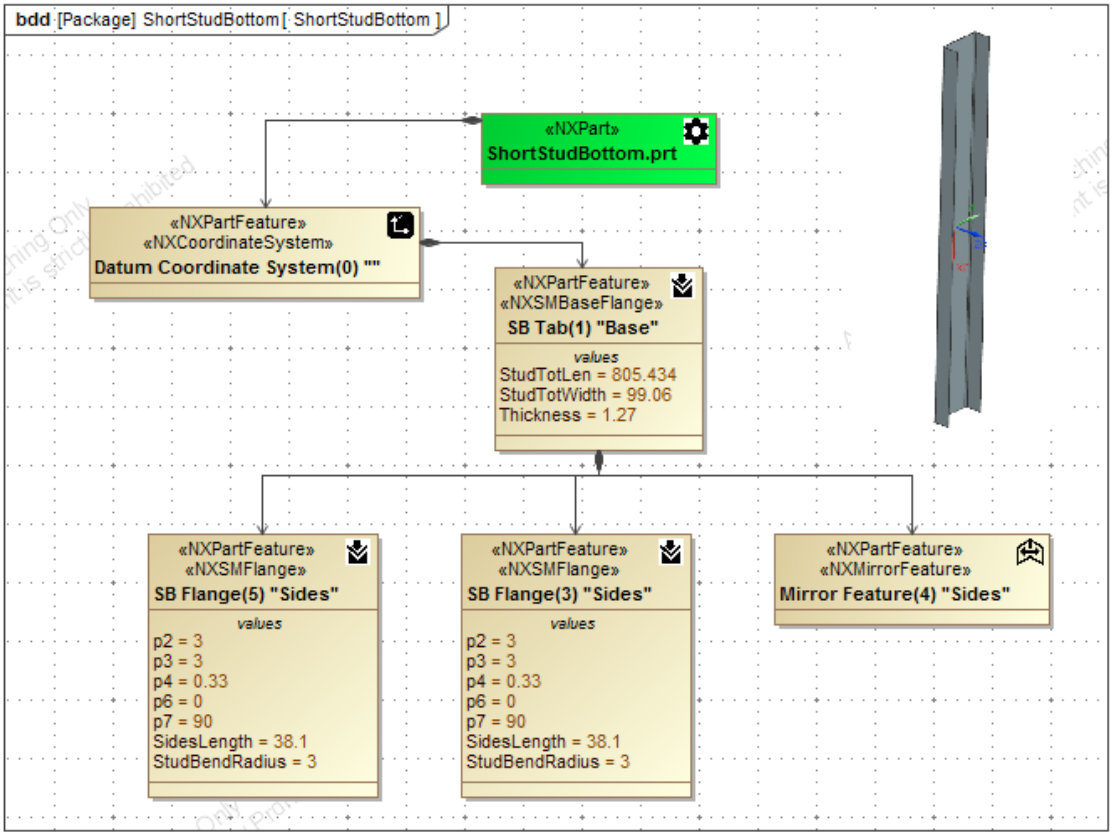


Figure 127: ShortStudBottom: feature-based decomposition

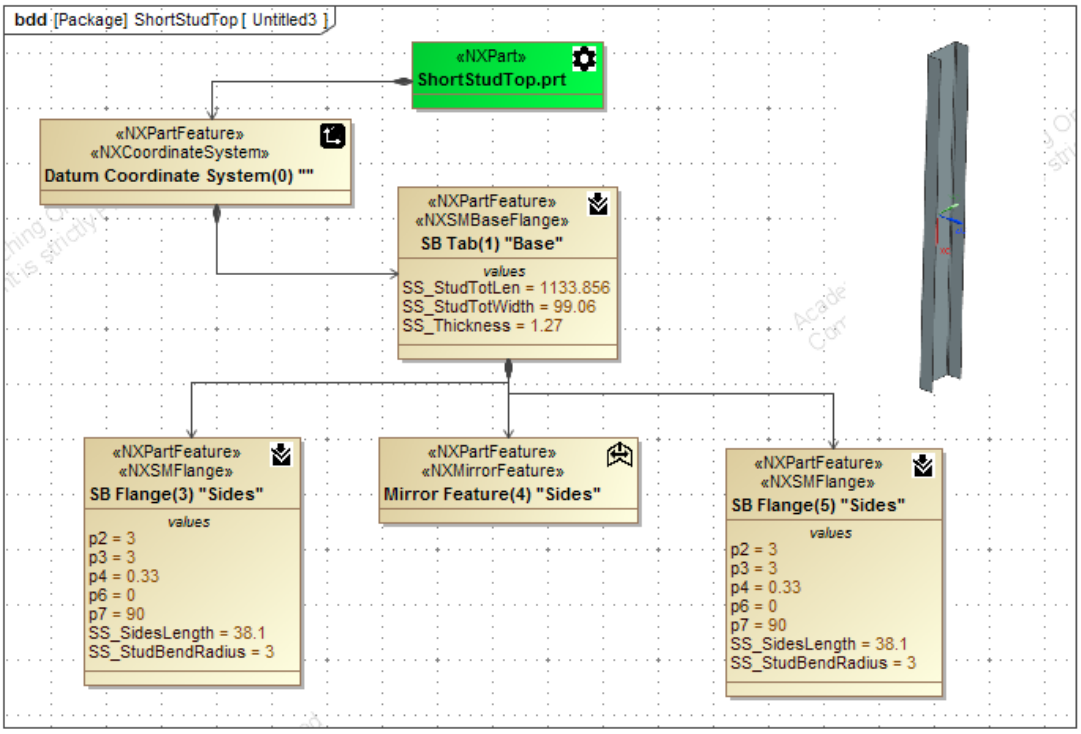


Figure 128: ShortStudTop: feature-based decomposition

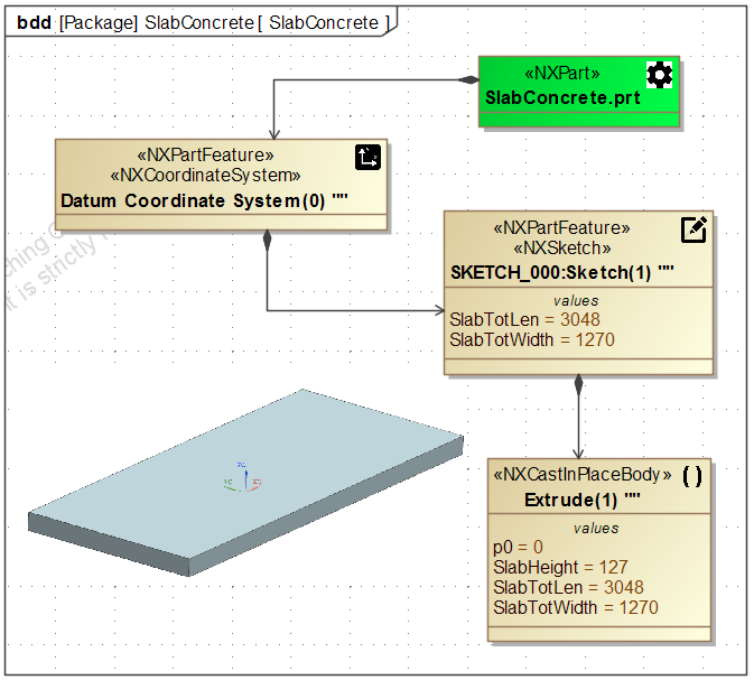


Figure 129: SlabConcrete: feature-based decomposition

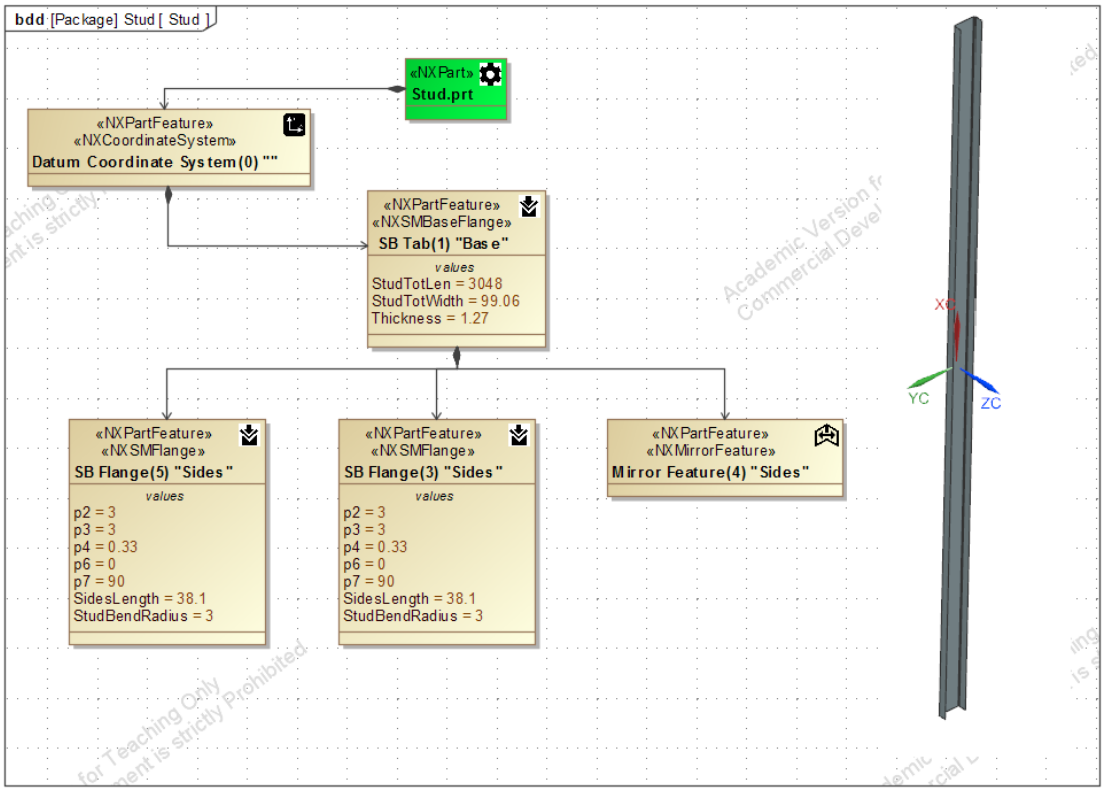


Figure 130: Stud: feature-based decomposition

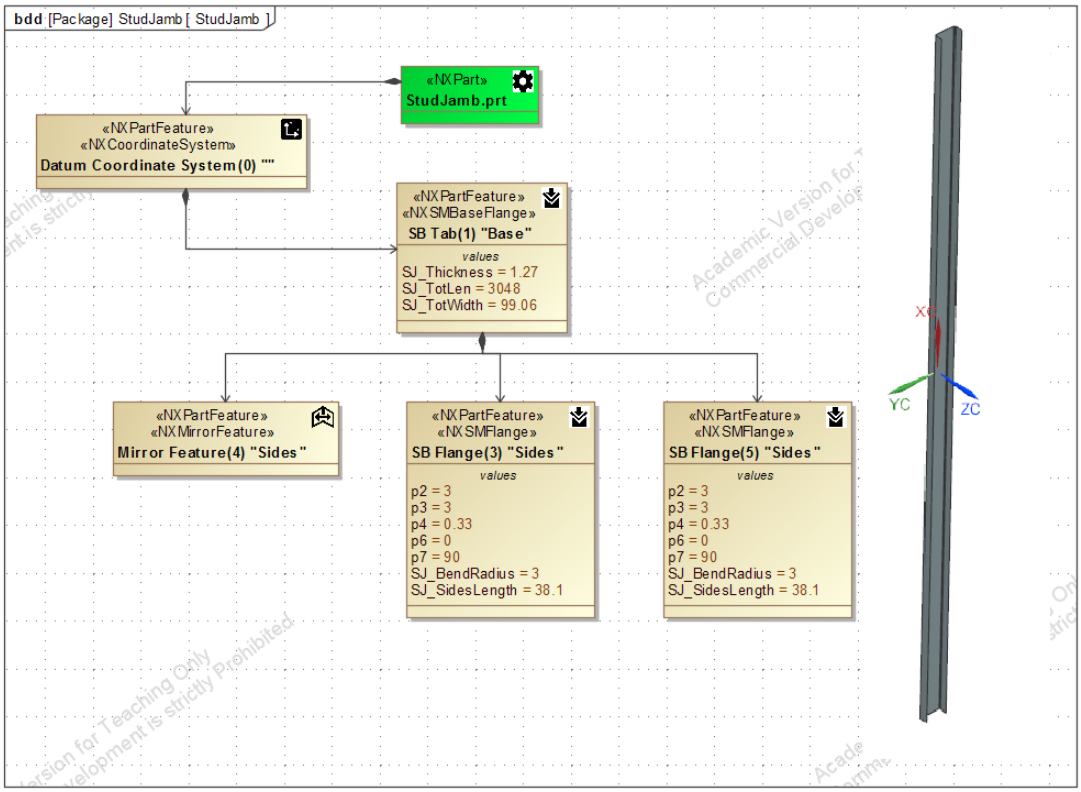


Figure 131: StudJamb: feature-based decomposition

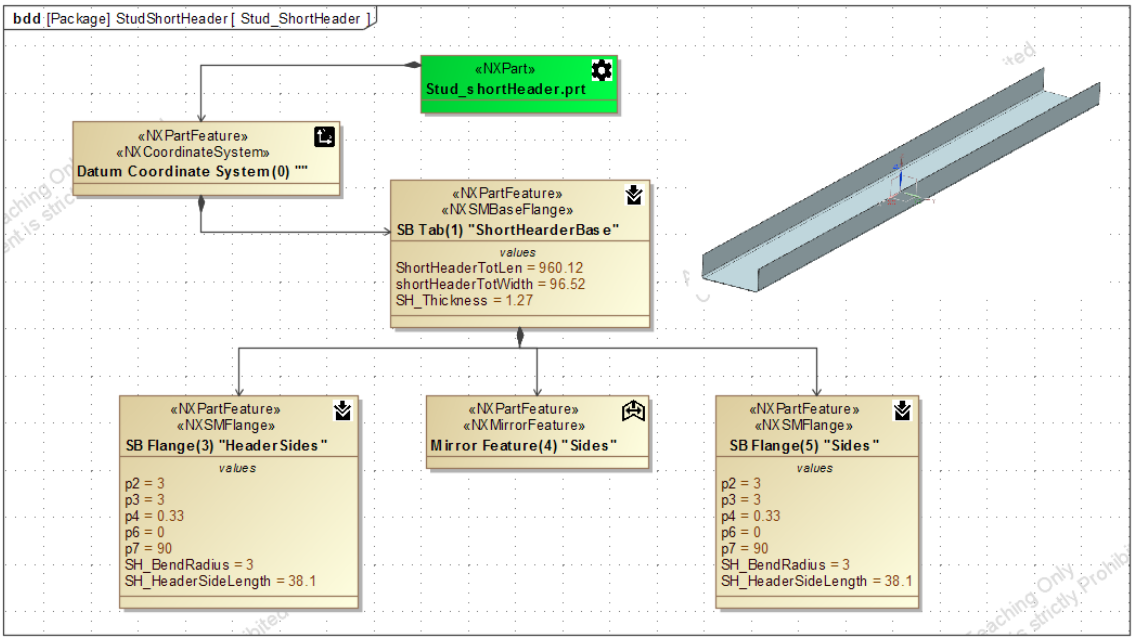


Figure 132: StudShortHeader: feature-based decomposition

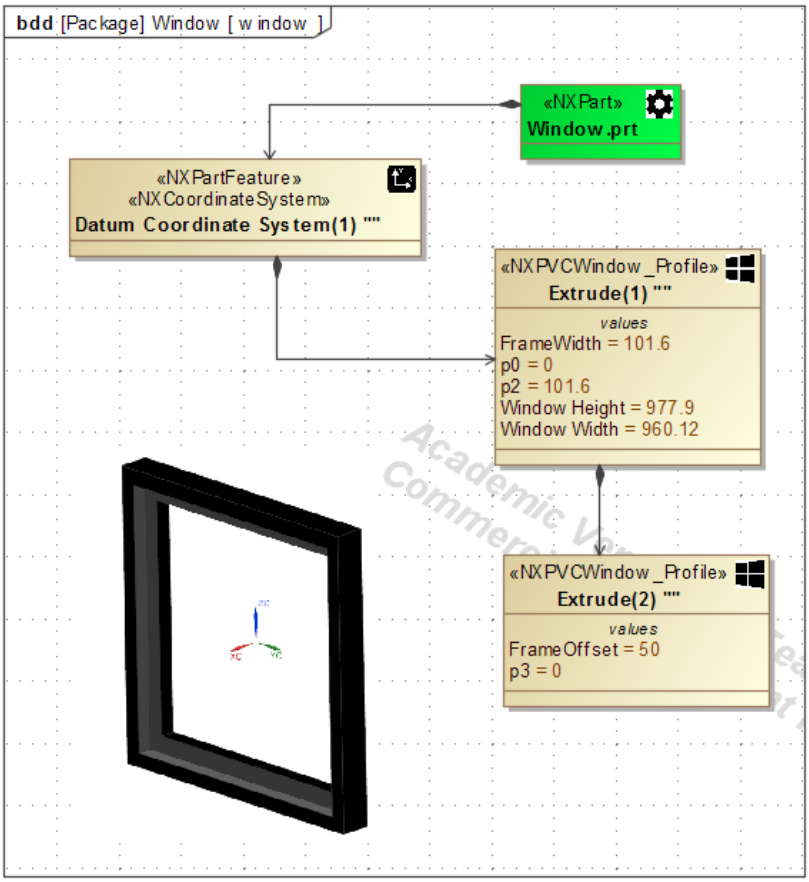


Figure 133: Window: simplified feature-based decomposition

7.2.3. Knowledge Allocation of the Four Components of the Studied Assembly

The following pictures show the results of the knowledge allocation stage. In this section a diagram with a Knowledge Allocation Matrix and a diagram with the representation in context of the feature-linked manufacturing specifications and design specifications will be given for each of the eleven components of the wall assembly.

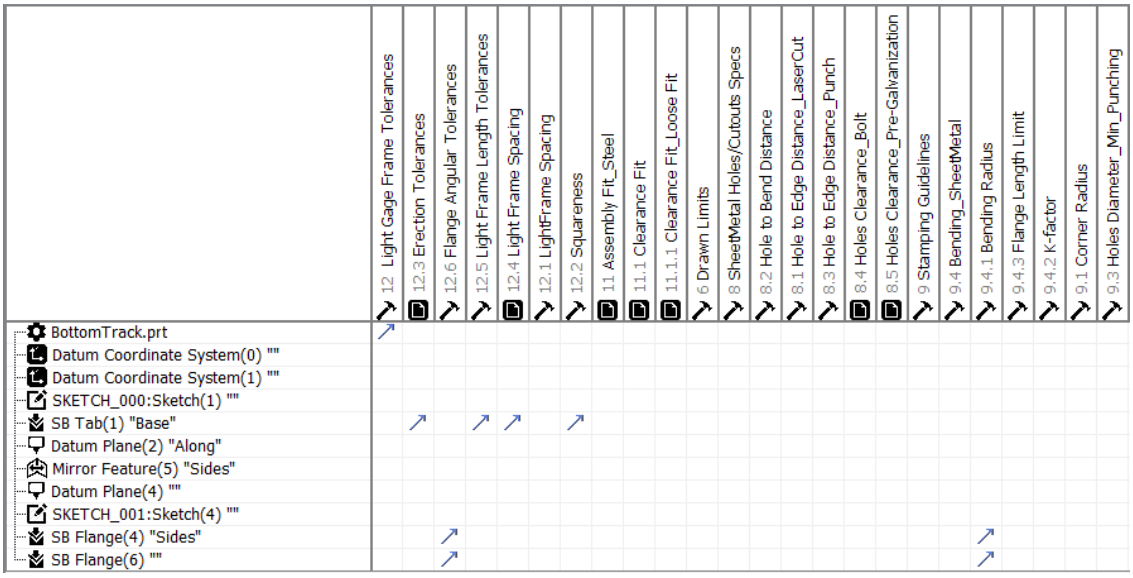


Figure 134: BottomTrack: Knowledge Allocation Matrix

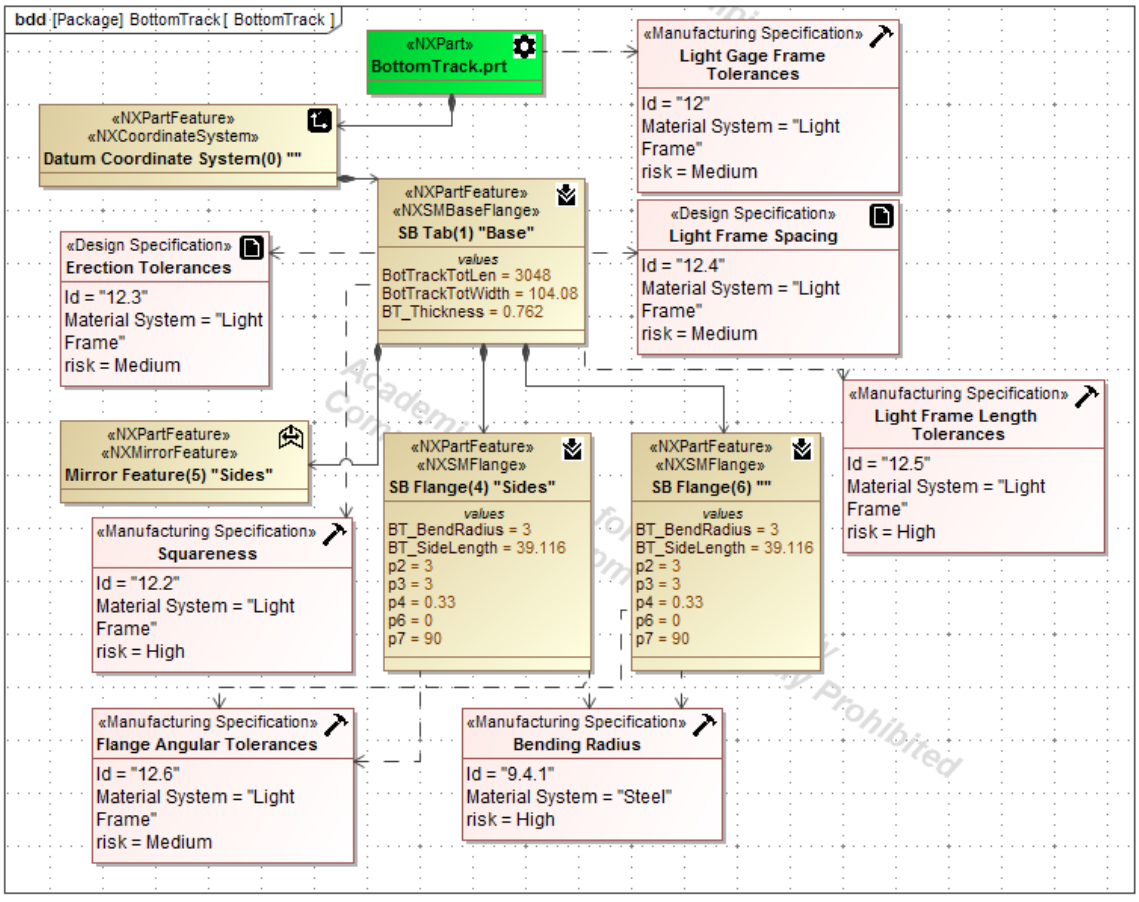


Figure 135: BottomTrack: with allocated Manufacturing and Design Specifications

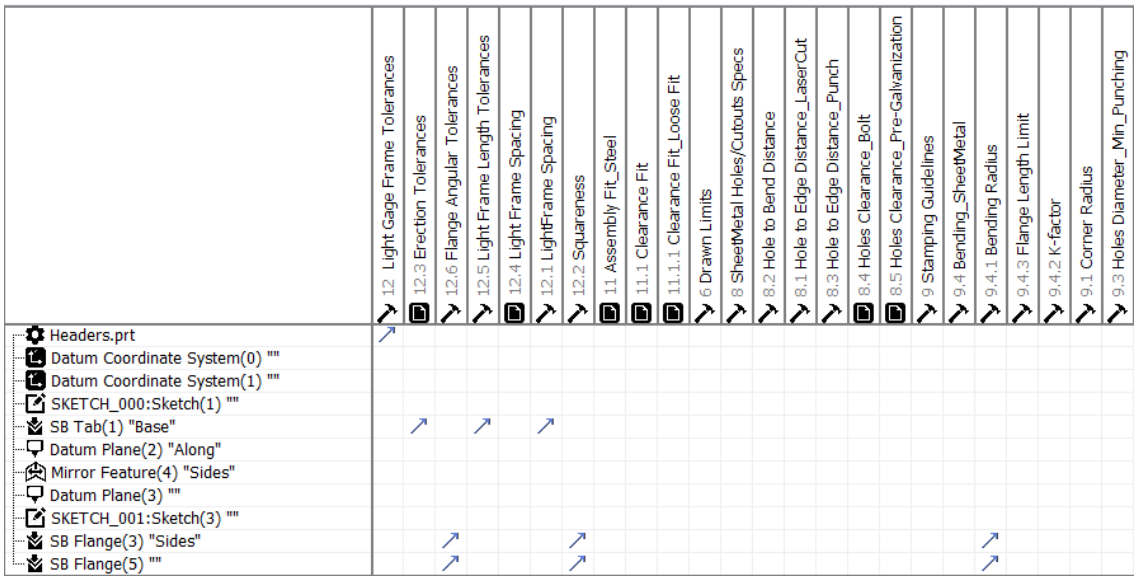


Figure 136: Headers: Knowledge Allocation Matrix

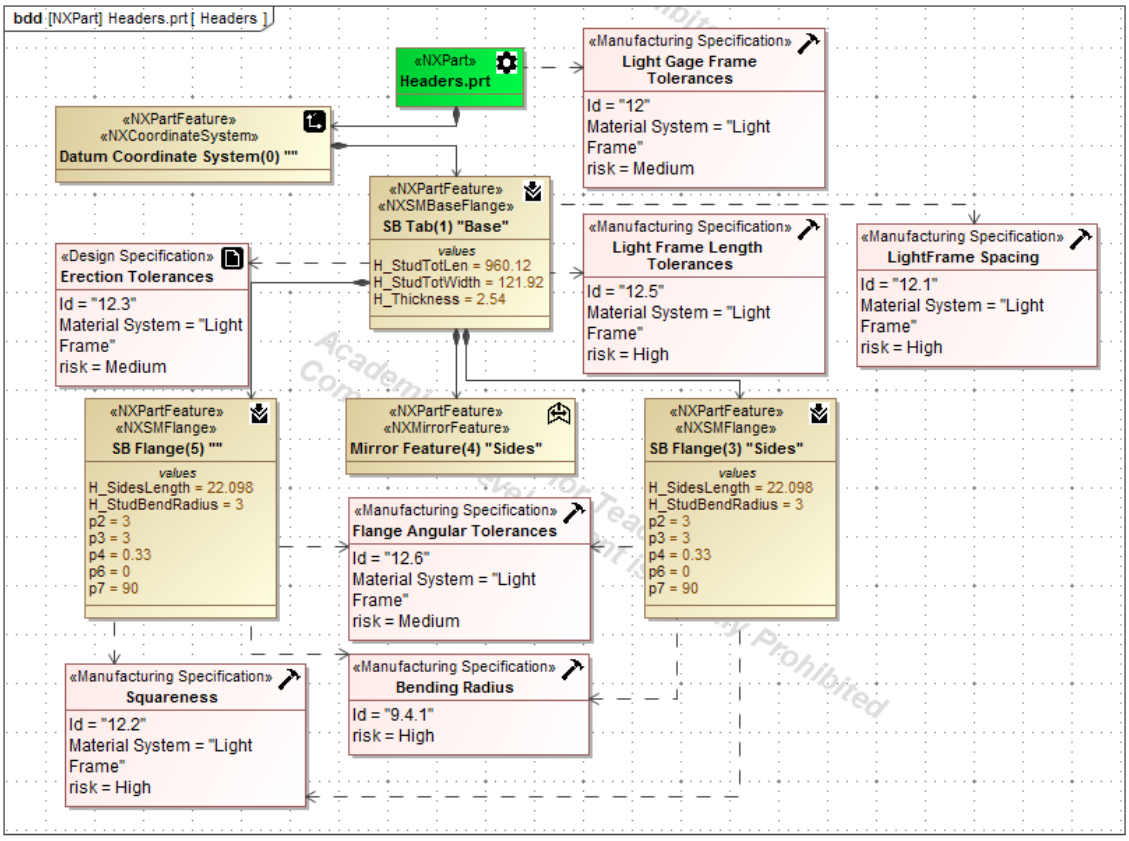


Figure 137: Headers with allocated Manufacturing and Design Specifications

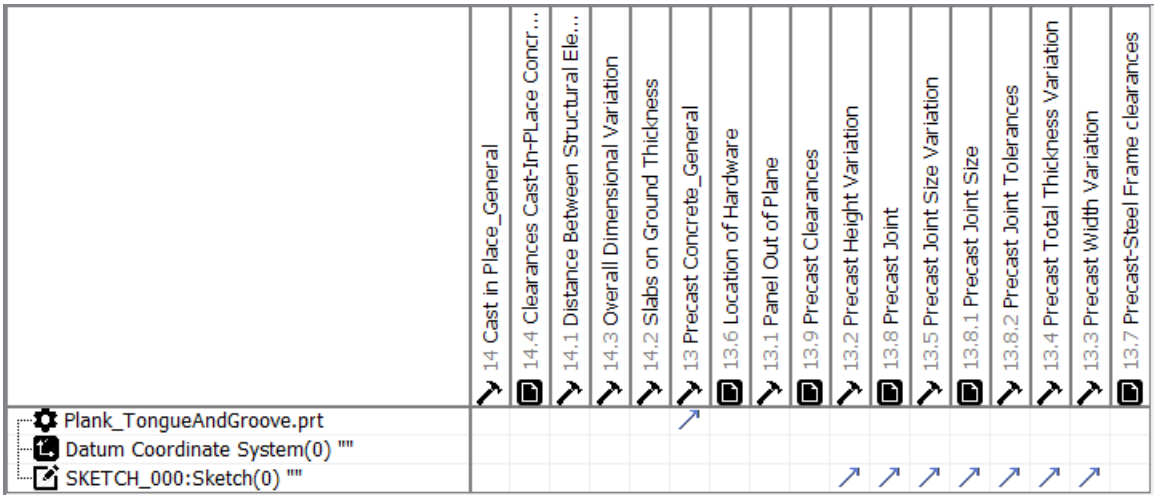


Figure 138: Precast Plank: Knowledge Allocation Matrix

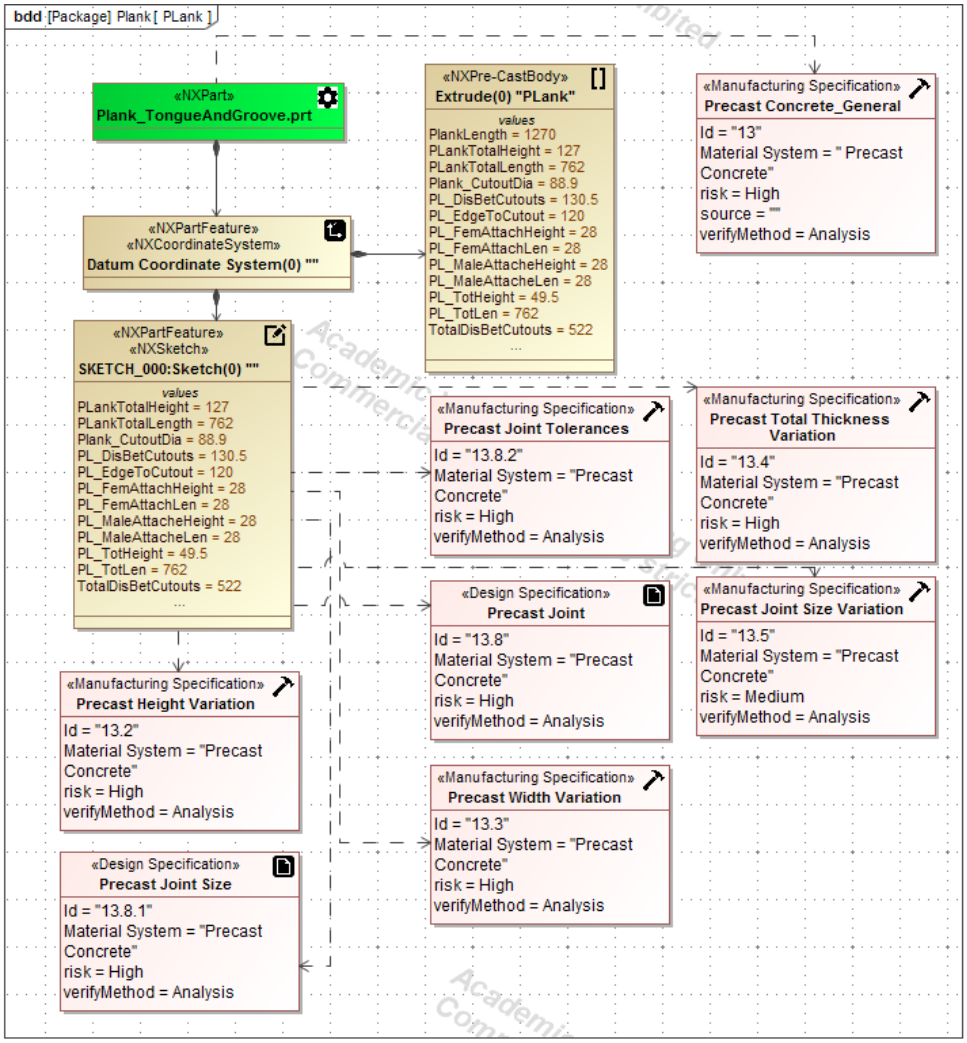


Figure 139: Precast Plank with allocated Manufacturing and Design Specifications

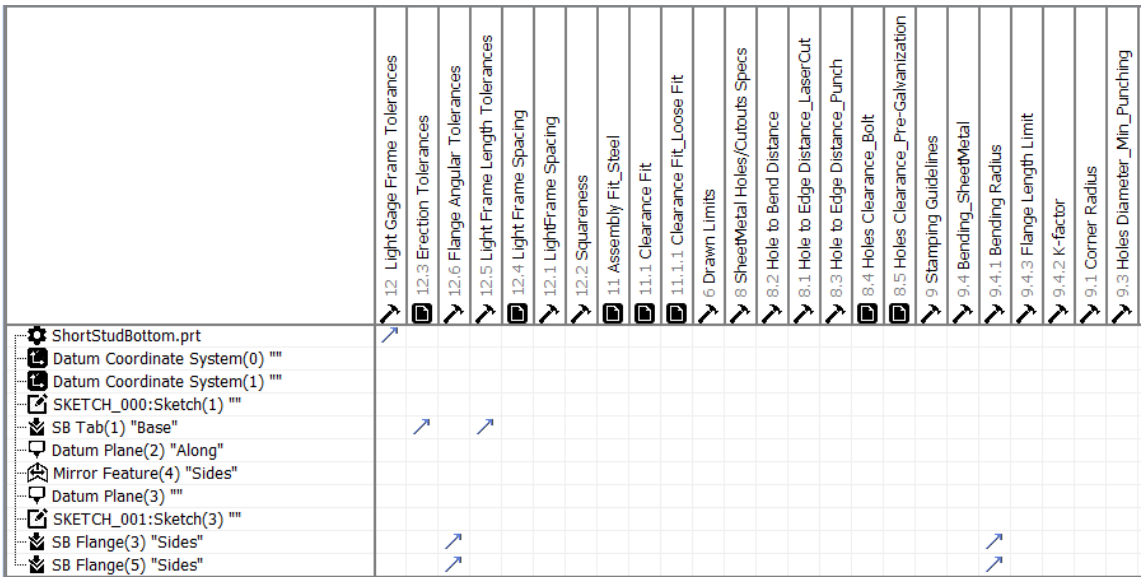


Figure 140: ShortStudBottom: Knowledge Allocation Matrix

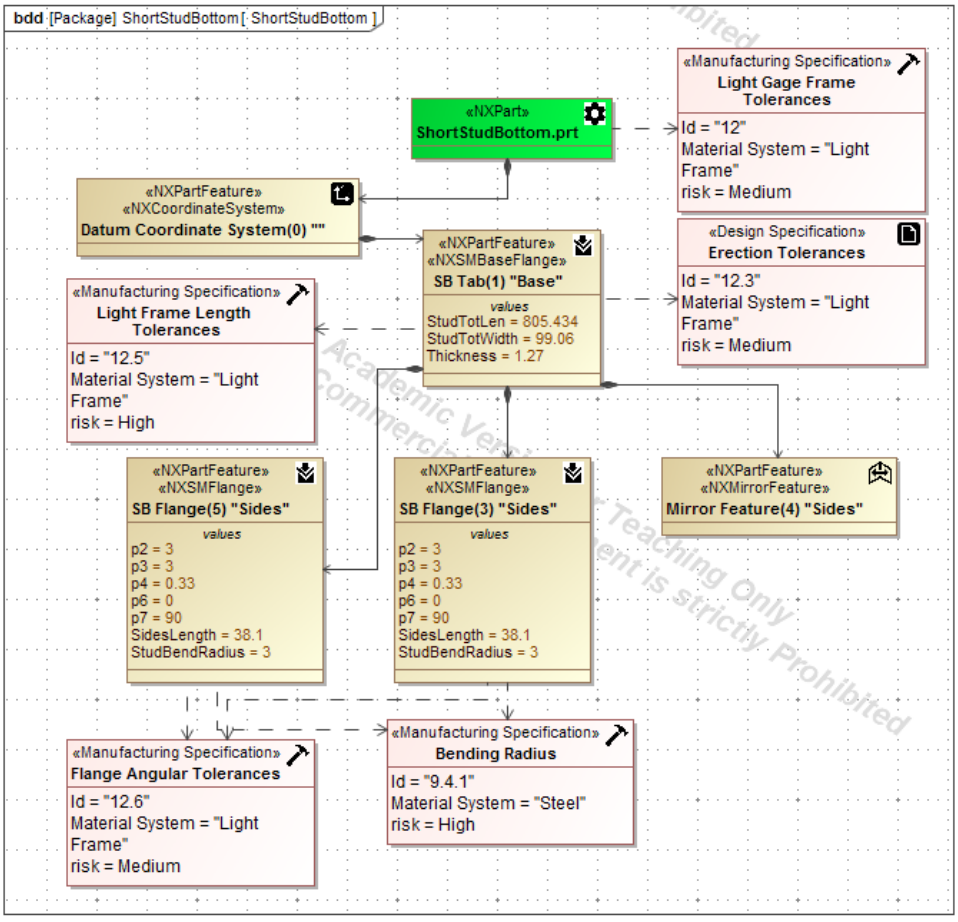


Figure 141: ShortStudBottom with allocated Manufacturing and Design Specifications

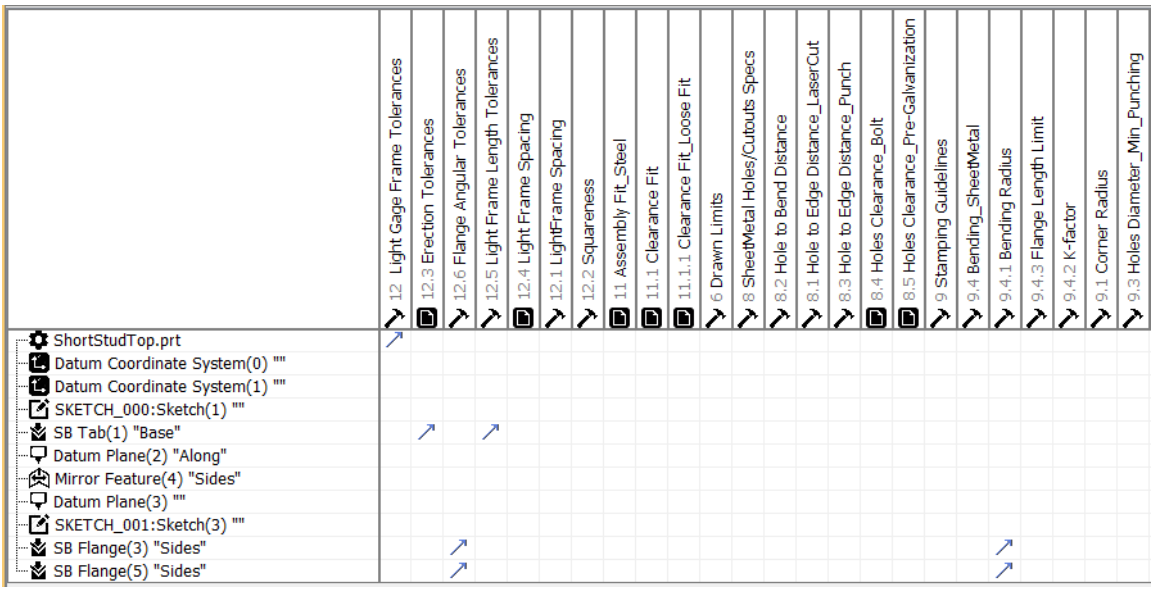


Figure 142: ShortStudTop: Knowledge Allocation Matrix

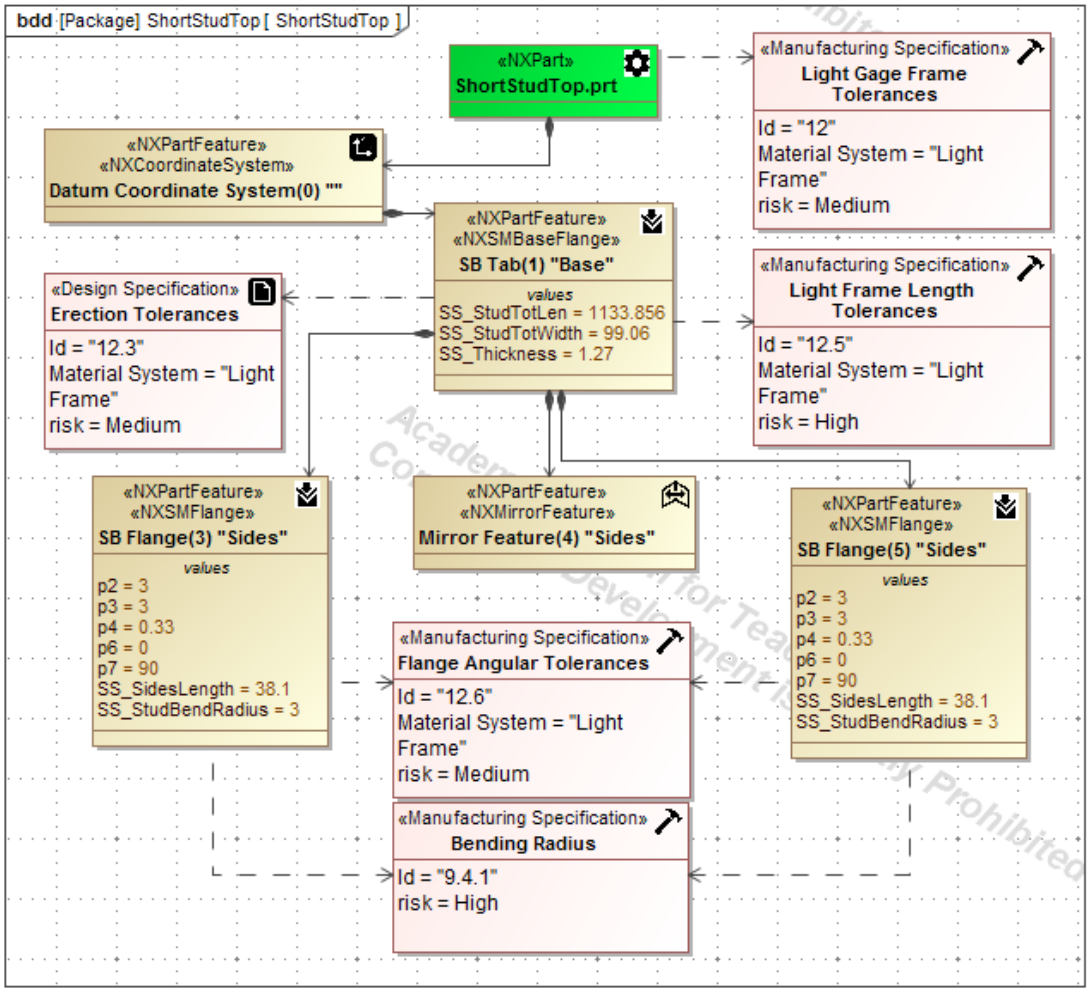


Figure 143.: ShortStud with allocated Manufacturing and Design Specifications

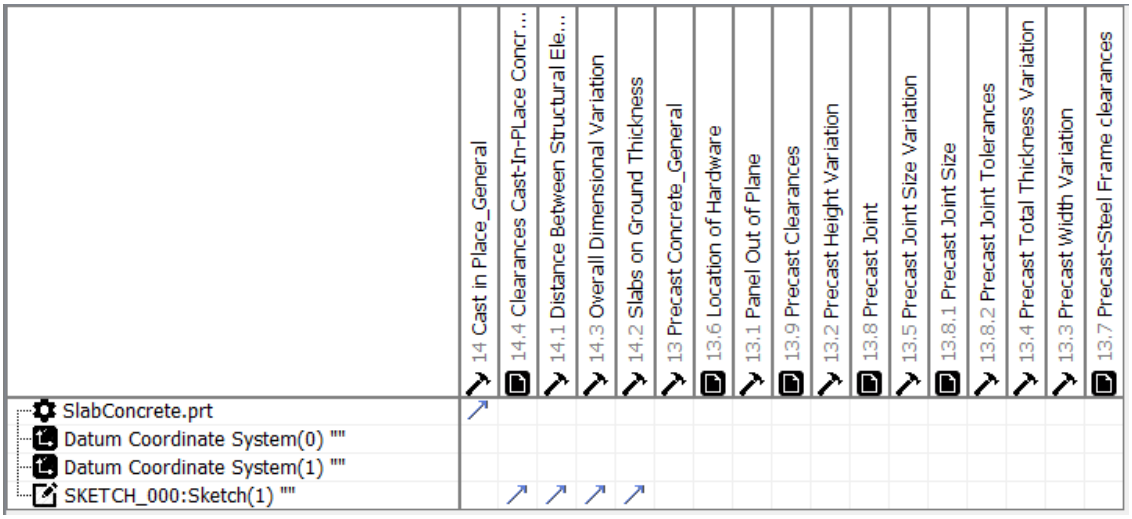


Figure 144: SlabConcrete: Knowledge Allocation Matrix

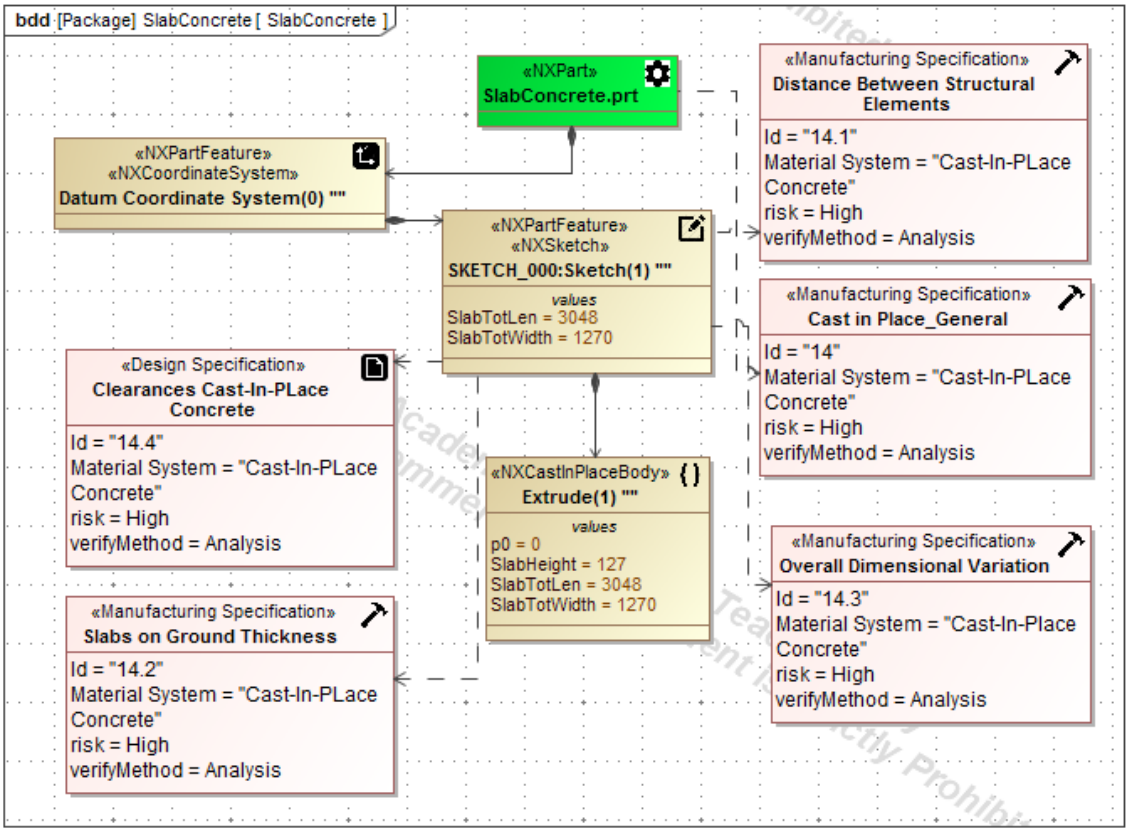


Figure 145: SlabConcrete with allocated Manufacturing and Design Specifications

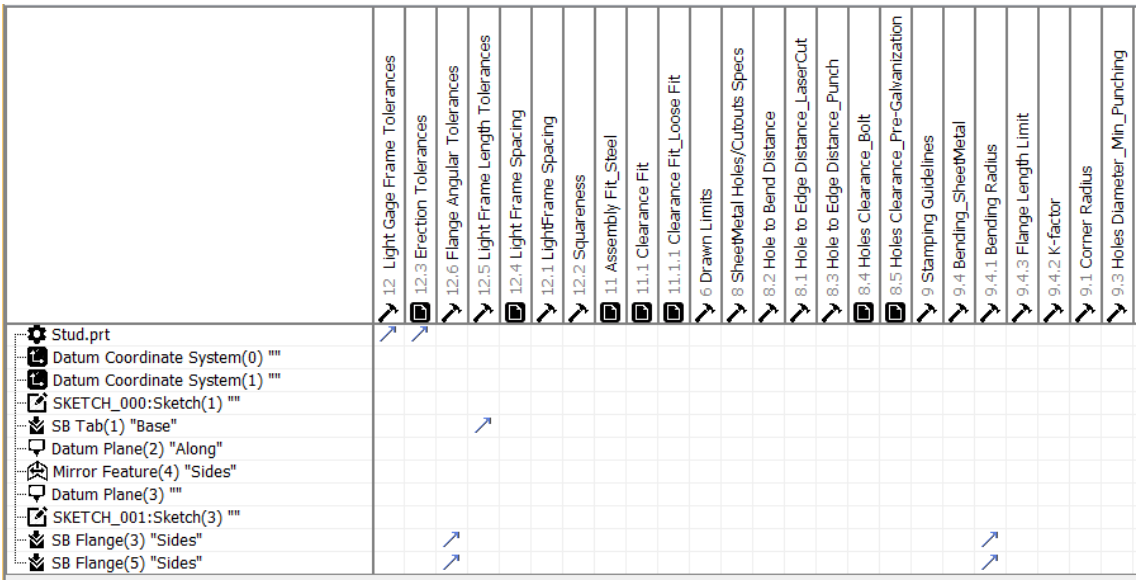


Figure 146: Stud: Knowledge Allocation Matrix

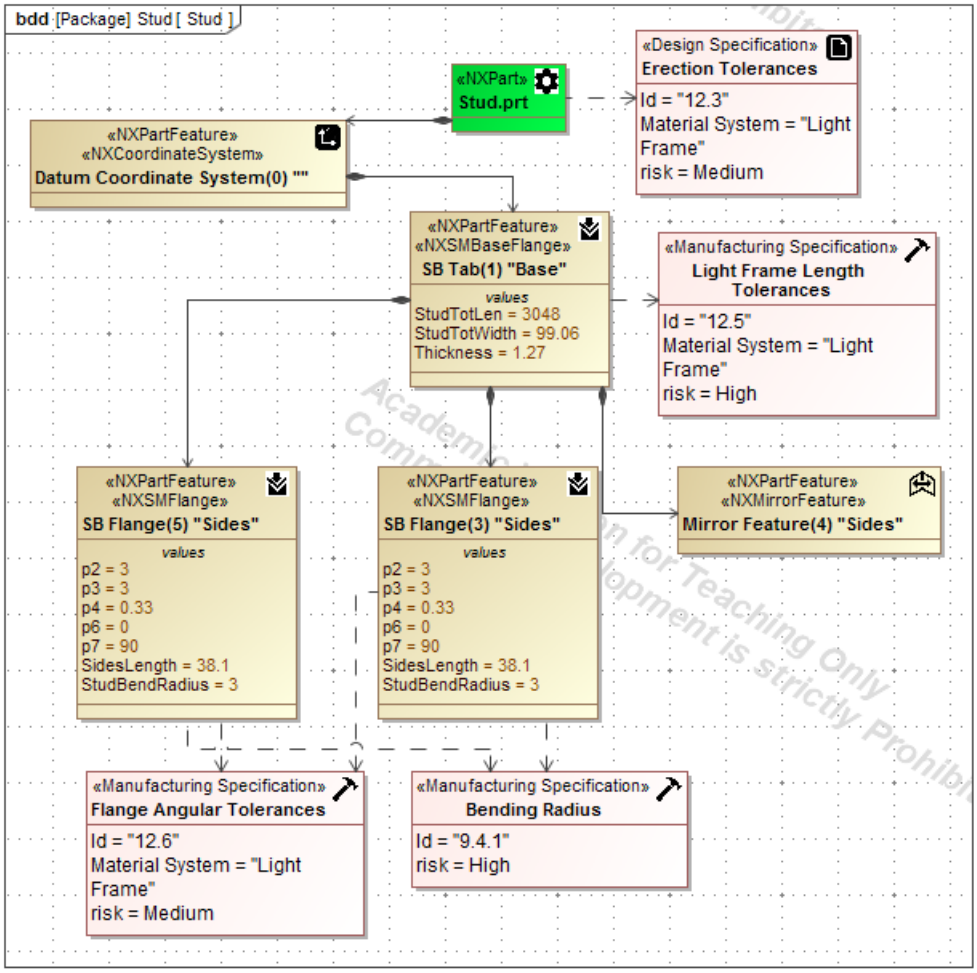


Figure 147: Stud with allocated Manufacturing and Design Specifications

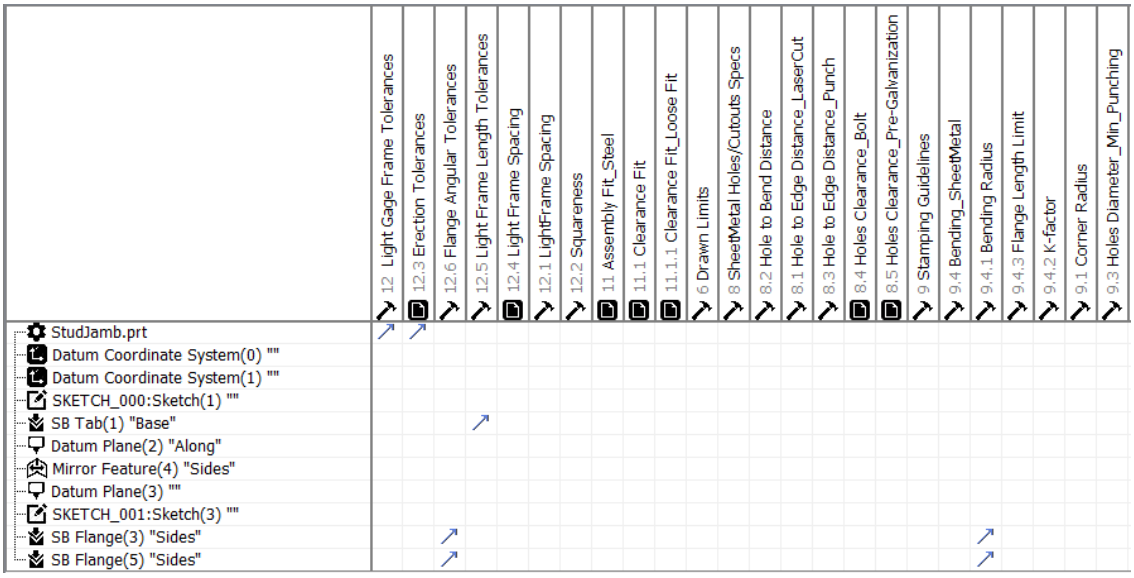


Figure 148: StudJamb: Knowledge Allocation Matrix

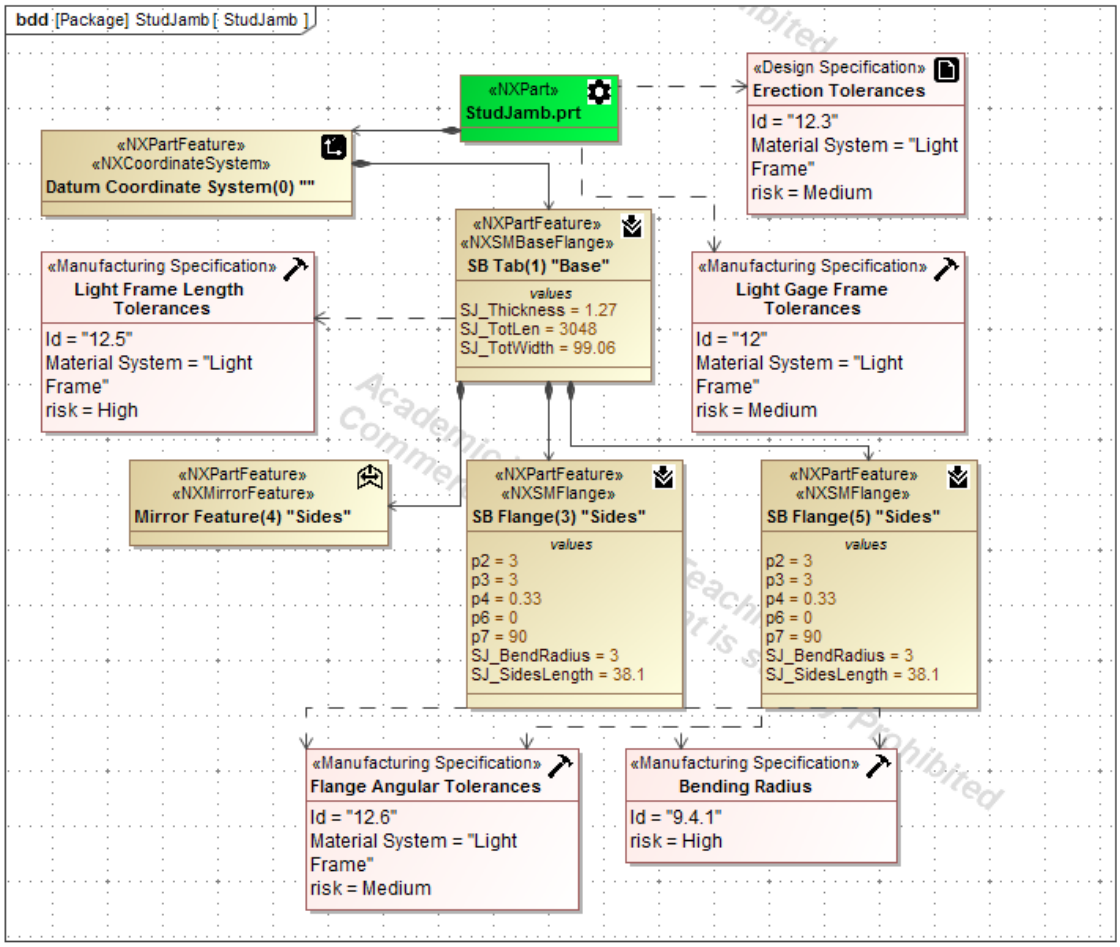


Figure 149: StudJamb with allocated Manufacturing and Design Specifications

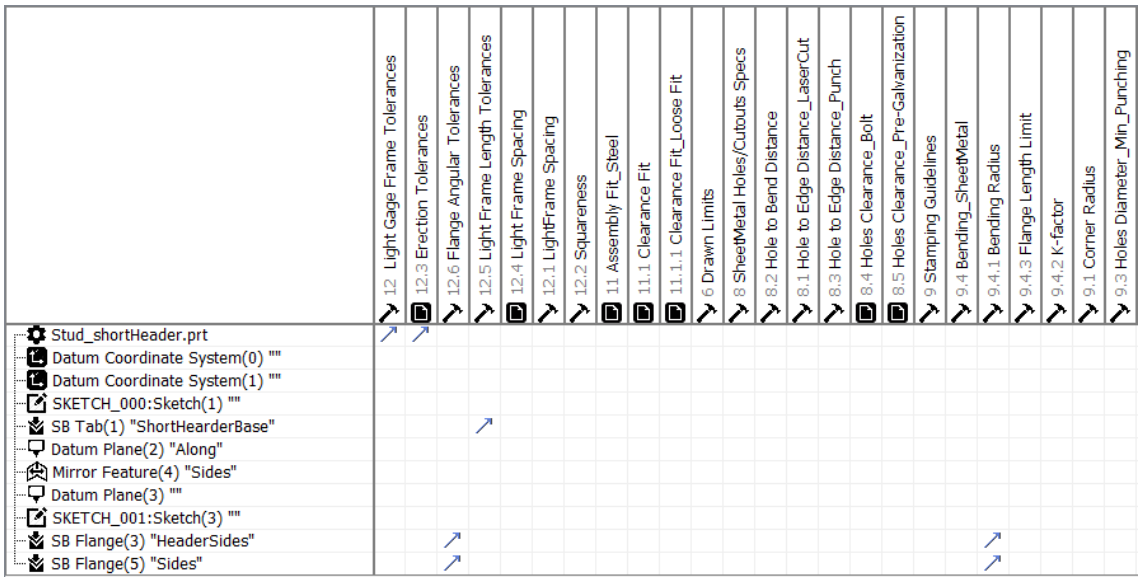


Figure 150: Stud_ShortHeader: Knowledge Allocation Matrix

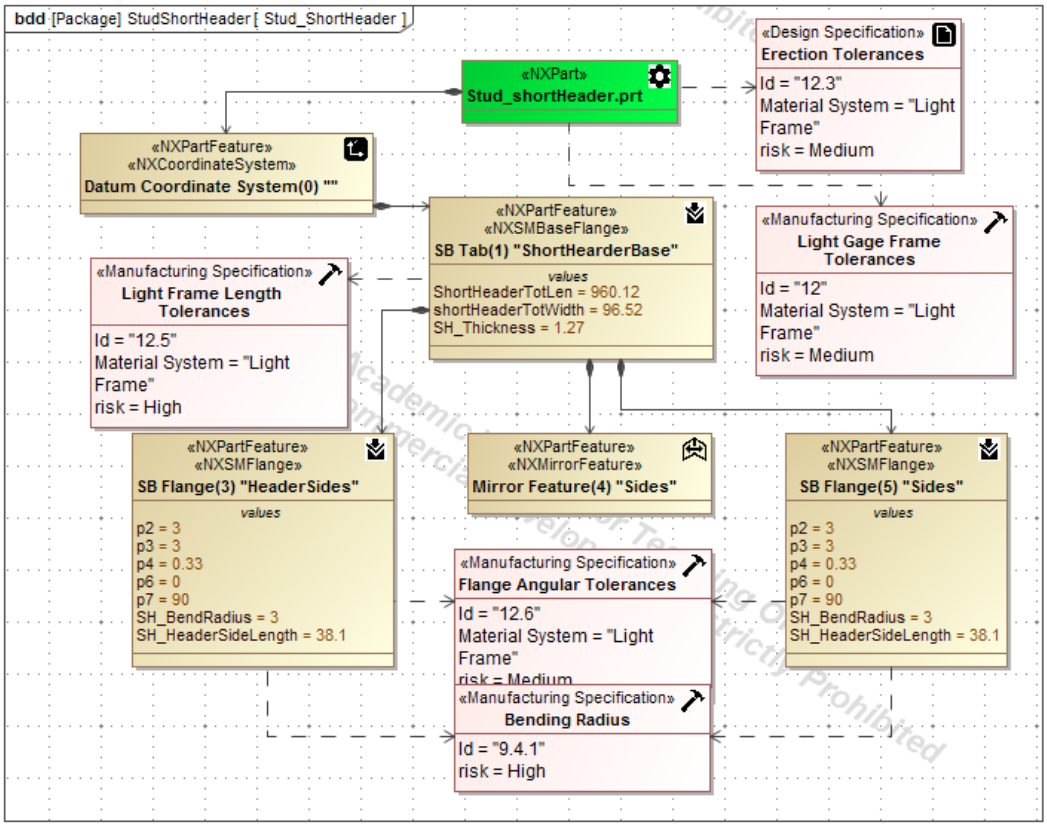


Figure 151: Stud_ShortHeader with allocated Manufacturing and Design Specifications

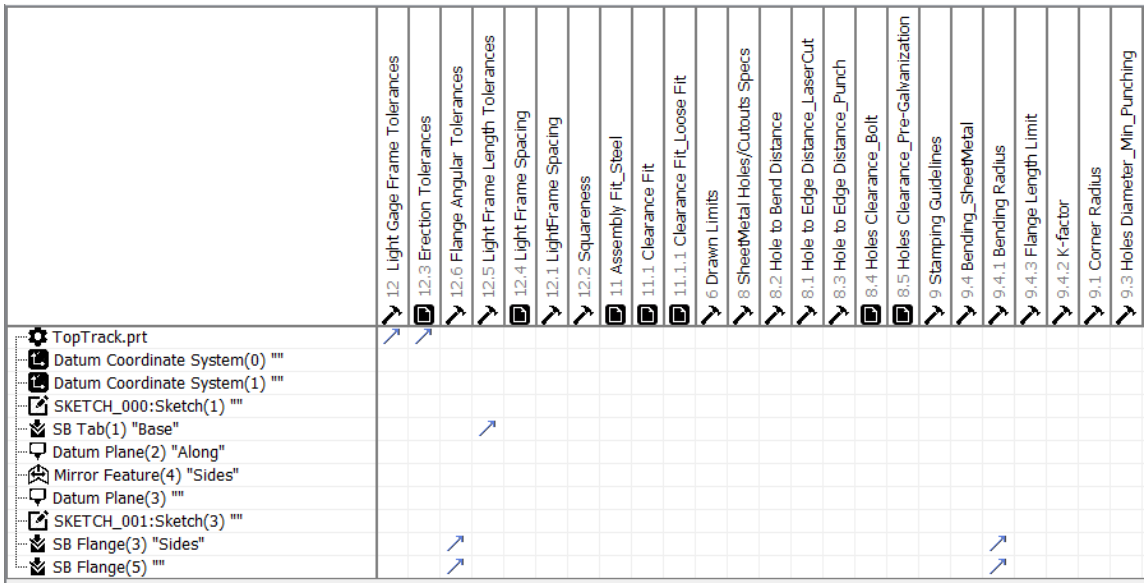


Figure 152: TopTrack: Knowledge Allocation Matrix

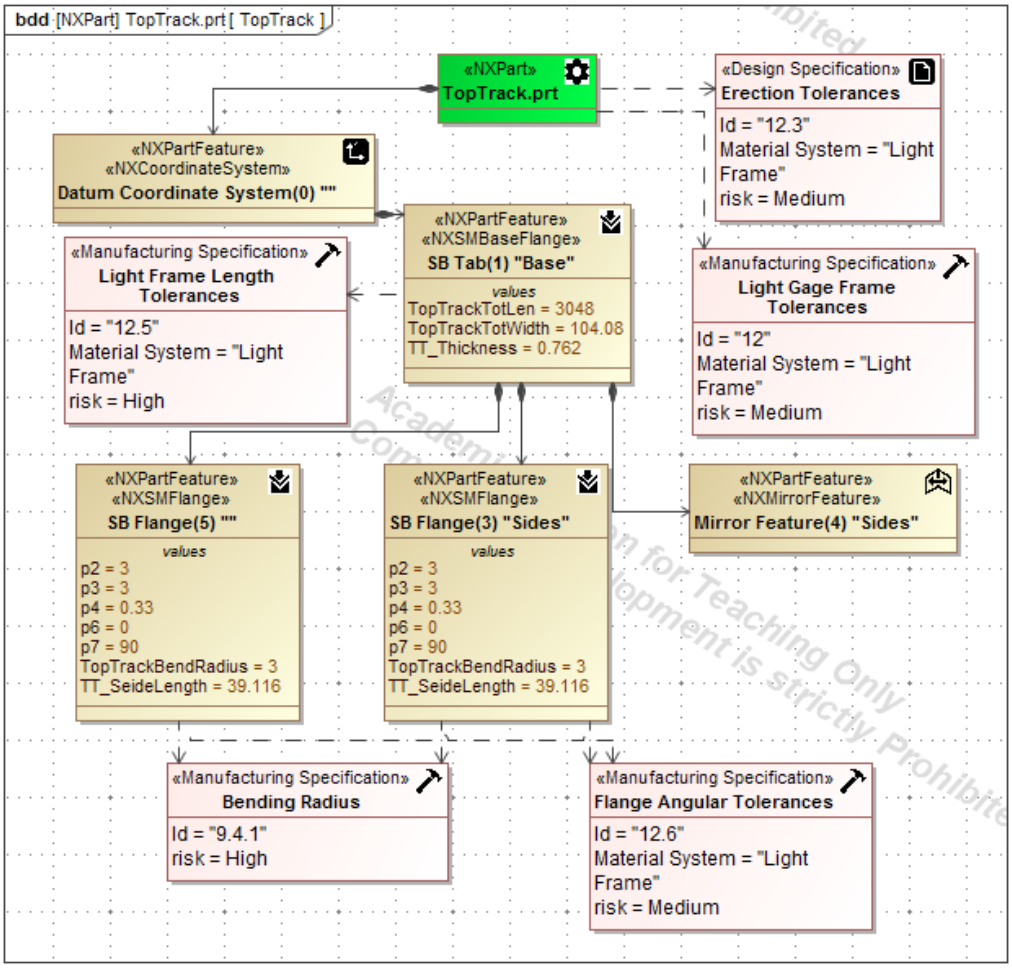


Figure 153: TopTrack with allocated Manufacturing and Design Specifications

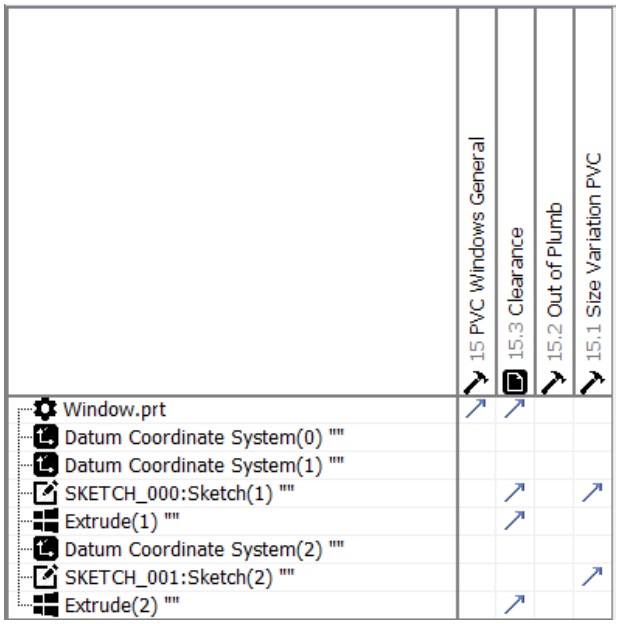


Figure 154: Window: simplified Knowledge Allocation Matrix

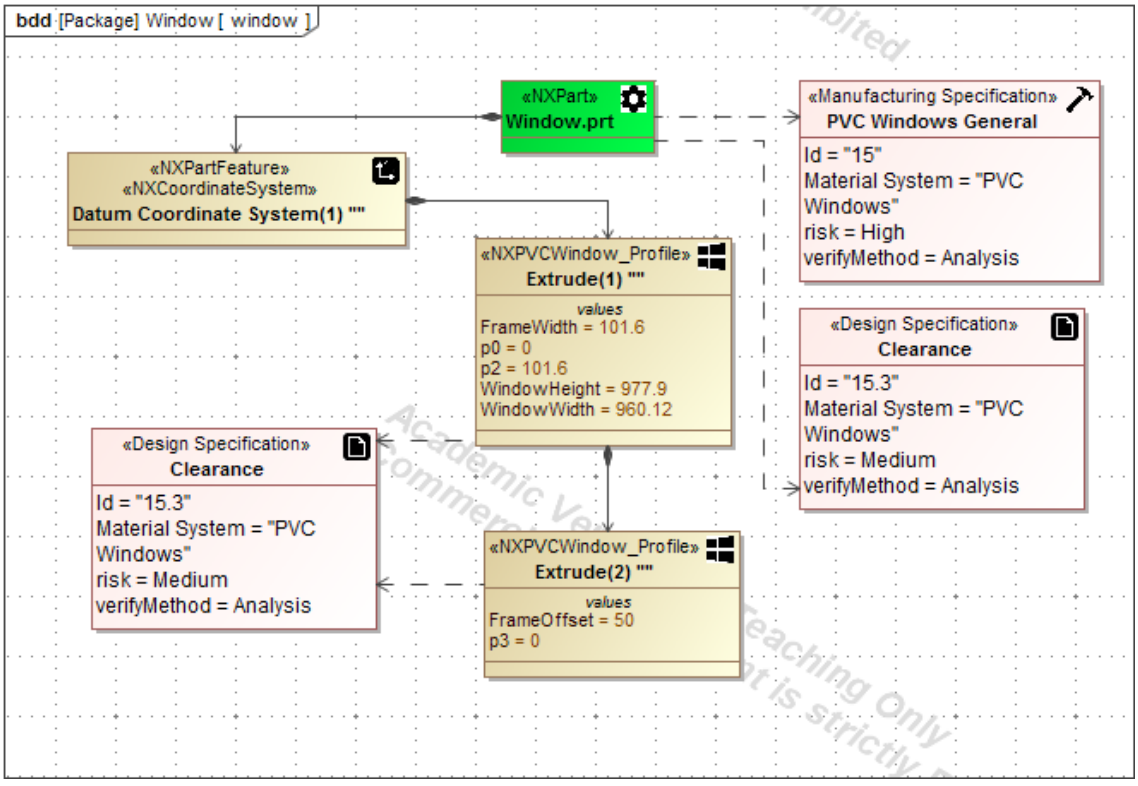


Figure 155: Window with allocated Manufacturing and Design Specifications

7.2.4. Parametric Executions of SDCT: Manufacturing and Design Specifications

The following figures show the specific analysis context diagrams of each of the eleven components of the wall assembly being studied. For each component, the results of the parametric executions of the SDCT are also offered. As in previous examples, it is important to pay attention to the performance values (gauge icon) and target values (target icon) in the parametric executions results pane. Performance values will serve as tolerances and clearances specifications for shop drawings, and target values will update their CAD classifier parameters as "new nominal" information. Furthermore, as a convention, target values can hold the same name of their original classifier, or can be called as "CentDim," which stands for "centered dimension."

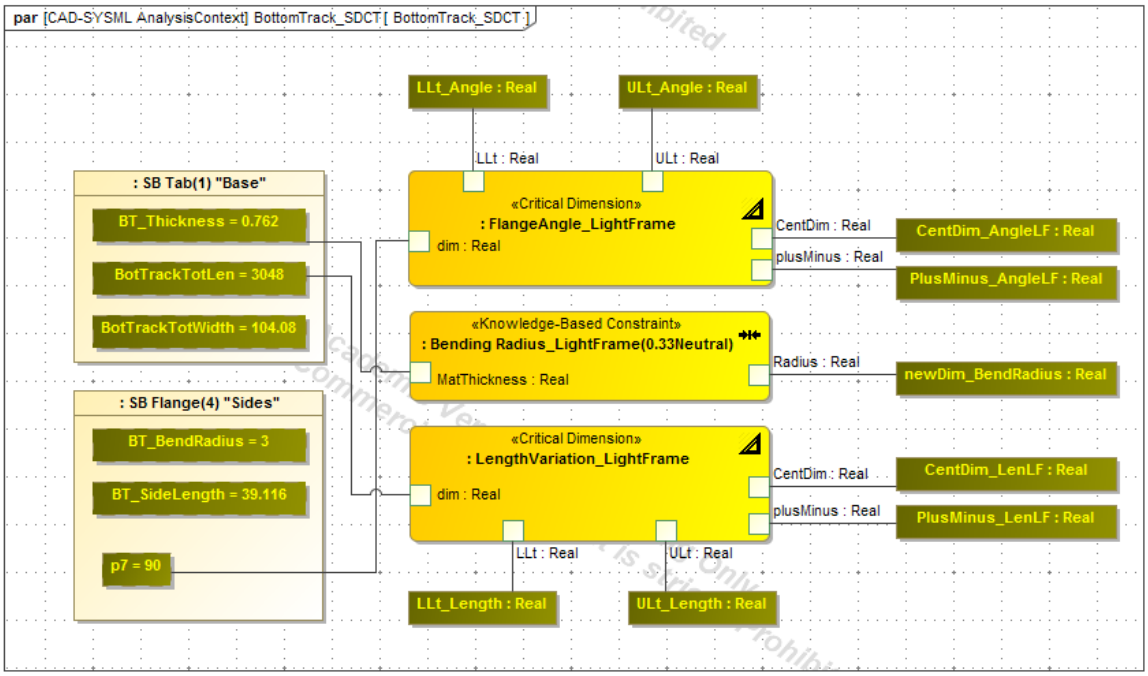


Figure 156: Parametric Execution Analysis Context for SDCT Bottom Track

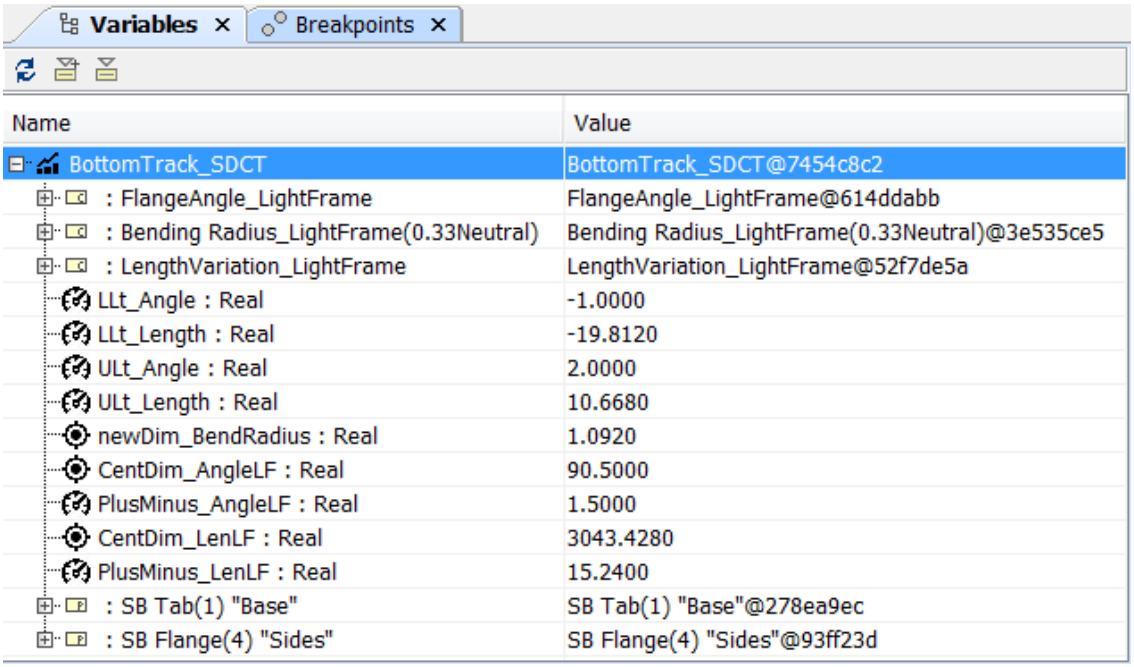


Figure 157: SDCT Parametric Execution results for Bottom Track

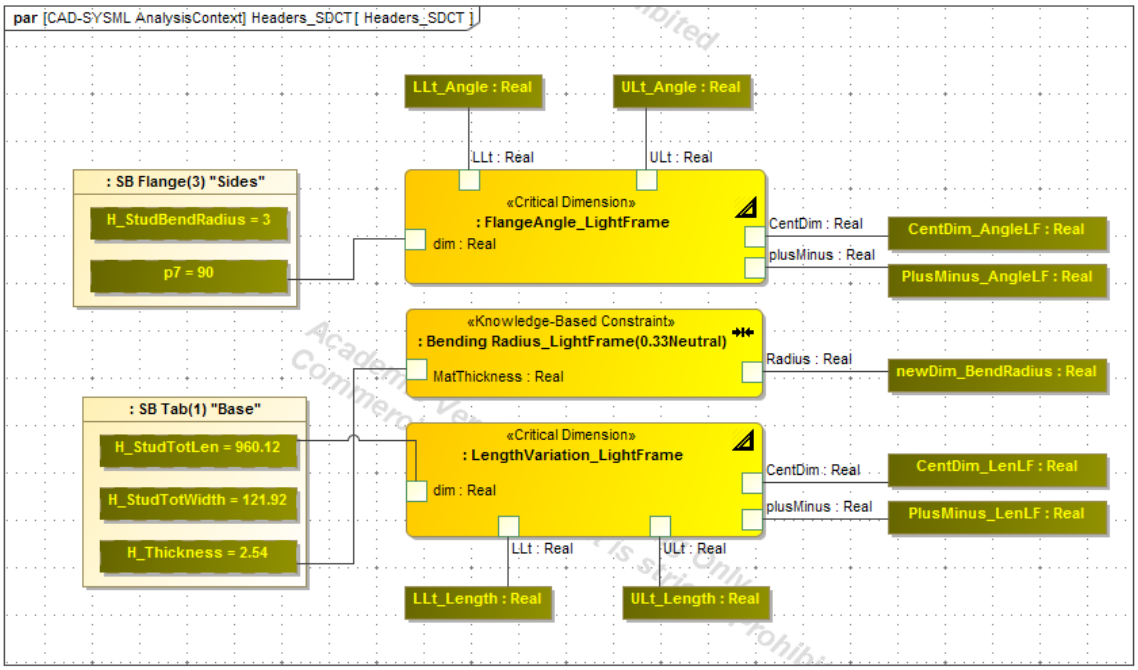


Figure 158: Parametric Execution Analysis Context for SDCT Headers

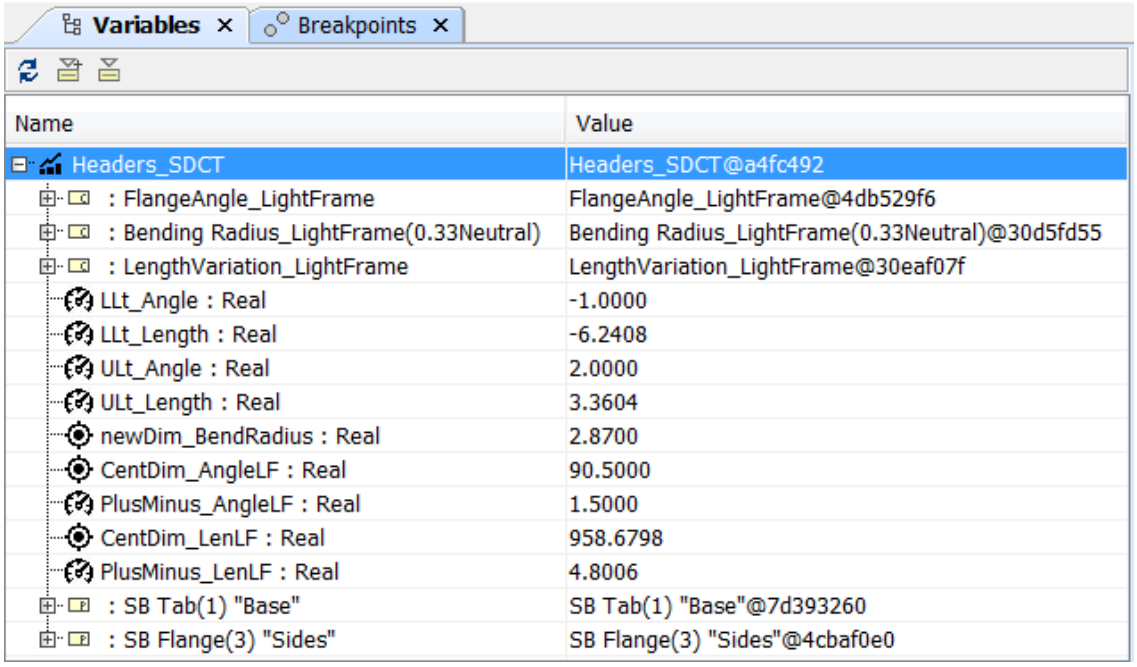


Figure 159: SDCT Parametric Execution results for Headers

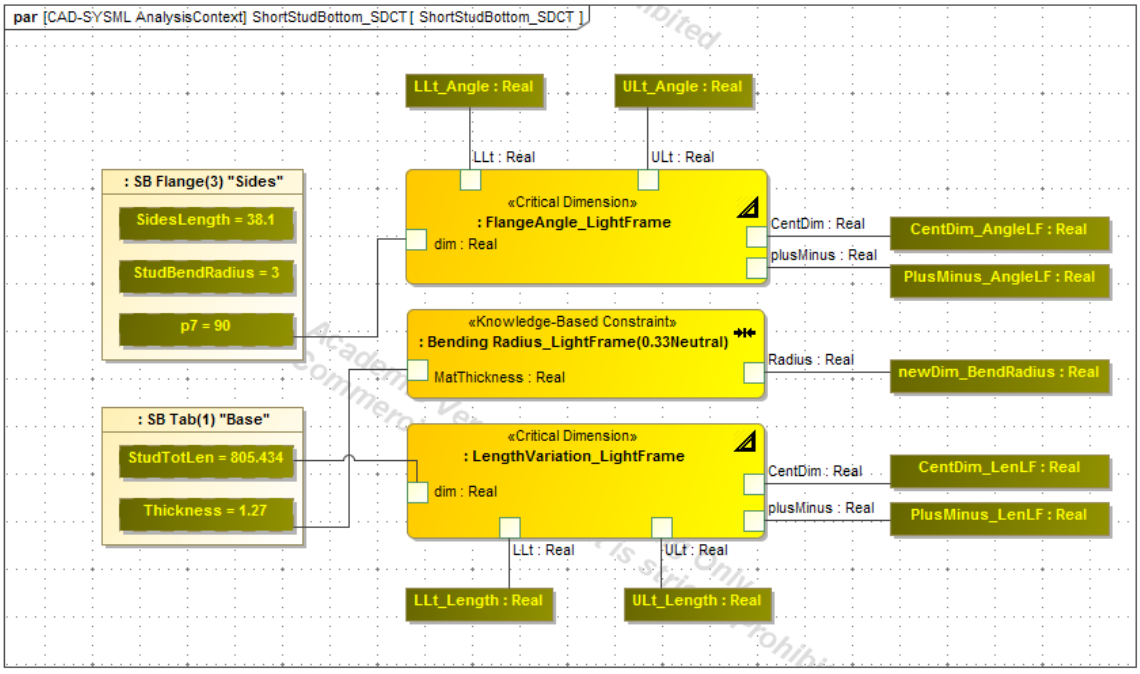


Figure 160: Parametric Execution Analysis Context for SDCT ShortStudBottom

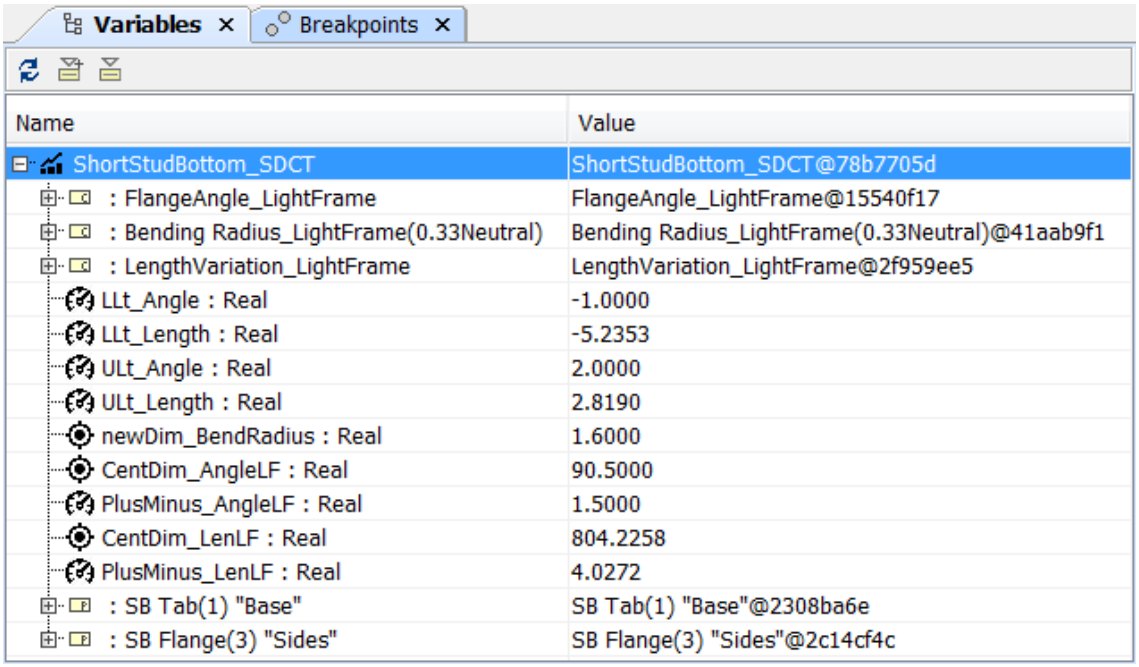


Figure 161: SDCT Parametric Execution results for Short Stud Bottom

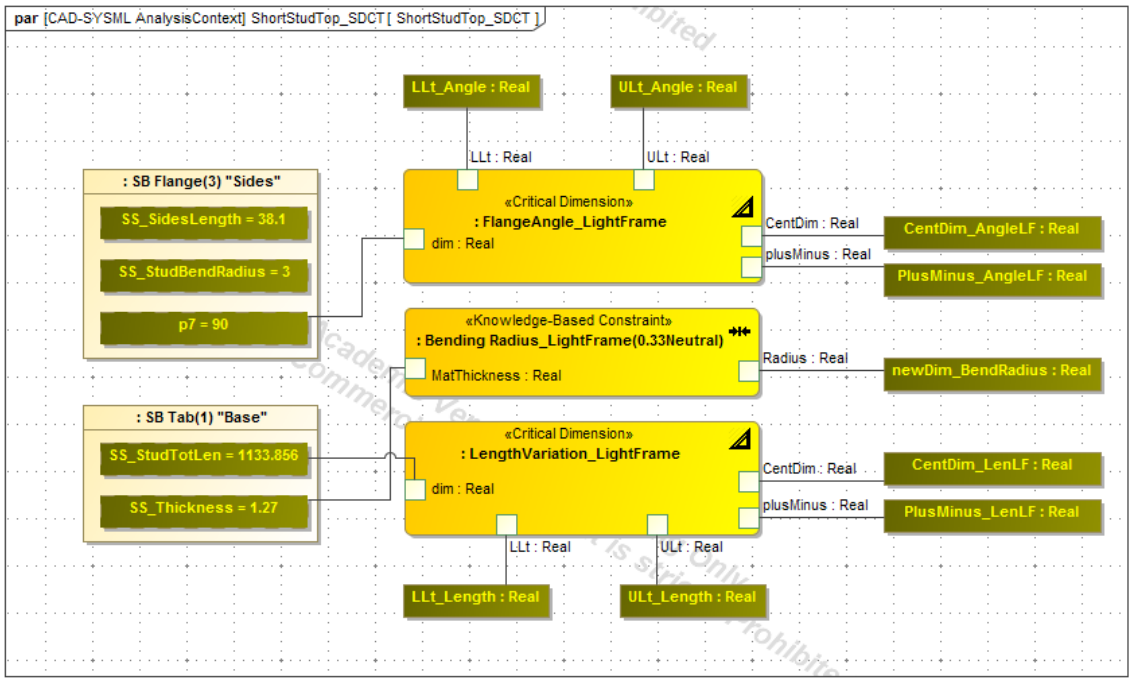


Figure 162: Parametric Execution Analysis Context for SDCT ShortStudTop

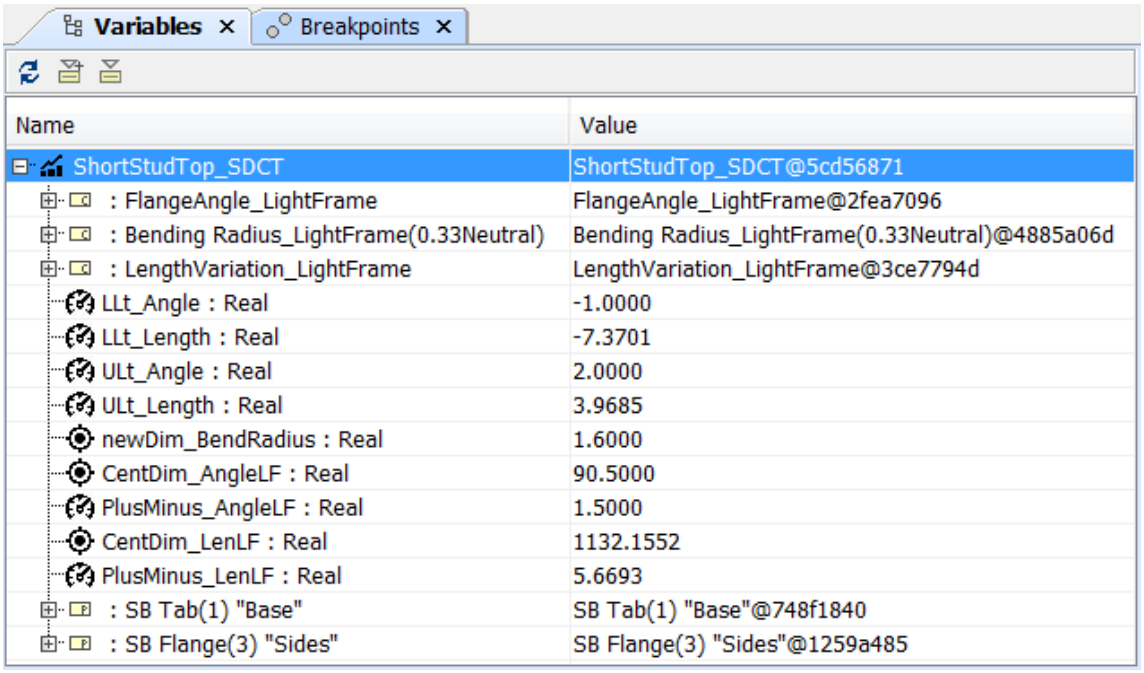


Figure 163: SDCT Parametric Execution results for ShortStudTop

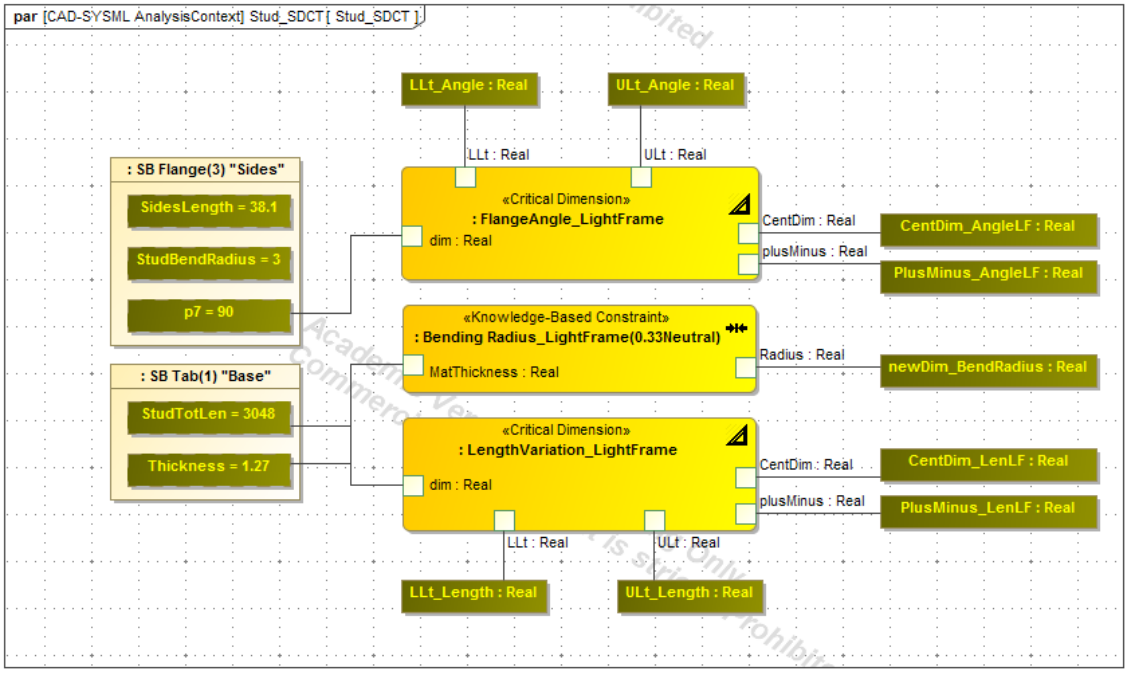


Figure 164: Parametric Execution Analysis Context for SDCT Stud

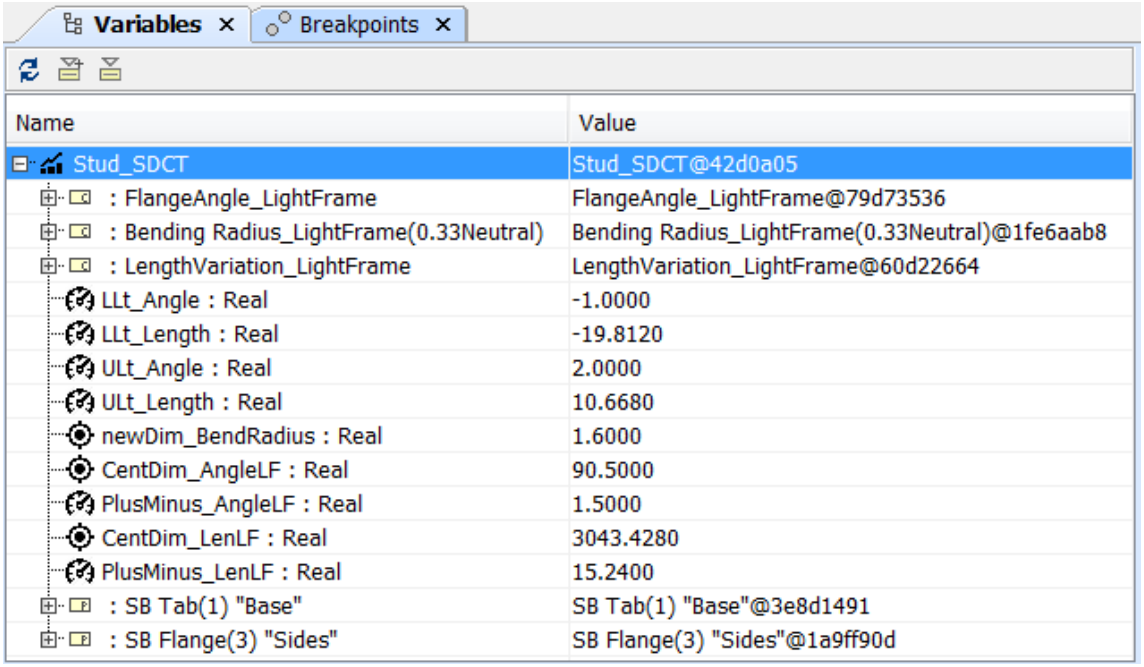


Figure 165: SDCT Parametric Execution results for Stud

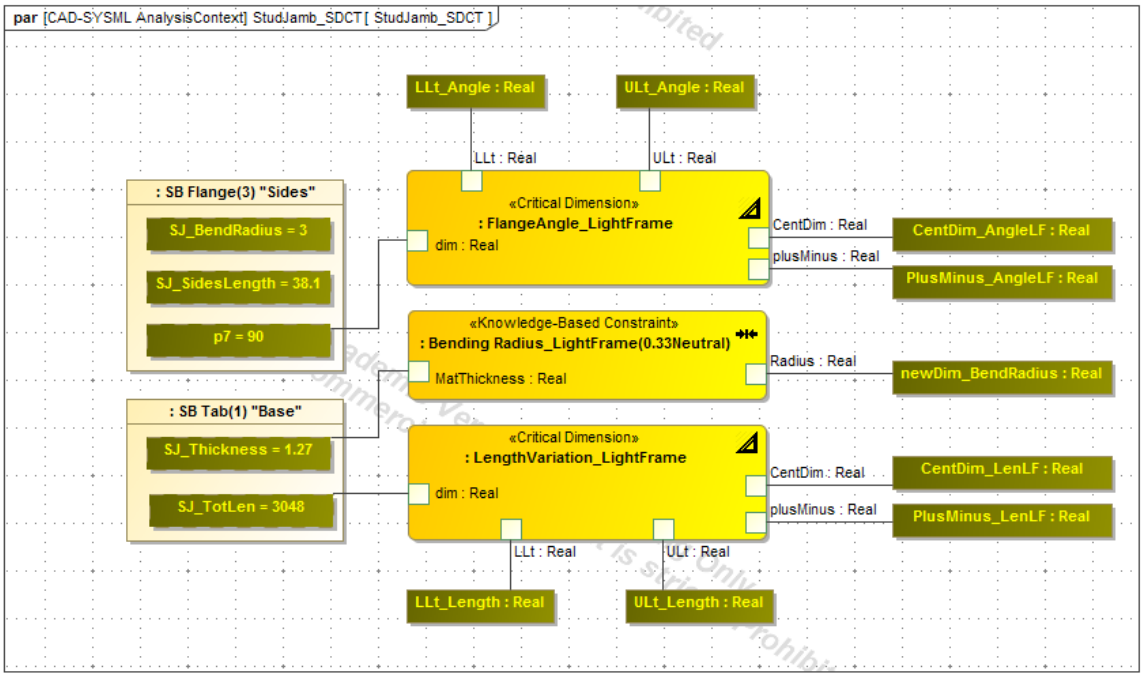


Figure 166: Parametric Execution Analysis Context for SDCT StudJamb

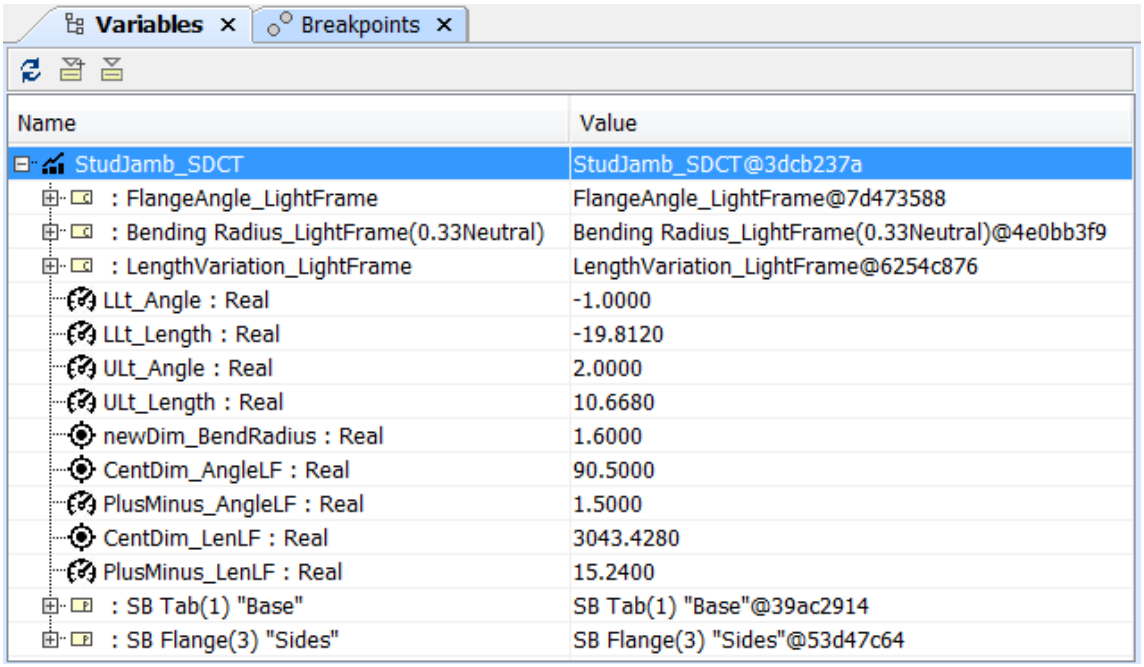


Figure 167: SDCT Parametric Execution results for Stud Jamb

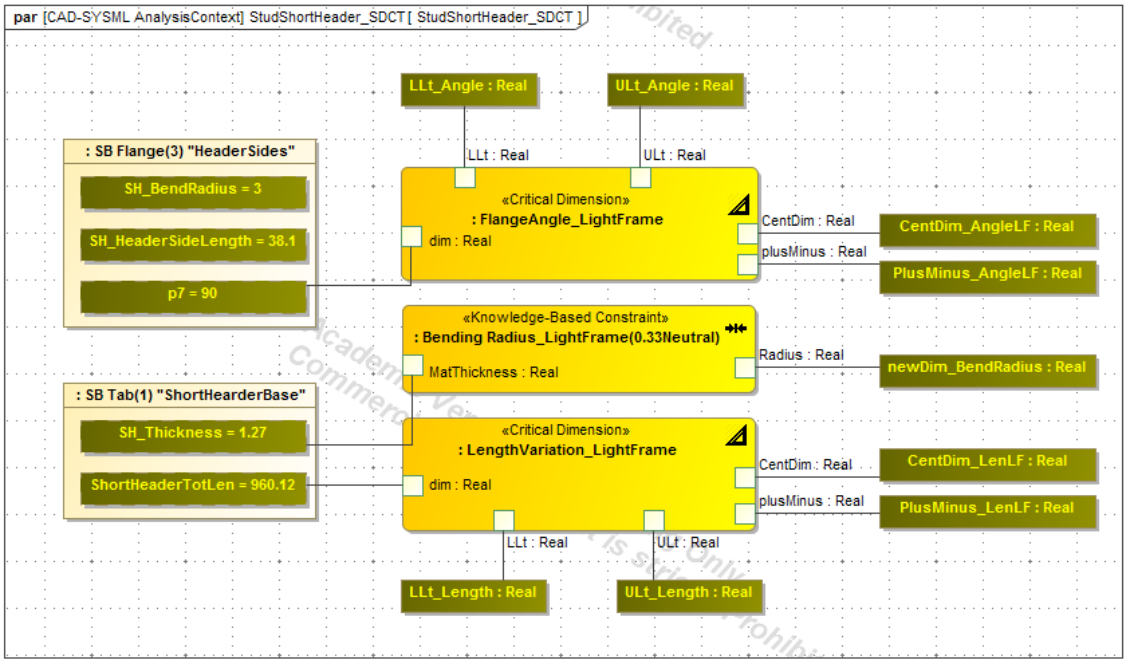


Figure 168: Parametric Execution Analysis Context for SDCT StudShortHeader

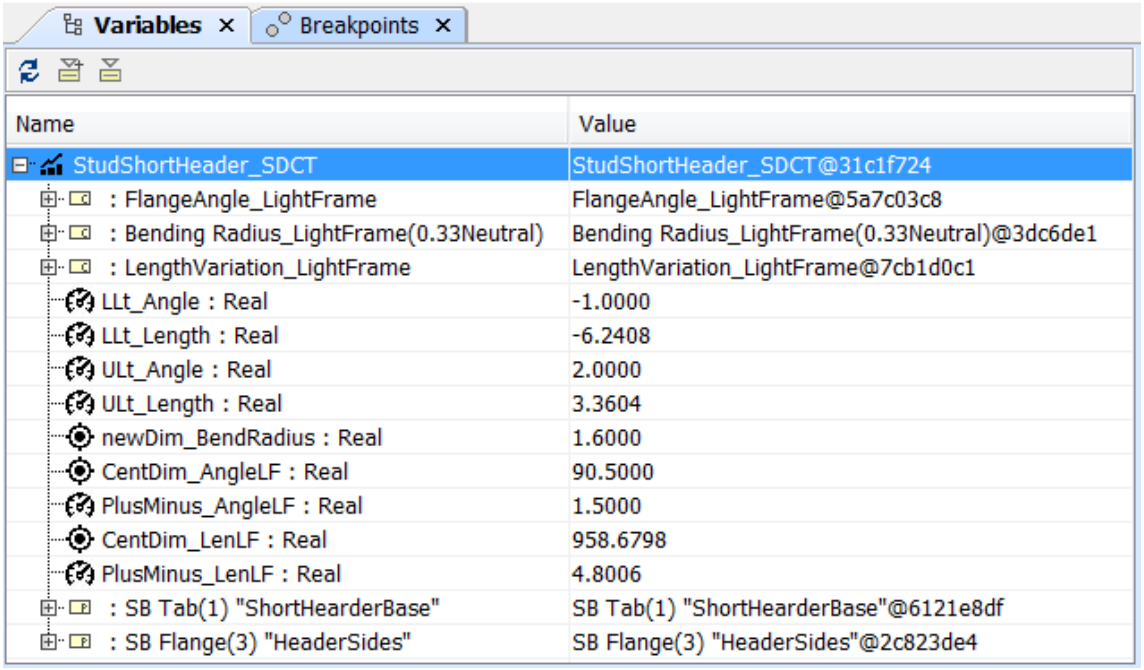


Figure 169: SDCT Parametric Execution results for Stud Short Headers

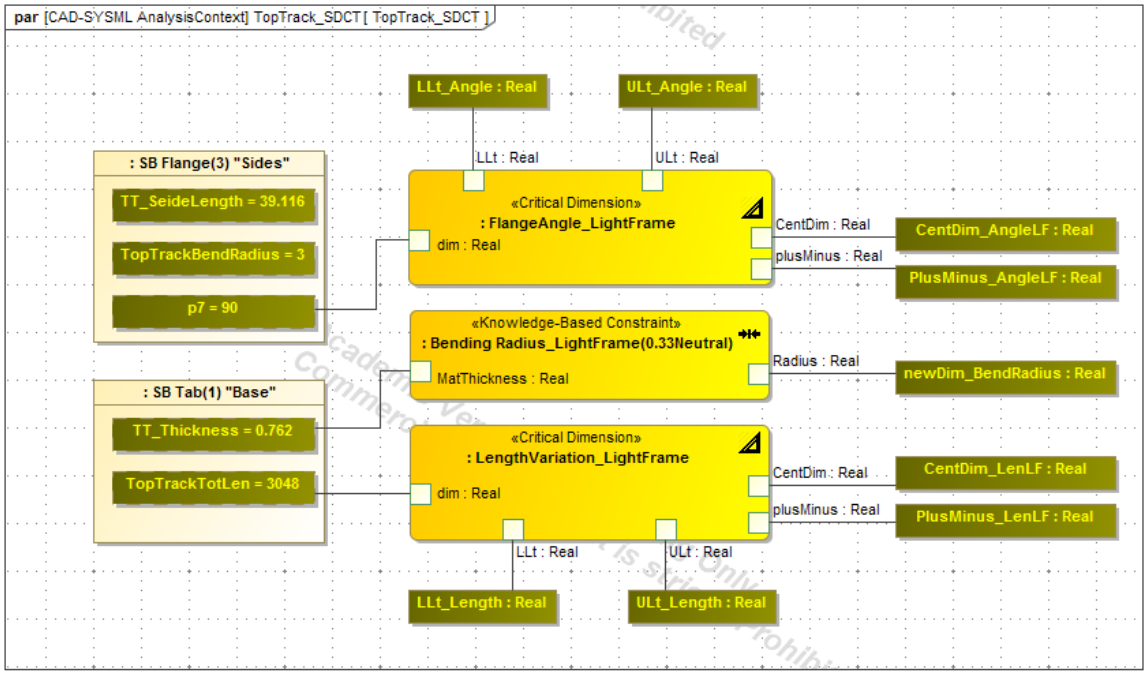


Figure 170: Parametric Execution Analysis Context for SDCT TopTrack

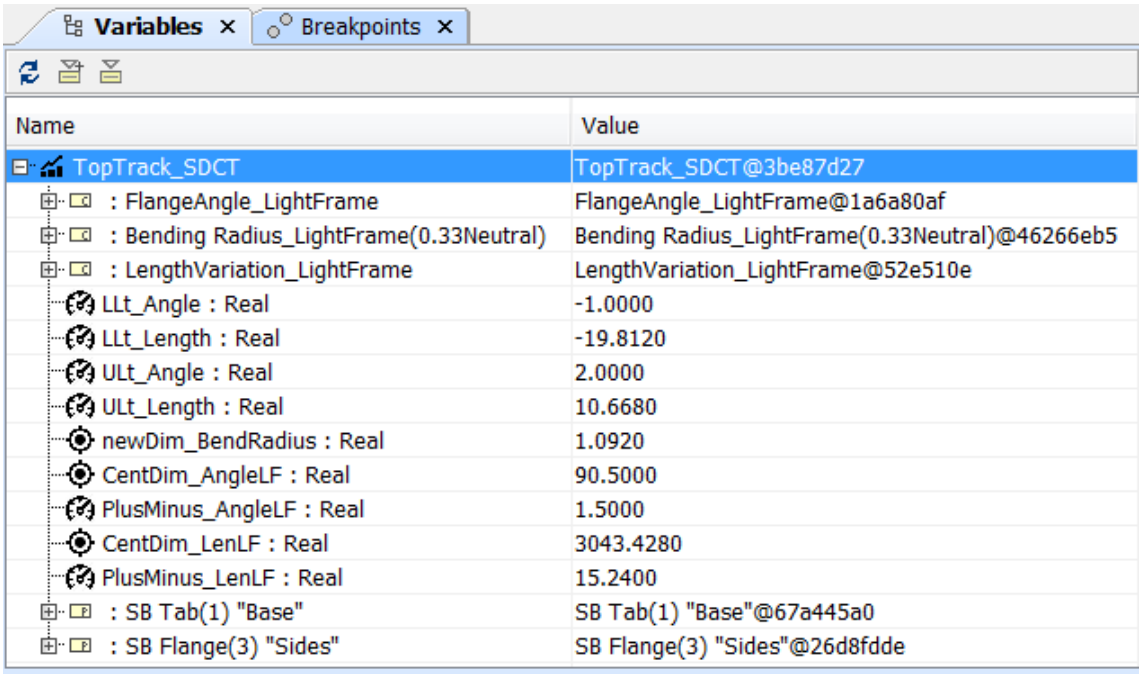


Figure 171: SDCT Parametric Execution results for Top Track

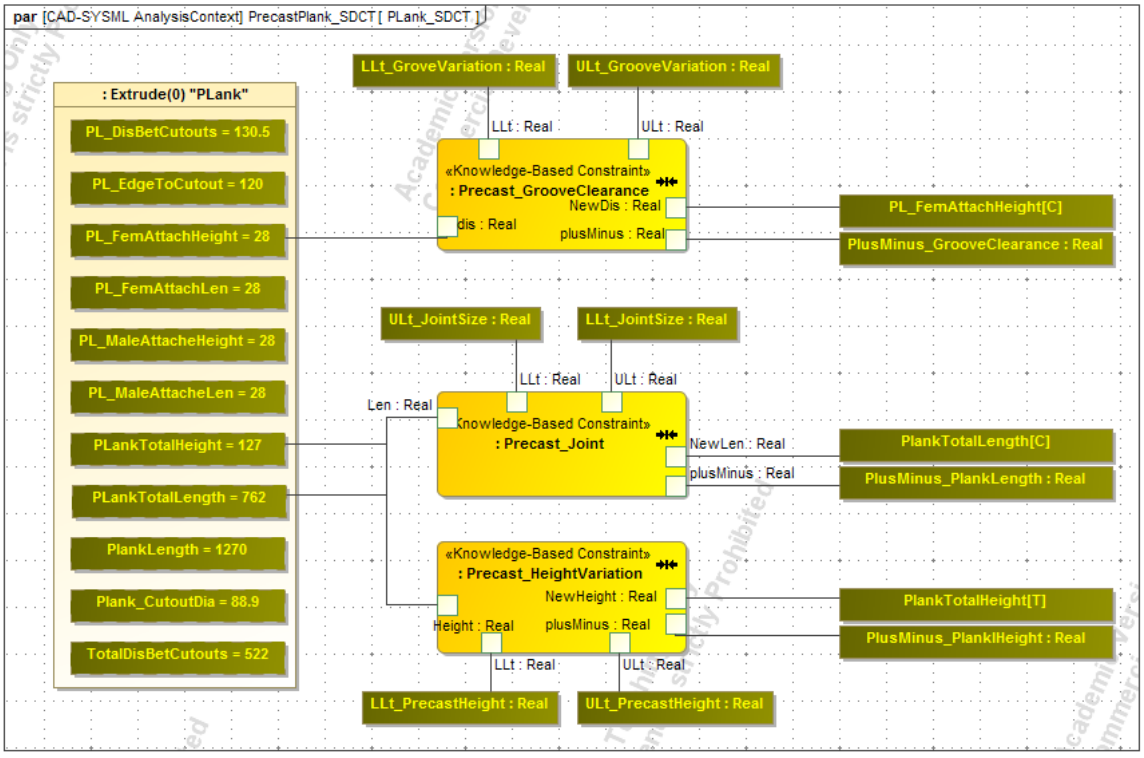


Figure 172: Parametric Execution Analysis Context for SDCT Precast Plank

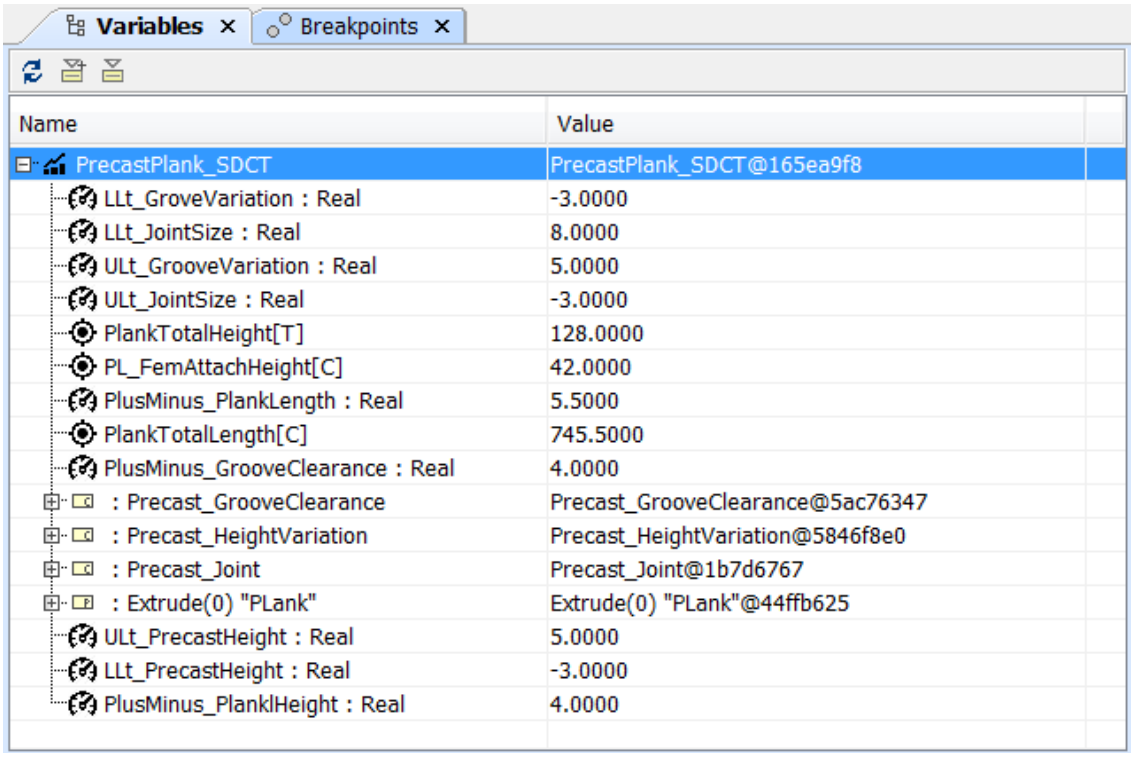


Figure 173: SDCT Parametric Execution results for Precast Plank

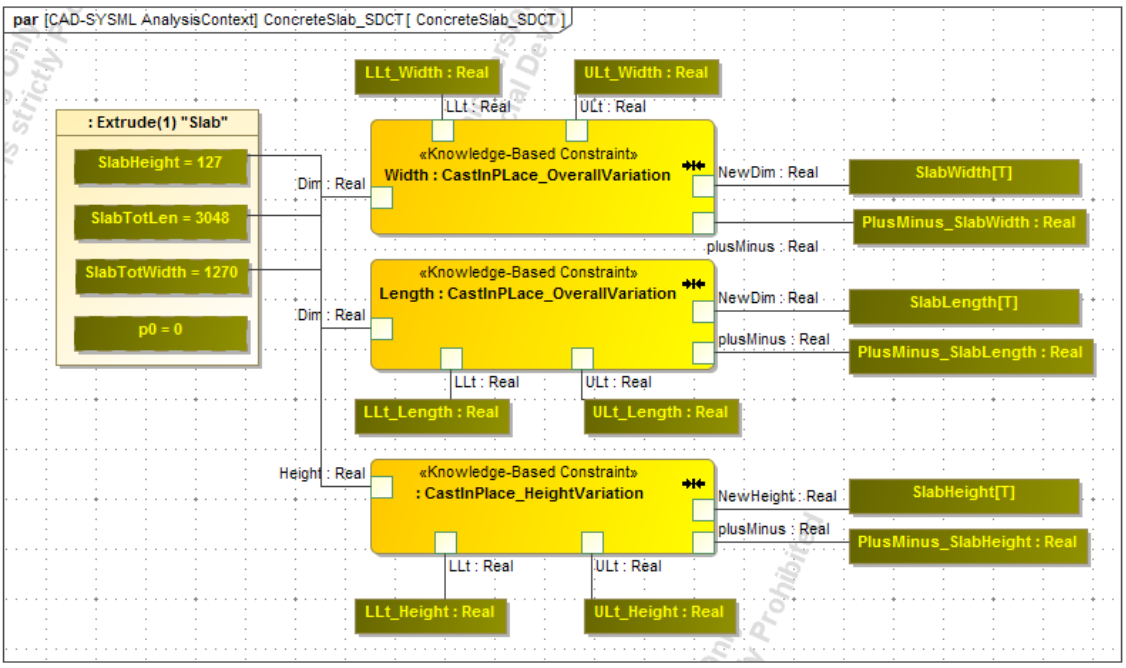


Figure 174: Parametric Execution Analysis Context for SDCT Concrete Slab

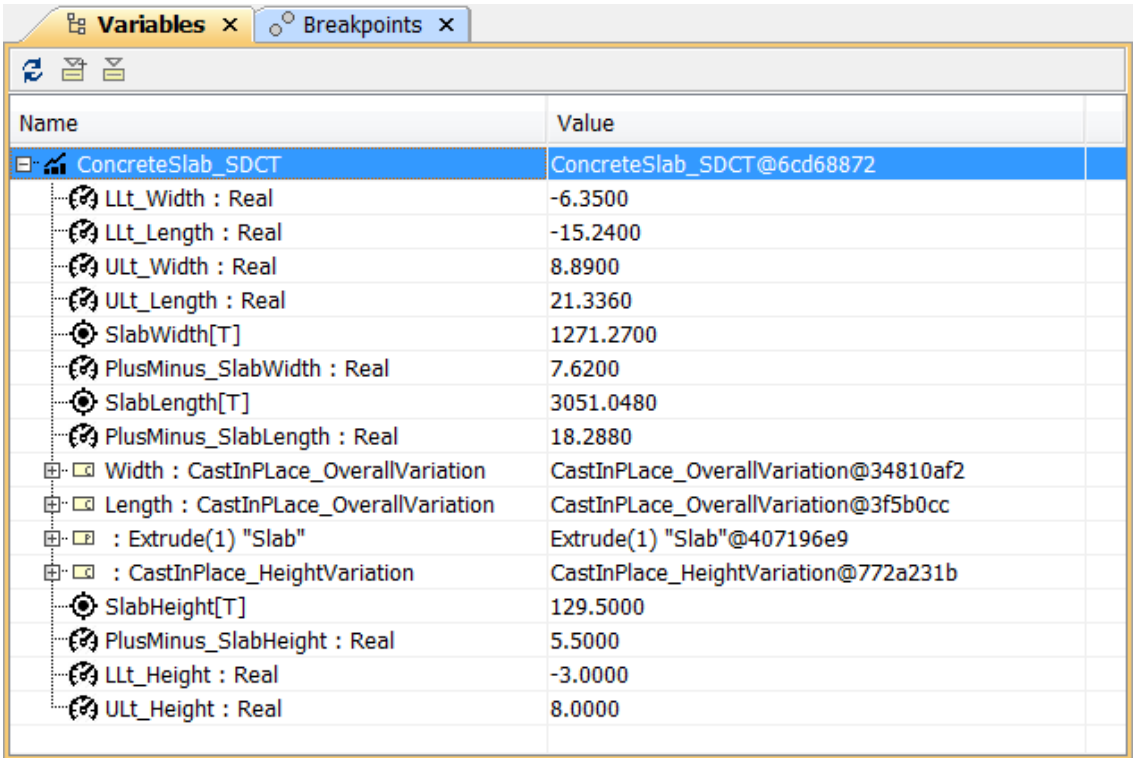


Figure 175: SDCT Parametric Execution results for Concrete Slab

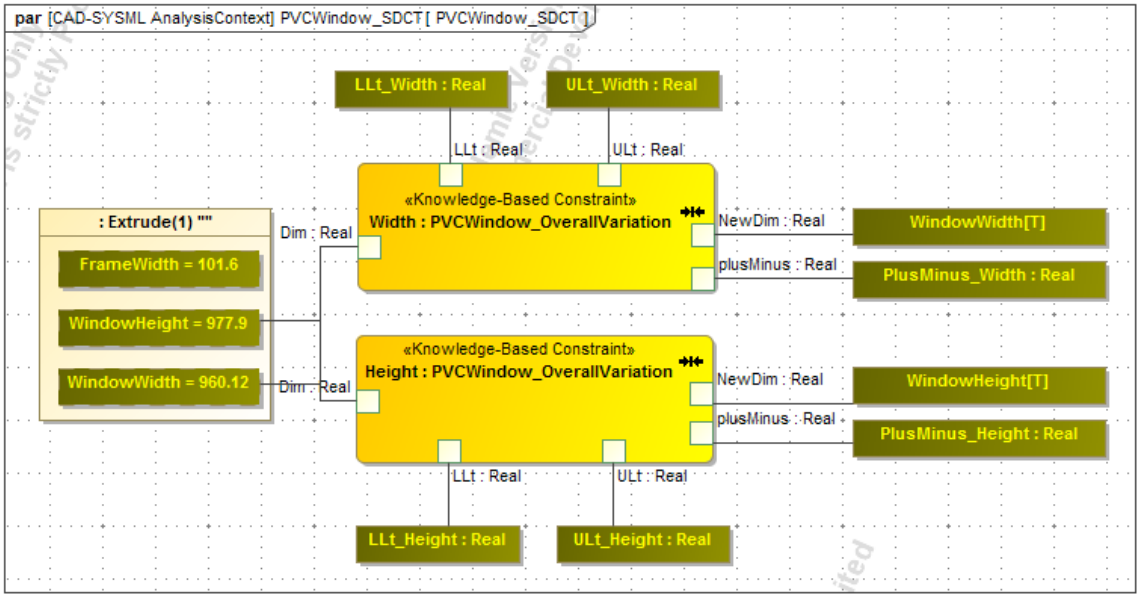


Figure 176: Parametric Execution Analysis Context for SDCT PVC Window

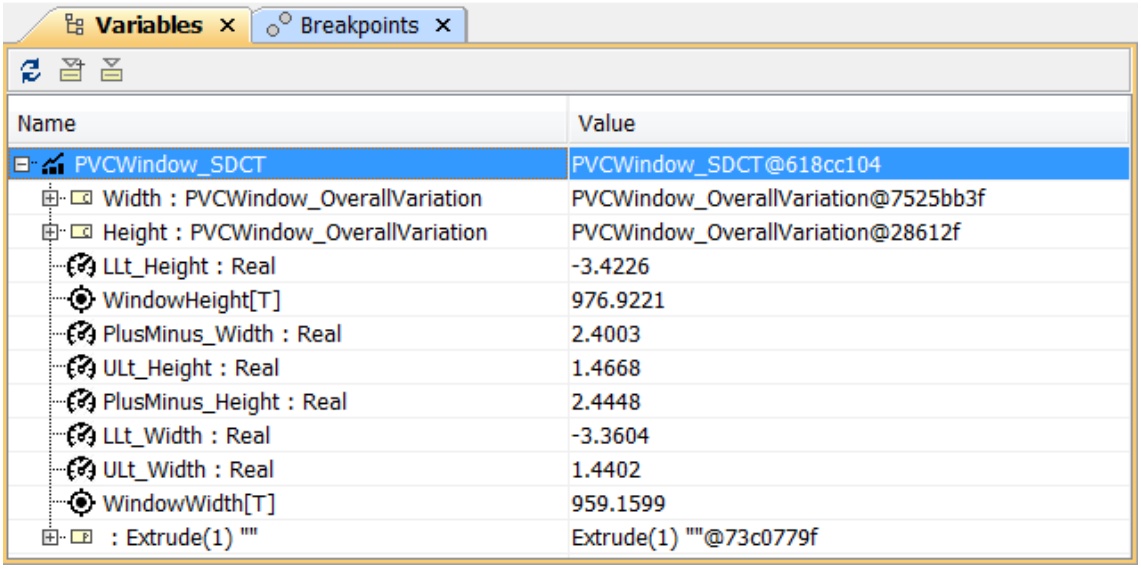


Figure 177: SDCT Parametric Execution results for PVC Window

7.2.5. Wall Assembly HCT evaluation

Figure shows a general description of the wall assembly being considered in this third case study. After all the SDCT analysis have been executed, it is time to get all components (NXPart) together to evaluate clearances and assembly tolerances by means of a HCT procedures of all the involved material systems.

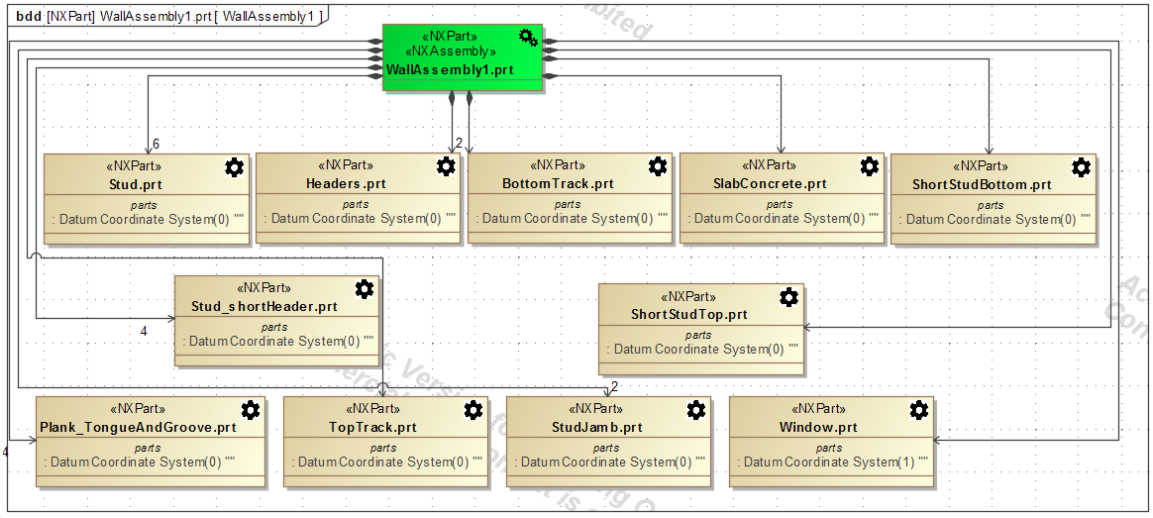


Figure 178: Wall Assembly: case study 3

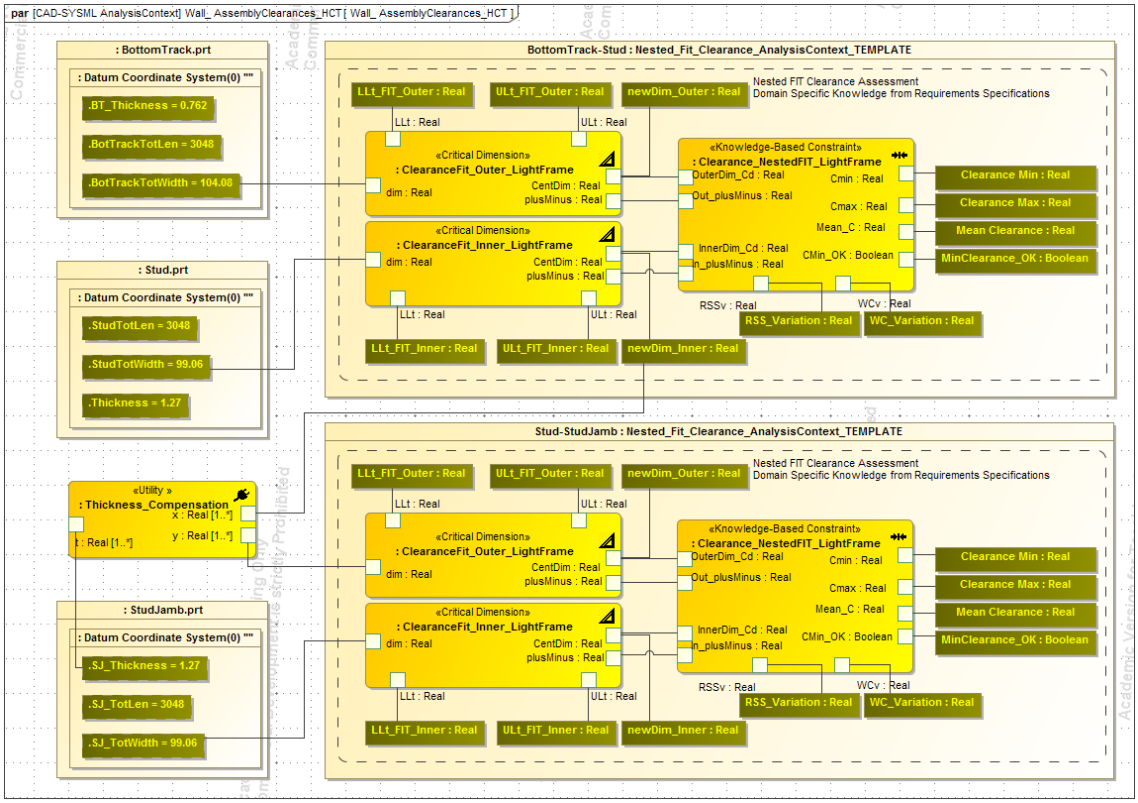


Figure 179: Wall Assembly clearances evaluation HCT Analysis Context 1: for the first group of nested components: Bottom Track: Stud: and StudJamb

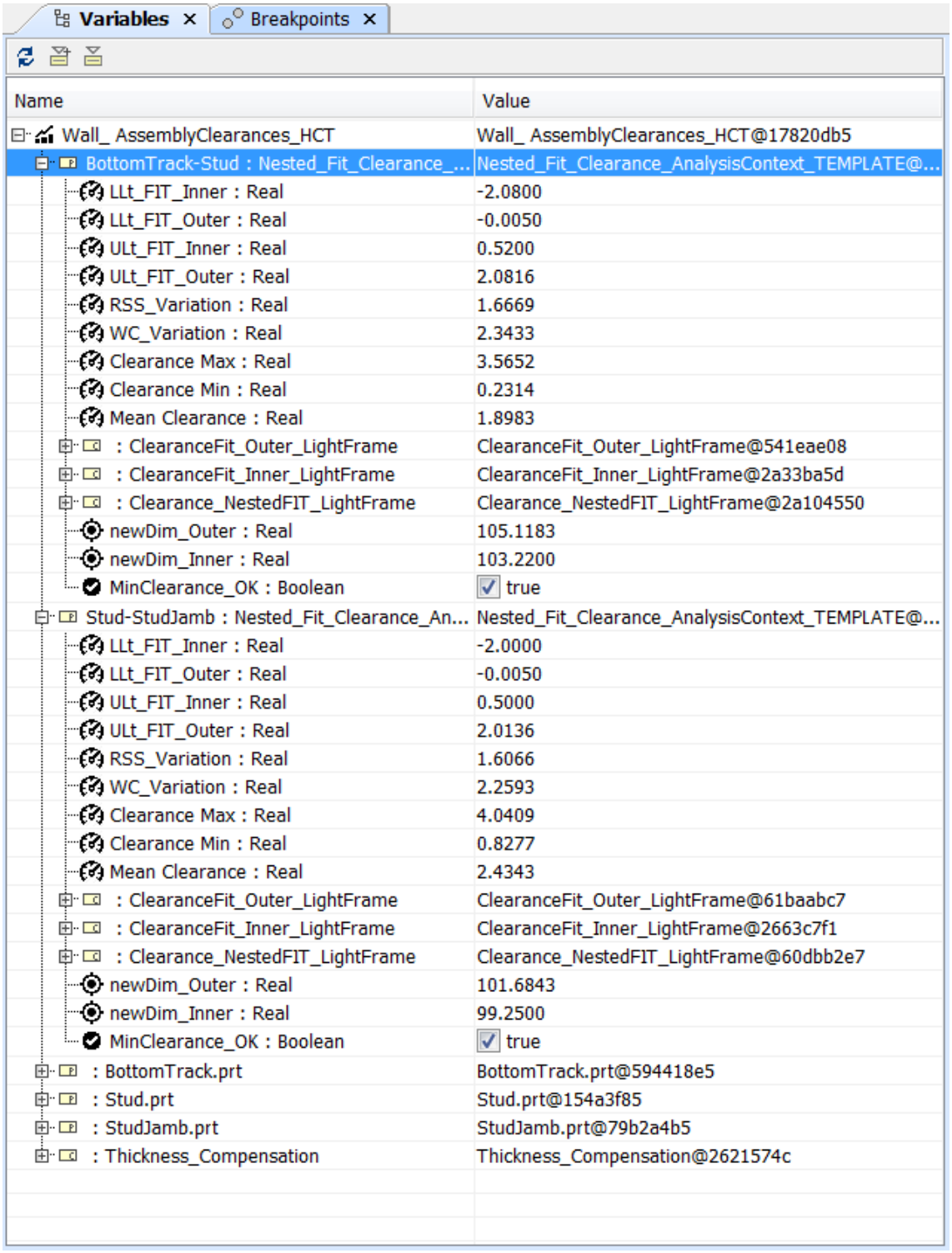


Figure 180: Wall Assembly Analysis Context 1 results: for the first group of nested components: Bottom Track: Stud: and StudJamb

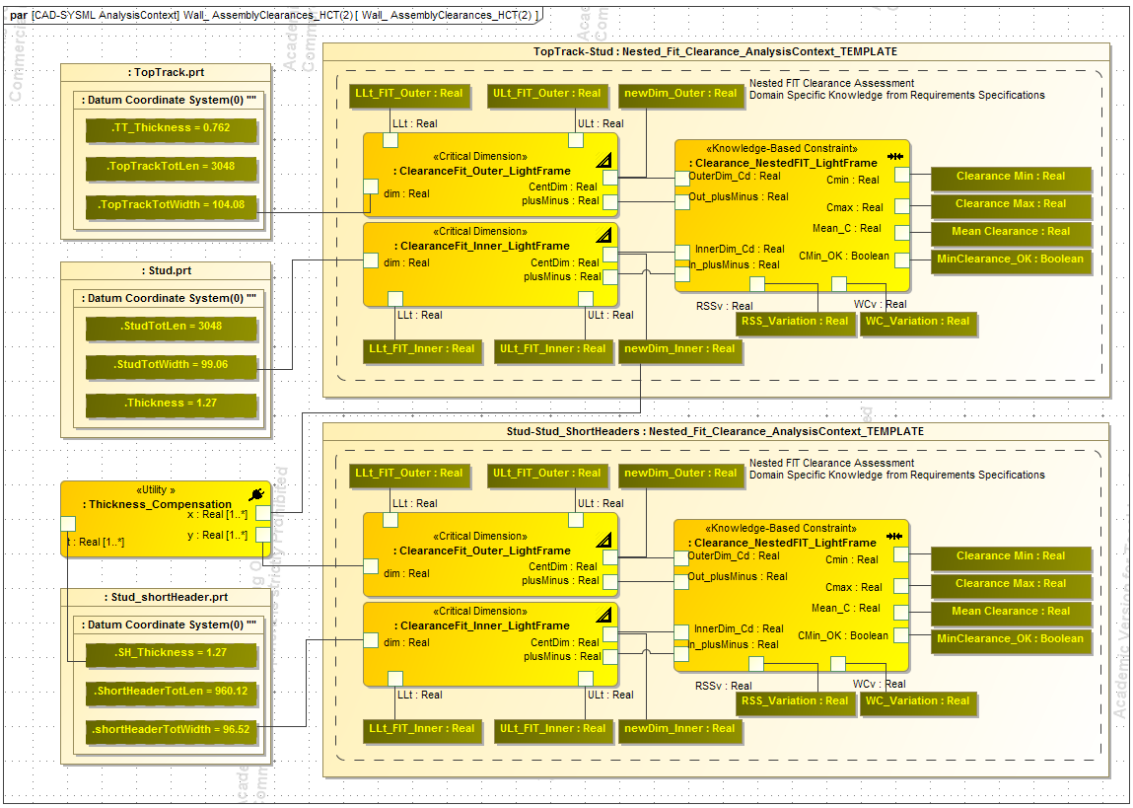


Figure 181:Wall Assembly clearances evaluation HCT Analysis Context 2: for the second group of nested components: Top Track: Stud: and Stud_ShortHeader

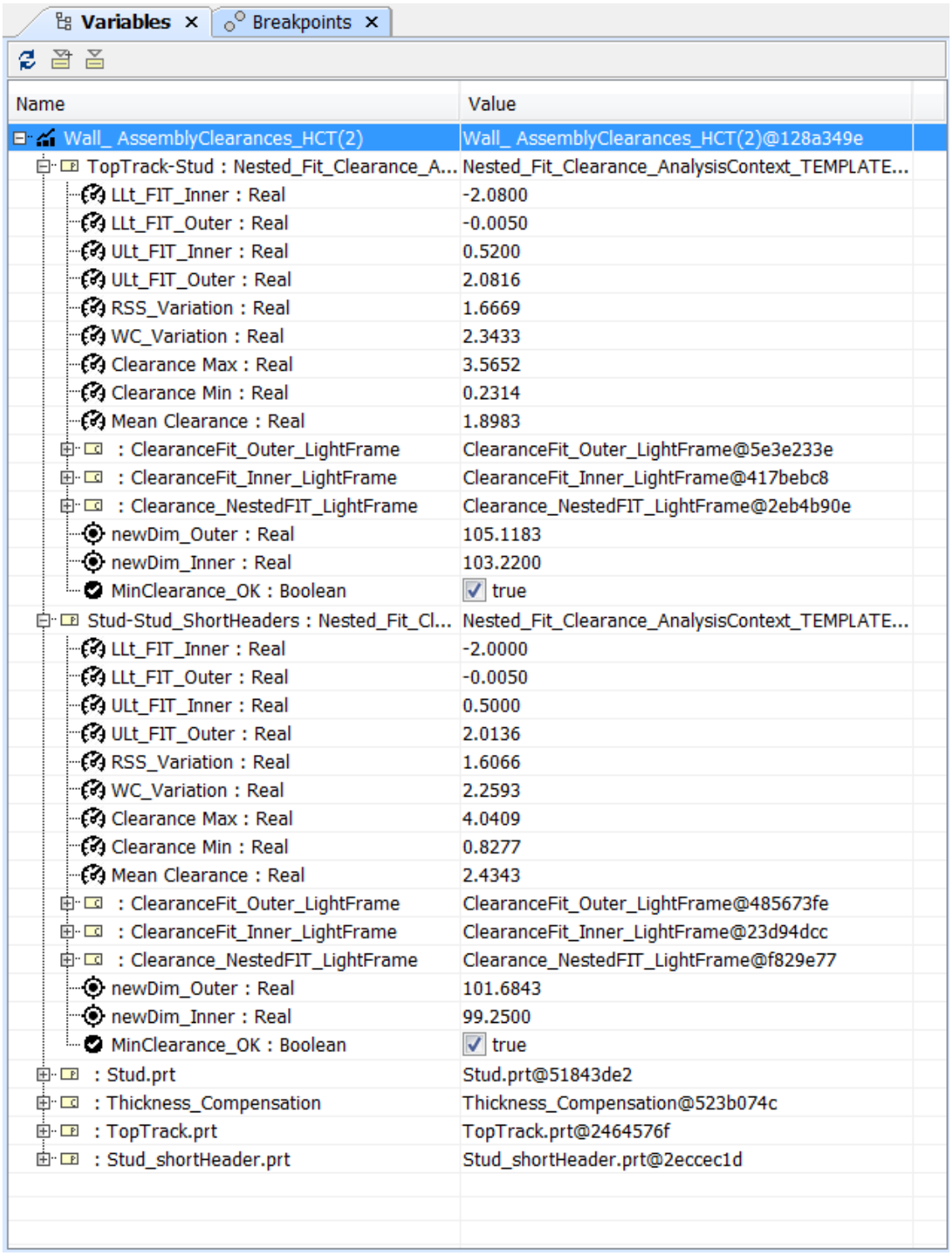


Figure 182: Wall Assembly Analysis Context 2 results: for the second group of nested components: Top Track: Stud: and Stud_ShortHeader

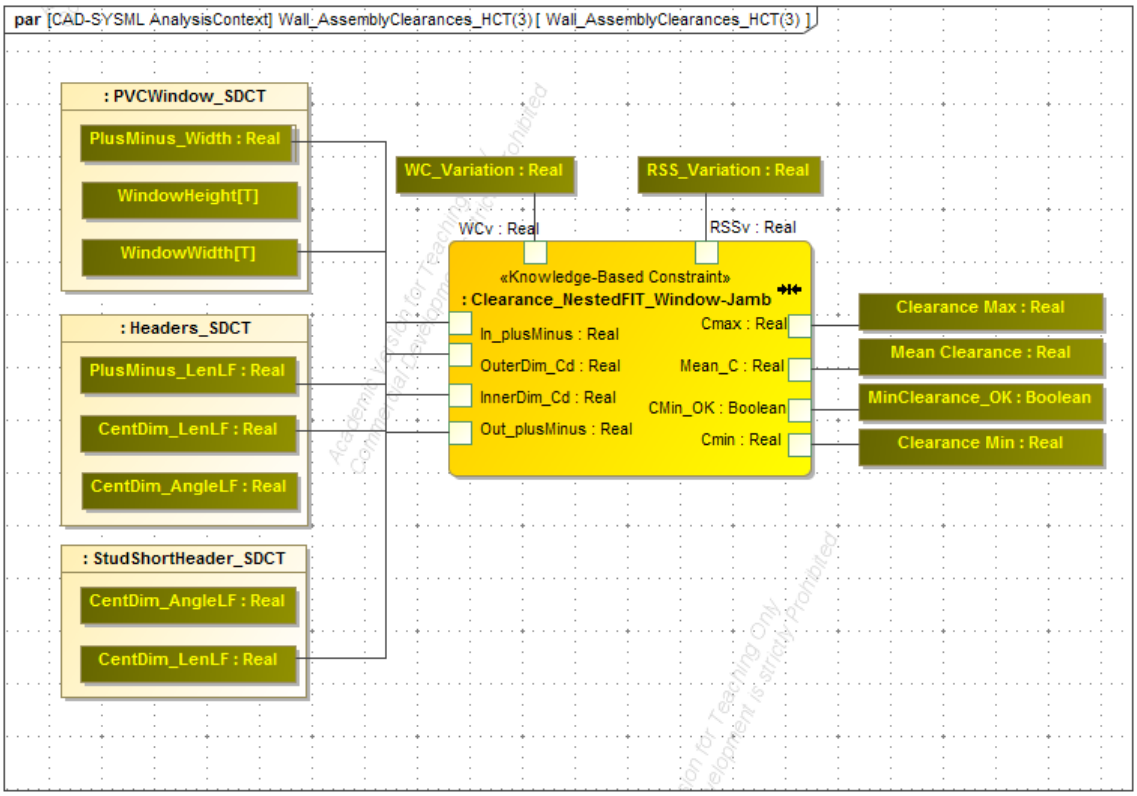


Figure 183: Wall Assembly clearances evaluation HCT Analysis Context 3 for the second group of nested components: StudShortHearders: Headers: and Window

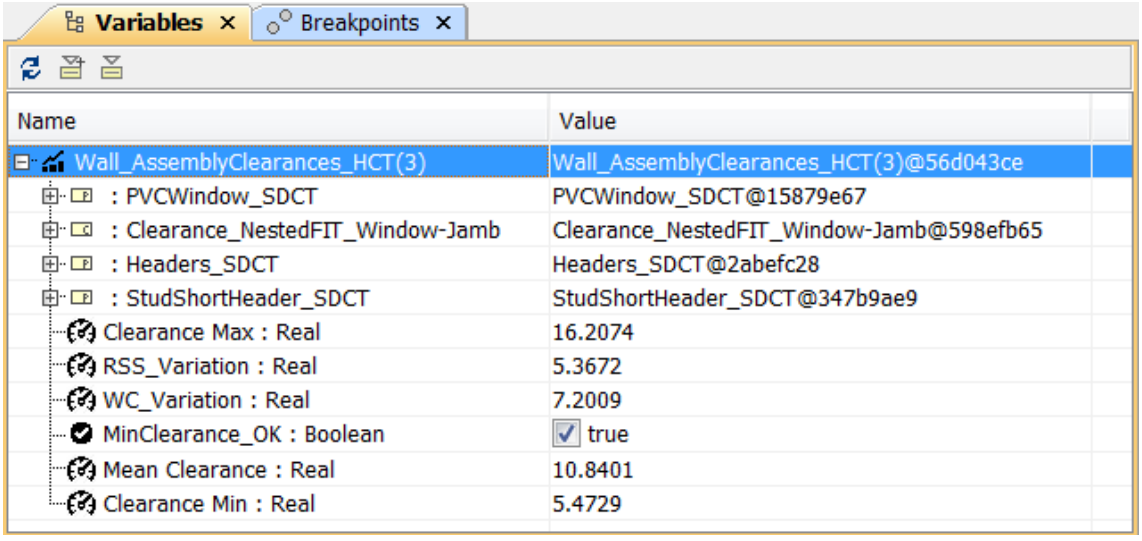


Figure 184: Wall Assembly Analysis Context three results: for the third group of nested components: StudShortHearders: Headers: and Window

7.2.6. Wall Assembly: Feature-Based Components Update based on Analyses Results

The following tables show the comparison of CAD parameters values before and after the knowledge-based manufacturing and design specifications analyses. The new values (green) have been sent to the CAD application for geometry update.

Table 8: Bottom Track CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	BT_BendRadius	3.0 // Used By ...	3	1.092
2	BT_Sheet_Metal_Bend_Radius	3	3	1.092
3	BT_SideLength	39.116	39.116	
4	BT_Thickness	0.762	0.762	
5	BotTrackTotLen	3048	3048	3043.428
6	BotTrackTotWidth	104.08	104.08	105.1183
7	SM_Validation_MIN_Punch_Tool_Cle	5	5	
8	SM_Validation_MIN_WEB_LENGTH	5	5	
9	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
10	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
11	Sheet_Metal_Material_Thickness	3	3	
12	Sheet_Metal_Neutral_Factor	0.33	0.33	
13	Sheet_Metal_Relief_Depth	3	3	
14	Sheet_Metal_Relief_Width	3	3	
15	[degrees]p7	90 // Used By ...	90	90.5

Table 9: Stud CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SM_Validation_MIN_Punch_Tool_Cle	5	5	
2	SM_Validation_MIN_WEB_LENGTH	5	5	
3	Sheet_Metal_Bend_Radius	3	3	1.6
4	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
5	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
6	Sheet_Metal_Material_Thickness	3	3	
7	Sheet_Metal_Neutral_Factor	0.33	0.33	
8	Sheet_Metal_Relief_Depth	3	3	
9	Sheet_Metal_Relief_Width	3	3	
10	SidesLength	38.1	38.1	
11	StudBendRadius	3.0 // Used By ...	3	1.6
12	StudTotLen	3048	3048	3043.428
13	StudTotWidth	99.06	99.06	103.22
14	Thickness	1.27	1.27	
15	[degrees]p7	90 // Used By ...	90	90.5

Table 10: Window CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	FrameOffset	50	50	
2	FrameWidth	101.6	101.6	
3	WindowHeight	977.9	977.9	976.9221
4	WindowWidth	960.12	960.12	959.1599
5	p2	101.6	101.6	

Table 11: Top Track CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SM_Validation_MIN_Punch_Tool_Cle	5	5	
2	SM_Validation_MIN_WEB_LENGTH	5	5	
3	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
4	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
5	Sheet_Metal_Material_Thickness	3	3	0.762
6	Sheet_Metal_Neutral_Factor	0.33	0.33	
7	Sheet_Metal_Relief_Depth	3	3	
8	Sheet_Metal_Relief_Width	3	3	
9	TT_SeideLength	39.116	39.116	
10	TT_Sheet_Metal_Bend_Radius	3	3	1.092
11	TT_Thickness	0.762	0.762	
12	TopTrackBendRadius	3.0 // Used By ...	3.0 // Used By ...	1.092
13	TopTrackTotLen	3048	3048	3043.428
14	TopTrackTotWidth	104.08	104.08	105.1183
15	[degrees]p7	90 // Used By ...	90	90.5

Table 12: StudShortHearders CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SH_BendRadius	3.0 // Used By ...	3.0 // Used By ...	1.6
2	SH_HeaderSideLength	38.1	38.1	
3	SH_Sheet_Metal_Bend_Radius	3	3	1.6
4	SH_Thickness	1.27	1.27	
5	SM_Validation_MIN_Punch_Tool_Cle	5	5	
6	SM_Validation_MIN_WEB_LENGTH	5	5	
7	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
8	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
9	Sheet_Metal_Material_Thickness	3	3	
10	Sheet_Metal_Neutral_Factor	0.33	0.33	
11	Sheet_Metal_Relief_Depth	3	3	
12	Sheet_Metal_Relief_Width	3	3	
13	ShortHeaderTotLen	960.12	960.12	970
14	[degrees]p7	90 // Used By ...	90 // Used By ...	90.5
15	shortHeaderTotWidth	96.52	96.52	99.25

Table 13: Concrete Slab CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SlabHeight	127	127	129.5
2	SlabTotLen	3048	3048	3051.048
3	SlabTotWidth	1270	1270	1271.27

Table 14: Headers CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	H_Sheet_Metal_Bend_Radius	3	3	2.87
2	H_SidesLength	22.098	22.098	
3	H_StudBendRadius	3.0 // Used By ...	3	2.87
4	H_StudTotLen	960.12	960.12	970
5	H_StudTotWidth	121.92	121.92	
6	H_Thickness	2.54	2.54	
7	SM_Validation_MIN_Punch_Tool_Cle	5	5	
8	SM_Validation_MIN_WEB_LENGTH	5	5	
9	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
10	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
11	Sheet_Metal_Material_Thickness	3	3	
12	Sheet_Metal_Neutral_Factor	0.33	0.33	
13	Sheet_Metal_Relief_Depth	3	3	
14	Sheet_Metal_Relief_Width	3	3	
15	[degrees]p7	90 // Used By ...	90	90.5

Table 15: Short Stud Bottom CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SM_Validation_MIN_Punch_Tool_Cle	5	5	
2	SM_Validation_MIN_WEB_LENGTH	5	5	
3	Sheet_Metal_Bend_Radius	3	3	1.6
4	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
5	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
6	Sheet_Metal_Material_Thickness	3	3	
7	Sheet_Metal_Neutral_Factor	0.33	0.33	
8	Sheet_Metal_Relief_Depth	3	3	
9	Sheet_Metal_Relief_Width	3	3	
10	SidesLength	38.1	38.1	
11	StudBendRadius	3.0 // Used By ...	3	1.6
12	StudTotLen	805.434	805.434	804.2258
13	StudTotWidth	99.06	99.06	103.22
14	Thickness	1.27	1.27	
15	[degrees]p7	90 // Used By ...	90	90.5

Table 16: Short Stud Top CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SM_Validation_MIN_Punch_Tool_Cle	5	5	
2	SM_Validation_MIN_WEB_LENGTH	5	5	
3	SS_SidesLength	38.1	38.1	
4	SS_StudBendRadius	3.0 // Used By ...	3.0 // Used By ...	1.6
5	SS_StudTotLen	1133.856	1133.856	1132.15
6	SS_StudTotWidth	99.06	99.06	103.22
7	SS_Thickness	1.27	1.27	
8	Sheet_Metal_Bend_Radius	3	3	1.6
9	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
10	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
11	Sheet_Metal_Material_Thickness	3	3	
12	Sheet_Metal_Neutral_Factor	0.33	0.33	
13	Sheet_Metal_Relief_Depth	3	3	
14	Sheet_Metal_Relief_Width	3	3	
15	[degrees]p7	90 // Used By ...	90	90.5

Table 17: Precast Plank CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	PL_DisBetCutouts	TotalDisBetCutouts/4	130.5	
2	PL_EdgeToCutout	120	120	
3	PL_FemAttachHeight	28	28	42
4	PL_FemAttachLen	28	28	
5	PL_MaleAttacheHeight	28	28	
6	PL_MaleAttacheLen	28	28	
7	PL_TotHeight	(PLankTotalHeight-PL	49.5	
8	PL_TotLen	PLankTotalLength	762	
9	PLankTotalHeight	127	127	128
10	PLankTotalLength	762	762	745.5
11	PlankLength	1270	1270	
12	Plank_CutoutDia	88.9	88.9	

Table 18: Stud Jamb CAD results update

N	Expression Name	Expression Original Definition	Expression Original Value	Expression Knowledge-Allocated Value
1	SJ_BendRadius	3.0 // Used By ...	3.0 // Used By ...	1.6
2	SJ_Sheet_Metal_Bend_Radius	3	3	1.6
3	SJ_SidesLength	38.1	38.1	
4	SJ_Thickness	1.27	1.27	
5	SJ_TotLen	3048	3048	3043.428
6	SJ_TotWidth	99.06	99.06	99.25
7	SM_Validation_MIN_Punch_Tool_Clear	5	5	
8	SM_Validation_MIN_WEB_LENGTH	5	5	
9	Sheet_Metal_Flat_In_Corner_Value	0.1	0.1	
10	Sheet_Metal_Flat_Out_Corner_Value	0.1	0.1	
11	Sheet_Metal_Material_Thickness	3	3	
12	Sheet_Metal_Neutral_Factor	0.33	0.33	
13	Sheet_Metal_Relief_Depth	3	3	
14	Sheet_Metal_Relief_Width	3	3	
15	[degrees]p7	90 // Used By ...	90	90.5

The following list offers a summary of the material-specific design and manufacturing constraints automatically applied to the CAD geometry of the present wall assembly case study:

- SDCT Precast joint clearances were specified
- SDCT Precast groove features were applied
- SDCT bending radii were calculated based on material thickness and geometry context
- SDCT contours of nested shapes were calculated to avoid clashes
- SDCT Cast-in-place dimensional variability was applied
- SDCT Window clearances were calculated
- HCT clearances were calculated and applied for all nested components
- HCT tolerances of clearances based on RSS, WC, were calculated

- HCT plus/minus, mean, and centered tolerances were calculated for all assembly clearances

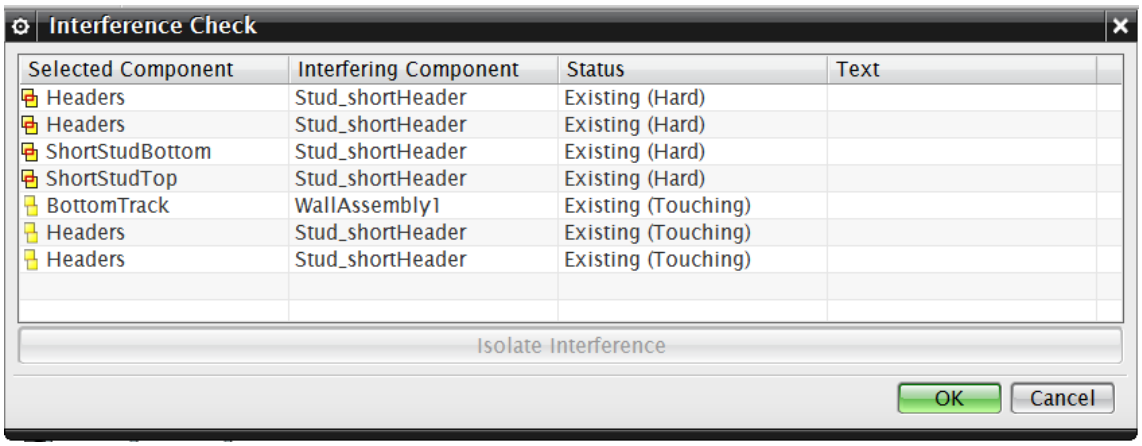


Figure 185: Wall assembly interference check after geometry update

After the analyses have been performed and the updated values have been sent back to the CAD application, a new interference check analysis will be executed (Figure). It's important to mention that no CAD value was modified within the NX environment. All modification come directly from the SysML model and has been executed by the developed application. Figure shows the evident difference in the number of interferences found before and after manufacturing analysis. Twenty-six hard clashes were identified before the analyses and only four after analyses. Also, the number of touching conditions was dramatically reduced. The focus of this dissertation was not CAD clashes, as they can be produced by many factors (bad design, for instance). However, the identification of clashes and further automatic correction based on applied knowledge of clearances and tolerances is not trivial. The hard interferences reduction happened because manufacturing knowledge was prescribed for each component. This knowledge was represented as manufacturing specifications, corrected bending radii, components lengths, and angles of flanges, among others. Also, touching conditions were

reduced because assembly knowledge, as with design specifications, was applied to several critical mating conditions of the assembly. This knowledge prescribed proper clearances for nested elements, installation of windows, and clearances for material system boundaries. Some examples of these improvements can be seen in Figure .

Interference Check before manufacturing analyses and tolerances and clearances allocation

Interference Check after manufacturing analyses and tolerances and clearances allocation

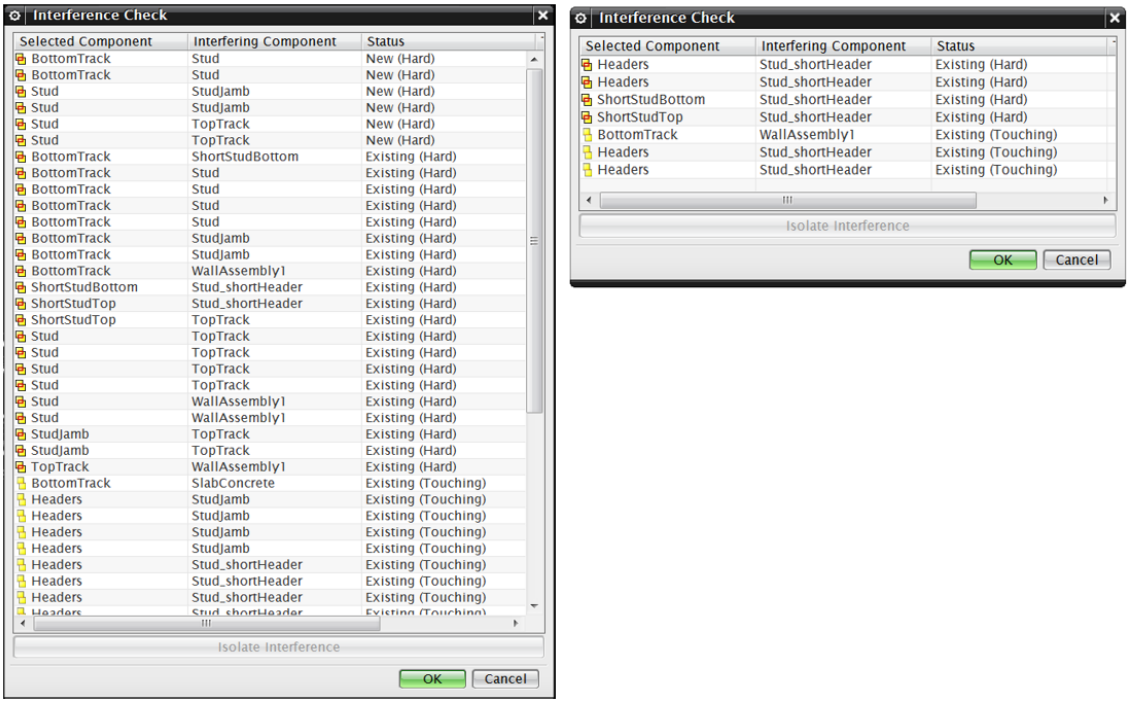


Figure 186: Interference check comparison: before and after manufacturing analysis

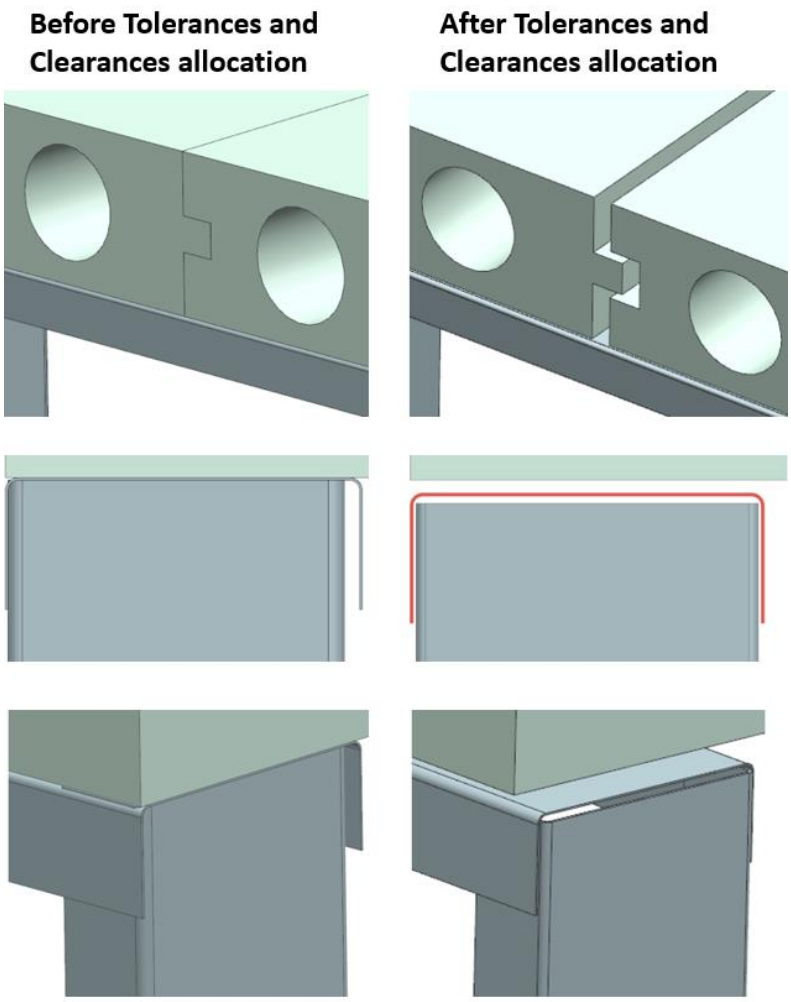


Figure 187: detail examples of some geometric updated after manufacturing and tolerances analyses

CHAPTER 8: Evaluation of the Proposed Implementation

Chapter 8 presents the evaluation of the application developed in this dissertation. the first subsection restates the three case studies included in the evaluation. Then, the four evaluation methods are introduced. For each of these methods, a brief explanation about the specific evaluation goal is offered. Further in the chapter, the positive aspects of the developed application are enumerated. At the end, unanticipated issues and resolutions of compliance analysis implementation and the potential for future advancement of the implementation are discussed

8.1. General

This dissertation has developed a framework for the integration of construction tolerance data into CAD models and three case studies to demonstrate and evaluate the main functionalities of the proposed implementation:

1. A cylindrical fit study among three telescopic components with two different material systems (cast-in-place concrete and steel);
2. A multi-feature, four-component, single-material (sheet metal) assembly of an architectural photovoltaic racking structure called QuadPod;
3. A light-gauge wall assembly with eleven components and four concurrent material systems (cast-in-place concrete, precast concrete, light-gauge framing, and PVC windows);

The main focus of each test has been the execution of five critical stages of the proposed framework: (1) feature-based structural decomposition, (2) knowledge

acquisition/allocation, (3) parametric execution, (4) knowledge validation, and (5) CAD geometry update. The goal for these case studies has been to assess the functionality of the automated geometric modifications implemented to demonstrate the integration of geometric and process knowledge to CAD through SysML. The set of elements and rules to perform this analysis have been created as a SysML profile or Domain Specific Language (DSL). This profile or DSL defines the elements, languages and processes from which to form and evaluate the studied models. The approach is based on the assembly> part> feature> parameter> value standard to describe geometry as seen in most solid modeling applications [40]. Although the definitive validation of the proposed modeling framework will require a more comprehensive analysis, including several comparisons between CAD and built geometry, for this dissertation, four complementary evaluation methods were applied:

1. **Evaluation with manufactured parts:** A large set of press-broken and stamped manufactured steel parts for the QuadPod were fabricated using the methodology of the proposed approach, then sent directly to the site without a preliminary fit-up in the factory. Some components, which were thought to be well-designed were inadvertently left out of the case-study, that is, they were designed using conventional expert judgement but not treated to the system-based allocation of GD&T. These components of the structure, which were not included in the analysis, created minor installation conflicts. on site – but these were able to be accommodated due to appropriate tolerances in the other parts . The assembly was determined to be “easily buildable” by the installation crew. Also, the assembly

crew stated that the specified clearances were such that there was no need to use mallets or other typical field-adjusting techniques.

2. **Evaluation with validation values built to assess the compliance of the geometry with domain-specific manufacturing, tolerances, and clearances:** In addition to verification of dimensional (numerical) constraints, the validation value stereotypes created for this implementation acted as a critical component of the knowledge verification stage. By means of nested Booleans (true/false), these value properties were able to traverse an entire integrated CAD-SysML model and verify that performance and target values met the intended specification.
3. **Evaluation comparing parameters before and after parametric executions:** The before and after value properties obtained from the CAD model were compared in design tables included in in Chapter 7. Using the updated values, and after CAD modifications were committed to the model, interference analyses were completed (see particularly the Wall Assembly case study in Chapter 7). Although, an interference analysis might produce incidental benefits that improve several issues due to modeling mistakes, this is not the focus of this dissertation. However, it was obvious after the exercise that by combining geometry and knowledge, several conflicting tolerances and clearances issues can be avoided. Identifying the clash is not helpful enough. We also need to understand why it was produced, and how to fix it. Furthermore, in the Wall Assembly case study, most of the interferences were automatically fixed after the manufacturing compliance analysis.

4. **Evaluation by comparing the proposed methodology with another approach of modeling geometric variability for construction:** The wall assembly case study had been previously developed using a "virtual as-built" modeling approach. This previous approach randomly selected a parameter value from five different values coming from typical tolerances calculations (RSS, WC, LLt, ULt, nominal)[16]. The previous approach failed to meet several of the assembly constraints due to a lack of topological consistency among the randomly selected values, for each critical dimension, after the parameters were altered. The new approach has proven to keep the same internal CAD topological consistency before and after modifications have been done on the critical dimensions.

After all these complementary systems evaluations have been performed the following sections address the results in three relevant categories:

1. Positive aspects of the implementation,
2. Manufacturing data obtained from the case study analyses,
3. Unanticipated issues and resolutions of Compliance Analysis implementation, and
4. Potential for Future Advancement of the Implementation.

---

[16] RSS: Root Square Sum. WC: Worst Case scenario. LLt: Lower Limit of tolerance. ULt: Upper Limit of tolerance. Nominal: Nominal geometry.

8.2. Positive Aspects of the Implementation

As discussed in this dissertation, one of the central problems of a tolerances modeling and allocation framework for the building industry is that rules and values of tolerances specifications are not provided in a geometric-specific context. Hence, for this implementation, tolerances allocation is performed as a factor of the critical dimension to be evaluated (case-based tolerances allocation). Another important matter of a multidisciplinary domain such as building construction is that tolerances and clearances data are either missing or fragmented during the design process. Therefore, for this implementation, it has been necessary to create data continuity by defining special modeling elements that can be programmatically concatenated in a single set of procedures that requires little user input, In other words, the parametric model of the nominal geometry are created with the *a priori* knowledge that the manufacturing compliance analysis would update the geometry, and the model had to be built with this in mind. This approach is meant to ensure that text-based requirements can be automatically mapped onto geometric features. And so at a general level, the main practical functionalities of the proposed application are:

- Adding knowledge-compliant, feature-oriented, case-based tolerances and clearances to the CAD model;
- Automatically assessing manufacturability of parts and assemblies to identify possible fabrication conflicts;
- Upgrading “nominal geometry” by adding feature-oriented considerations based on material-system-specific engineering and manufacturing knowledge; and

- Evaluating and validating tolerances and clearances specified for parts and assemblies.

In the evaluation stage in Chapter 7, numerous SysML diagrams have been created for each case study. They represent knowledge, structure, and behavior. These diagrams are not only graphic representations of an underlying object-oriented programming language, but also an intuitive way to combine geometric and non-geometric information in a unified language. This is one of the strengths of this implementation. Because knowledge is stored in encapsulated elements or blocks, they are easy to apply and combine with other elements stereotypes through SysML associations. Furthermore, this evaluation demonstrates the application of these reusable blocks of manufacturing knowledge to assess geometric variability and tolerances allocation. These blocks, described as <<Design Specification>> or <<Manufacturing Specification>> in the NXProfile, contain the rationale of specific tolerances or manufacturing rules and are automatically enforced through binding associations to specific CAD features. The results of design and manufacturing specifications analysis have been successfully converted in one of the following value types:

- Target values that are calculated values taken from the CAD structure and come back to the CAD structure as updated values;
- Validation values that are Booleans that verify that a specific rule has been met with a “true” or “false” statement; and
- Performance values that are computed real values that do not belong to a any specific CAD feature, but are used to verify some dimensional constraint or to

inform some part of the process or create shop drawings (for example, RSS or WC values).

8.3. Manufacturing data obtained from the case study analyses

The following list describes the data available to the user after performing the manufacturing compliance evaluation proposed in this framework:

- Tolerances and clearances specifications for each evaluated feature/component/assembly;
- Rationale regarding tolerances, clearances, and manufacturability through text-based requirements and specifications, which can be easily accesses from the system model;
- Mathematical expressions that assess geometry through manufacturing knowledge;
- Performance assessments that verify in real-time the quality of clearances of an assembly; and
- Automatically suggested manufacturing specifications based on assigned materials of the CAD model (this information is displayed in specifications allocation matrices).

8.4. Unanticipated issues and resolutions of Compliance Analysis implementation

A Siemens NX assembly has the same .prt file extension of a single part. However, in an assembly, the files (.prt) contains other files (.prt) as children, which are the components of the assembly. Based on the original approach of creating an empty

folder for each <<NXPart>> element to store updated instances from parametric execution, an error was identified. The reason for this error was that adding SysML packages[17] to blocks defies the topological internal structure of the SysML language. This problem was fixed by creating instances folders at the top level of any CAD import execution.

Also, during early versions of the implementation, the application understood intentional filtering of stereotypes as "inconsistencies." Rather, filtering stereotypes is intended to reduce the complexity of the model to keep only what is relevant for a specific "view," but still maintaining the topological structure of the model. Furthermore, when a feature name was changed in SysML and the consistency analysis was executed, the application loses its mapping ability. This means, the one-to-one comparison breaks at the renamed feature and for this reason, everything after the break is understood as an inconsistency. This problem was also fixed by using underlying references to the elements without considering their current name. Rather, for this we updated the process by assigning a GUID that belongs to every CAD element imported in SysML.

---

[17] A package is the model element that correspond to a "folder," which is used to store any kind of model elements and diagrams in SysML.

8.5. Current Challenges of the Implementation

Parametric modeling is largely used in the design and manufacture of parts and assemblies for engineering and construction. However, the development of highly complex models of parts and assemblies can lead to failures in the automation of processes or in the implementation of constraints. This procedure requires from the designer/modeler a deep understanding about how to specify fully constrained models to make meaningful analyses using the proposed tool. When multiple sets of parts and object-behaviors are considered, the complexity of parameters can lead to performance degradation or the propagation of mistakes.

The tolerances evaluation on assemblies is not a typical practice in parametric modeling – and the literature contains not enough guidelines on how to implement GD&T within a parametric modeling environment. Certain common features within parametric solids models do not work well with compliance analysis developed here. For example, features are often generated as patterns. The location of these patterned features cannot be applied separately for each instance of the pattern as a single parameter manages all of the instances. Although, it is possible to create pattern tolerances relations within the SysML model, bringing variability of pattern parameters back to the CAD model remains a challenge. For the purpose of this dissertation, every component has been modeled as separate entity.

Another important consideration to improve the proposed framework is naming semantics. At this point of the implementation, obtaining meaningful analyses requires designers to be consistent in the way they name features and parameters. This situation occurs because the user is responsible for accurately connecting constraints parameters to

CAD parameters by means of binding connectors. A more intuitive (or automated) way to make such connection would enhance the robustness of the system, while reducing modeling time.

8.6. The Role of the System Architect

Similar to the functions of a BIM manager, which deal specifically with IT interoperability, the proposed framework requires an actor that assumes all the responsibilities for the creation of SysML integration profiles, and proper knowledge allocation during coordination of trades and stakeholders. Also, besides modeling integration, another important functions of the systems architect is knowledge integration. This task involves the creation and maintenance of design specifications and standards based on data collected from the different stakeholders of the project. With this domain-specific knowledge, another important task arises: automation of all routines that will ensure that the required knowledge is properly applied during design and analysis. This task includes knowledge allocation to the imported models, and the creations of reusable analysis contexts (model views) that will assess the compliance of the building geometry. For the systems architect, a proper coordination with all stakeholders is critical, especially at the beginning of the project when all data needs to be collected and properly conveyed from every actor and material system. After all models have been imported/linked the systems architect will guarantee that knowledge verification (e.g. SDCT and HCT analyses) is executed right on time to meet project schedule of to discuss corrective actions with the trade-specific design teams. The following SysML activity Diagram depicts an example of project coordination centralized in the system architect roles to explain how this new actor fits with current activities of the BIM workflow.

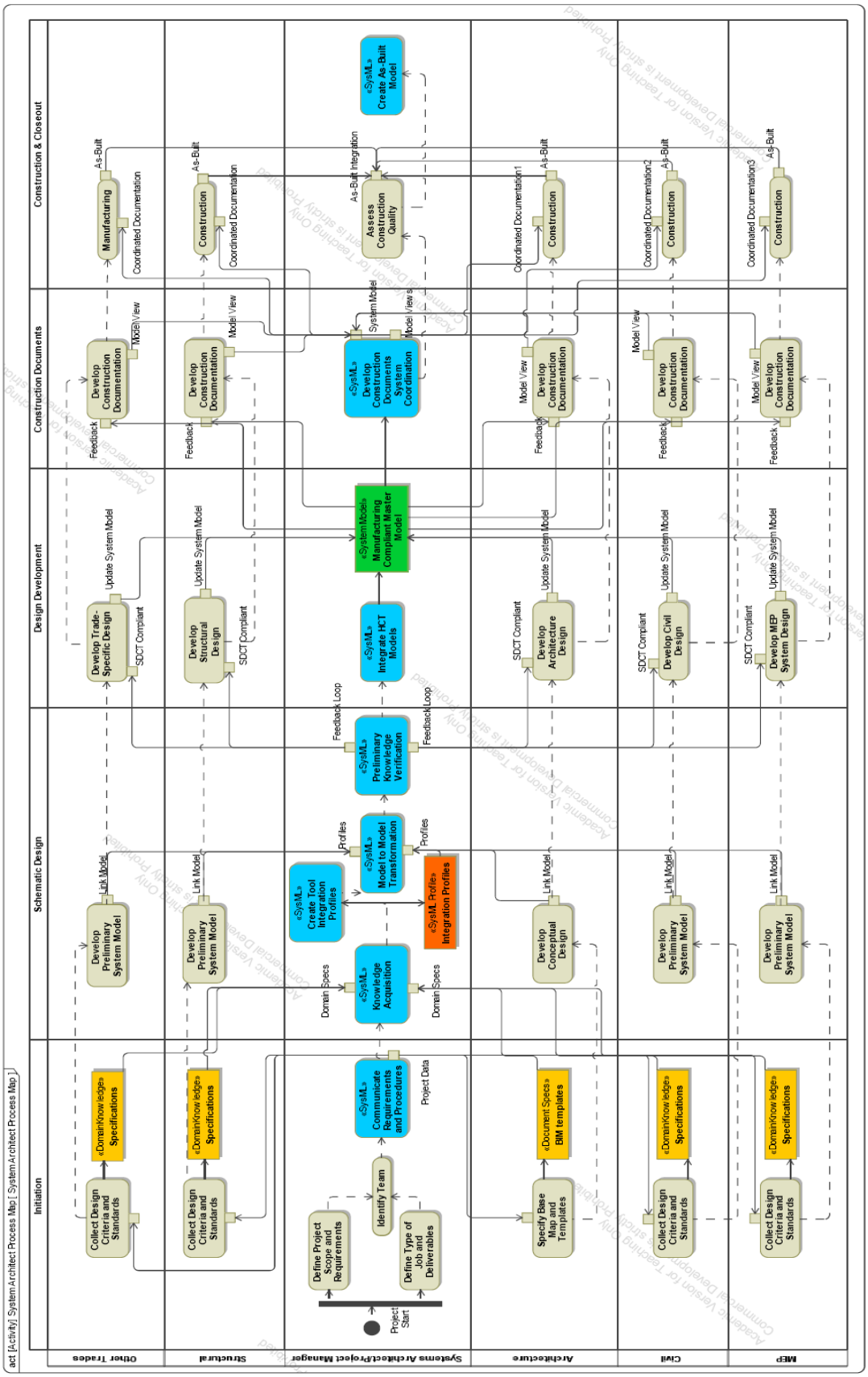


Figure 188, System Engineer Process Map

CHAPTER 9: Conclusions

This dissertation has developed a feature-based, knowledge-aided modeling framework that integrates a parametric CAD tool with a system modeling platform to assess geometric variability and manufacturability in building construction. Furthermore, the work facilitates the representation of material-specific knowledge across different material systems to help designers identify conflicting manufacturing requirements and specifications This framework provides high-level descriptions of tolerances requirements and manufacturing specifications on the system model side, which becomes a low-level description of feature-based (geometric) tolerances allocation on the CAD side. Complex tolerances calculations are performed by a mathematical engine and then automatically allocated in the CAD model. In addition to the computational implementation that demonstrates the proposed compliance analysis framework, this dissertation provides a comprehensive review of tolerances for building design, engineering and construction that clarifies the methods of GD&T, and identifies those methods that best suited for model-based integration. The dissertation also provides a review of model-based systems engineering and its associated modeling language, SysML.

This project was motivated by the obvious fact that nominal geometry is not achievable during construction stages, a fact that is exacerbated by the fact that designers using CAD/BIM tools have few tools by which to virtually assess the build-ability of their designs. This condition leads to drawbacks affecting cost, schedule, and quality of buildings. This situation occurs due to the multi-party nature of the construction lifecycle, the highly stochastic outcomes from construction processes, and the heterogeneousness

and complexity of building assemblies. From a computational perspective, the lack of common ground between design and construction stages when developing design specifications, the lack of manufacturing-specific knowledge available for designers in the early stages of design, the lack of manufacturing compliance and verification methods for BIM models, and the lack of multidisciplinary consistency among tools and stakeholders are also identified as main causes of inaccuracies in construction. This dissertations argues that the aforementioned problems can best be addressed in the context of an integrated knowledge-modeling platform. The thesis was demonstrated through the development a model-centric architecture that enabled the integration of dissimilar domain-specific tools in a single platform and modeling language, SysML. The developed implementation focused on model integration and model consistency among System Engineering models, mathematical engines, and BIM (CAD) models.

9.1. Research Questions and Hypothesis

- *RQ1: It is possible to represent and store machine-readable manufacturing knowledge to parametrically assess manufacturability and tolerances of CAD geometry in the early stages of building design?*

Yes. Most of the technical knowledge about material systems, such as tolerances and clearances, material standards, building codes, and other documents that contain manufacturing know-how describe manufacturing and design rules as a mathematical expression, logical expression, or simple numeric values (dimensions). This condition is critical to establish machine-readable protocols for three reasons:

1. Mathematical and logical expressions are highly interoperable, which means they are easily transferred between geometric and non-geometric tools through standard programming protocols.
2. The storing of mathematical or logical expressions, which can be adapted to formally represent a piece of manufacturing knowledge, is the central characteristic of the <<constraints>> SysML modeling element. The <<constraints>> stereotype also allows straightforward connectivity with text-based requirements and specification by means of dependency associations. This condition enables the coexistence of a text-based rationale (Requirements or specifications) and its mathematically-described assessment tool (Constraint blocks).
3. The depiction of a point set, as a CAD shape, by a single real-valued function of point coordinates is a traditional problem of analytical geometry that is based on mathematical, logical, and topological expressions [149]. Furthermore, in parametric solid modeling, mathematics allows the independent handling of each CAD feature parameter. Through the feature-based structural decomposition approach developed in this dissertation, CAD geometry has been successfully translated from a geometry-based environment (in Siemens NX) to a non-geometry based environment. In the latter, the combination of the numeric depiction of parametric CAD geometry with the <<constraints>> elements has allowed the integration of knowledge-based and geometric features, which has been the main focus of this project.

- *RQ2: It is possible to develop a computationally integrated modeling framework among Model Based Systems Engineering models, mathematical engines, and BIM (CAD) models?*

Yes, the high level of generality of the SysML language, which enables the formal representation of the data structure of almost any engineering tool by means of SysML profiles or Domain Specific Languages (DSL) has allowed the computational integration of MagicDraw SysML, Siemens NX, and Maplesoft Maple. This integration has been developed through a JAVA application that accesses the API modules of each of the integrated tools. Although the integration of MagicDraw SysML with Maple has recently been included in Maple 18 by[18] Maplesoft, at the start of this dissertation, there was very little work done on the integration of SysML with NX or any other CAD package. Thus, the proposed SysML-CAD integration framework is an important contribution to the engineering domain as it allows better geometric-centered analysis and optimization.

- *RQ3: Can we predict conflicting tolerances interactions among different material systems from different vendors before creating building assemblies on the site?*

Although the results described here are positive, to fully answer this question, more testing must be done. So far, four different evaluation procedures have been

[18] Part of the Maple-Siemens NX integration work developed by the author of this dissertation has been included in the version 18 of Maple, which is commercially available.

executed (see Table 19, below). The results of each evaluations supports the conclusion that the framework is an effective tool to predict conflicting tolerances interaction. Definitive case studies that tracked fabrication of similar (or identical) building systems, composite of several multi-material building assemblies, and the validated against their built instances, would be the ultimate proof.

From the case studies presented here, it is clear that the developed implementation has identified and automatically corrected several manufacturing knowledge-related interferences of the multi-material, nominally-designed wall assembly. In addition, the QuadPod case study considered several processes of the structural steel and sheet metal domains such as metal forming, metal cutting, assembly clearances and fastening, galvanizing, and others. Although steel sheet metal and structural steel are both of the same underlying material they are use fabrication procedures, which adds a level of heterogeneity to the case study.

Table 19: Validation procedures

N	Case Study	Validated against manufactured parts	Validated against buit-in values	Validated by comparing design tables before/after analysis	Validated by comparing analyses of two different approaches
1	Cylindrical Fit (2 material system)				
2	QuadPod Assembly (1 material system)				
3	Wall Assembly (4 material systems)				

Then, restating and separating this dissertation hypothesis on its constituting parts we can confirm:

The seamless integration of parametric CAD geometry with a system-level modeling environment (a) allows the feature-based allocation of manufacturing specifications (b), based on material-specific knowledge and processes constraints (c),

and also identifies complex conflicting interactions of tolerances (d) across multi-material building assemblies(e).

9.2. Contributions

This dissertation contributed to the BIM and architectural design fields by developing and implementing a knowledge-based modeling framework to assess manufacturability and identify/prevent negative tolerances interaction in the manufacturing and assembly of building components. In addition, the dissertation contributed to the systems engineering and computer science fields by demonstrating a system modeling platform, integrated into a CAD application, through a novel model-to-model transformation approach.

9.2.1. Expected research contributions of the present dissertation

Innovative model-to-model transformation methodology:

This general contribution for construction and engineering aims for the development of a structural, feature-based decomposition approach of parametric CAD models into System Models. This method programmatically integrates two different data structures (but could be any number) by recreating the meta-model of the CAD application through a graph-based representation in SysML. This machine readable/executable framework links feature-based geometry to manufacturing know-how through knowledge-based mathematical and logical constraints. Although many project have been completed to integrate engineering analysis with SysML, none of them has successfully integrated feature-based parametric CAD geometry.

Domain expert advice about manufacturing and assembly processes:

This contribution to the construction field enables the programmed allocation of material-specific knowledge for components and assemblies based on geometric context and material type. The integration of specific features of geometric data with a system modeling tool allows rule-based design and solve operations that otherwise require extensive manual data interpretation and translation. Furthermore, linking features (geometry and function) and manufacturing knowledge in the same interface allows designers to develop a better understanding of the impact of their design actions. This platform encourages the designer to link CAD features to <<requirements>>, <<design specifications>>, and <<manufacturing specifications>> to verify different levels of project compliance. The methodology is clearly extensible to domains beyond that of geometric tolerancing and compliance analysis.

One truth, multiple model views:

This contribution to the engineering and construction domains centralizes project requirements, geometry, evaluation, and design specifications in a single integrated modeling environment. Much work in the development of BIM tools that support architectural design, including much of the work at Georgia Tech, has focused on the development of workflows that involve multiple translations of data. But these translations, even when tightly scripted, often suffer from the loss of sematic clarity. The model integration developed in this dissertation provides geometric data to numerous domain-specific tools – in a bi-directional manner.

Finally, it is argued that reinforcing a system-level semantic layer on the BIM environment will facilitate the representation of geometric and non-geometric interactions of a building project. For example, this methodology, based on a central

model (system model), enables the integration of several model views and tolerances analysis from different vendors. This approach can potentially replace the current industry standard that specifies tolerances allocation as a separate task for each material system.

Model consistency method:

The implementation of an on-demand model-to-model and tool-to-tool consistency assessment and model data update will hopefully lead to better model consistency, and at least the automated identification of inconsistencies between different abstractions of the model. This feature in the present framework has been established due to the fact that many of the negative interactions among building components are caused by data fragmentation and inconsistencies among different model views and tools.

9.3. Recommendations for Future Research and Development

Although, important contributions of this dissertation can be recognized from the previous section 9.2.1., there is still a long way to go to achieve a fully automated integration of knowledge and geometry.

Automatic allocation of variational data on shop drawings is an opportunity for further development. In this dissertation, all manufacturing-compliant parameters that belong to the CAD geometry are seamlessly updated from SysML. This is, the outcome of the evaluation is a "manufacturing-compliant, design-compliant nominal geometry." However, the complete variational data of tolerances and clearances values such as WC, RSS, centered dimensions, LL, UL, plus/minus, are stored in the SysML tables obtained from the analyses results. Although this approach greatly helps to convey variational data more efficiently than the current state of the art, the ability to automatically add

variational data within shop drawings would produce a greater level of automation and model consistency.

Also, this dissertation has seamlessly integrated one CAD (BIM) application into a system model environment. Furthermore, this dissertation has proven that this kind of integrations is possible by translating the meta-model (data structure) of the CAD application into a DSL (profile) in SysML. However, if more BIM tools need to be integrated, it will require a specific DSL and implementation code for each of them. Although, this approach is feasible and replicable, it would be more effective to create a single application/DSL that can translate any CAD package into a system model environment. Likewise, this project did not deal with interoperability in the sense of creating neutral files to go from one CAD package to another. Rather, what this project proposes is the simultaneous integration of disparate modeling environments without the need of an exchange file. Still, I envision an nonproprietary integration approach that can incorporate any CAD by means of a "standard integration method," which would probably use the IFC[19] or STEP descriptive methods as foundation for this translation.

Finally, domain-specific feature-based modeling environments, as sheet metal of Siemens NX, are critical for a knowledge-based integration. It is unlikely that

---

[19] The Industry Foundation Classes (IFC) data model describe building and construction industry data. It is a platform neutral, open file format specification that is not controlled by a single vendor or group of vendors.

manufacturing knowledge will refer to a virtual geometric entity (a point or a line) as they are representational, but not existent in the real world. Rather, manufacturing knowledge refers to features, which are a set of virtual modeling elements that create design intent on their topological aggregation. Unfortunately, feature-based design tools have only created domain-specific modeling environment for few material systems, mostly for aerospace or automotive engineering. Then, an important research area would be the integration of several other construction-specific feature based modeling environments in current parametric modeling tools.

9.4. Concluding Remarks

This dissertation has covered several aspects of the building construction domain. The understanding and prevention of construction variability is a complex matter that involves not just geometry, but also a deep familiarity with each of the building component manufacturing and assembly processes. For each material system, there are people that develop and advance this knowledge throughout their entire lives – and the expert knowledge of these actors are embodied, but not completely nor consistently, in design and construction specifications. This fact makes the design of building parts and the assembly of these parts scenario even more complex. We can highlight this complexity by imagining the lifecycle of a specific feature on a specific part – and ask a series of questions.

- How is a single feature created?
- What are the processes from which this feature can be built?
- Based on different manufacturing processes available, how different will be our feature when finally built?

- Which are other features will the feature be linked to?
- How do we know if our feature is meeting a design requirement or a specification requirement?
- How do we know what will happen to an assembly if our feature is changed? Ultimately, how do we know if our feature will be what we want our feature to be?

To answer these questions requires an immense amount of knowledge in countless fields, and yet, based on the unsolvable modeling uncertainty that motivated this research, we will never get an absolute answer. And so, considering this scenario, the goal of this dissertation has **not** been to gather all construction knowledge and make it automatically available, as this would be an endless endeavor. Rather, the goal of this dissertation has been to create a platform for this knowledge to be gathered, stored and applied by means of innovative computational workflows, which are both coordinated and consistent.

The cheapest construction inaccuracy to fix is the one that you prevented. The sole success of this goal has required advancements in different aspects of BIM and the systems engineering domain. The specific focus of this project, manufacturing compliance and construction tolerances, have been surveyed from different perspectives so that the meta-requirements for modeling the knowledge are understood. There are, however, still many aspects of building semantics, behaviors, and workflows that have not been considered in this work. In the future, a complete representation of the building as a whole system will capture all functional and behavioral interactions that occur across different domains and stages of the building lifecycle. This ideal scenario will result in

not just tolerances attributable to fabrication accuracy but also behavioral considerations that affect their variation. With proper development, the framework proposed by this dissertation could create a new kind of building design paradigm: A modeling environment that virtually and simultaneously brings to the table all domain experts, anytime that building feature is created.

APPENDIX 1: Implementation Code:

Controller:

ContaintmentTreeContextPopConfigurator:

```
package gov.nasa.jpl.imce.sysmlnxsync.controller;

import
gov.nasa.jpl.imce.sysmlnxsync.actions.ImportNXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.ImportNXPartWithFilte
r;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.InternalUpdate;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.InternalValidate;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.LinkNXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.ResolveNXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.UpdateFromNXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.UpdateToNXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.actions.ValidateAgainstNXPart
;

import java.util.Collection;

import com.nomagic.actions.ActionsManager;
import
com.nomagic.magicdraw.actions.BrowserContextAMConfigurator;
import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import com.nomagic.magicdraw.ui.browser.Tree;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpe
cification;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

     /**
      * Configuration of the pop up menu that appears when
right clicking in the containment browser
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */
     public class ContainmentTreeContextPopConfigurator
implements BrowserContextAMConfigurator {

          private static final boolean
isPackageWithInstanceChildren(Object object) {
               if (!(object instanceof Package)) {
                    return false;
               }
               Package userPackage = (Package)object;

               // Check if it has any children which are
properties
               Collection<Element> children =
userPackage.getOwnedElement();
               for (Element child : children) {
                    if (child instanceof
InstanceSpecification) {
                         return true;
                    }
               }
               return false;
          }

          /**
           * Configure the containment browser context menu
- you can extend this by adding more classes that
           * extend class MDAction
           */
          @Override
          public void configure(ActionsManager
actionsManager, Tree tree) {
               // You may want to do some checks here to
see which element is currently selected
               // E.g. if you want to check whether a
package was selected, you can write something like
```

```
                // if(tree.getSelectedElement() instance of
Package) { ...
                Object userObject =
tree.getSelectedNode().getUserObject();


     Application.getInstance().getGUILog().log("userObject
" + userObject );

                Project project =
Application.getInstance().getProject();
                if (userObject instanceof Package) {

     actionsManager.getLastActionsCategory().addAction(new
ImportNXPart());

     actionsManager.getLastActionsCategory().addAction(new
ImportNXPartWithFilter());
                    if
(isPackageWithInstanceChildren(userObject)) {

     actionsManager.getLastActionsCategory().addAction(new
InternalUpdate());

     actionsManager.getLastActionsCategory().addAction(new
InternalValidate());
                    }

                } else if (userObject instanceof Class) {
                     Class userClass = (Class)userObject;
                     Stereotype nxPartStereotype =
StereotypesHelper.getStereotype( project, "NXPart" );

                    if
(StereotypesHelper.hasStereotype(userClass,
nxPartStereotype)) {

     actionsManager.getLastActionsCategory().addAction(new
UpdateToNXPart());

     actionsManager.getLastActionsCategory().addAction(new
UpdateFromNXPart());

     actionsManager.getLastActionsCategory().addAction(new
LinkNXPart());
```

```
     actionsManager.getLastActionsCategory().addAction(new
ValidateAgainstNXPart());

     actionsManager.getLastActionsCategory().addAction(new
ResolveNXPart());
                    }
               }
          }

          @Override
          public int getPriority() {
               return 0;
          }

     }
```

PluginMain:

```
package gov.nasa.jpl.imce.sysmlnxsync.controller;

import
com.nomagic.magicdraw.actions.ActionsConfiguratorsManager;
import com.nomagic.magicdraw.core.Project;
import com.nomagic.magicdraw.plugins.Plugin;
import
com.nomagic.magicdraw.plugins.ResourceDependentPlugin;

/**
 * Main entry point into the plugin
 *
 * @author francisco.valdes@jpl.nasa.gov,
 */
public class PluginMain extends Plugin implements
ResourceDependentPlugin {
     public static final boolean DEBUG = true;

     /**
      * Perform any potentially necessary cleanup when
the plugin is unloaded
      */
     @Override
     public boolean close() {
          return true;
     }

     @Override
     public String getPluginName() {
          return "SysMLNXSync";
     }

     @Override
     public String getPluginVersion() {
          return "1.0";
     }

     /**
      * This function is called after isSupported()
has been called. Any initialization
      * should be done at this point
      */
     @Override
     public void init() {
          // Smoke test: show a message dialog
```

```
     ActionsConfiguratorsManager.getInstance().addContainme
ntBrowserContextConfigurator(
                    new
ContainmentTreeContextPopConfigurator()
               );
          }

          @Override
          public boolean isPluginRequired(Project p) {
               return false;
          }

          /**
           * isSupported allows for pre-loading checks to
be performed, i.e. one could check at
           * this point whether NX is installed and return
false if that is not the case
           */
          @Override
          public boolean isSupported() {
               return true;
          }

     }
```

NXConnection:

MapleNXEngine:

```
package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collection;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.w3c.dom.Document;
import org.w3c.dom.NodeList;
import org.xml.sax.SAXException;

import com.maplesoft.externalcall.MapleException;
import com.maplesoft.openmaple.Algebraic;
import com.maplesoft.openmaple.Engine;
import com.maplesoft.openmaple.EngineCallBacksDefault;
import com.maplesoft.openmaple.MString;
import com.sun.xml.bind.StringInputStream;

/**
 * Implementation of NX connection using open maple
interface
 *
 * @author francisco.valdes@jpl.nasa.gov,
 */

public class MapleNXEngine implements NXEngine {
     private static Engine engine;

     private static void initializeMaple() {

          // Create a new Maple Engine object
          try {
               if (engine == null) {
                    String[] mapleEngineArgs = new
String[0];
                    //mapleEngineArgs[0] = "java";
                    engine = new
Engine(mapleEngineArgs, new EngineCallBacksDefault(), null,
null);
```

```
                    }
               }
               catch (MapleException e) {
                    // TODO Auto-generated catch block
                    e.printStackTrace();
               }
          }


          public MapleNXEngine() throws
NXConnectionException {
               try {
                    initializeMaple();
                    engine.restart();
                    String cmd = "CAD:-NX:-
OpenConnection():";
                    engine.evaluate( cmd );
                    engine.evaluate( "[foo], [bar];" );
               } catch (MapleException me) {
                    throw new NXConnectionException();
               }
          }

          @Override
          public void closeConnection() {
               try {
                    engine.evaluate("CAD:-NX:-
CloseConnection();");
                    engine.evaluate( "[foo], [bar];" );

               } catch (MapleException me) {
                    me.printStackTrace();
               }
          }

          @Override
          public boolean closePart(NXPart part) {
               // TODO Auto-generated method stub
               return false;
          }

          @Override
          public Collection<String> getComponentList(NXPart
part) {
               // TODO Auto-generated method stub
               return null;
          }
```

```
          @Override
          public Collection<NXExpression>
getExpressions(NXPart part) {
               // TODO Auto-generated method stub
               return null;
          }

          @Override
          public Collection<NXFeature> getFeatures(NXPart
part) {
               // TODO Auto-generated method stub
               return null;
          }

          @Override
          public Collection<NXExpression>
getParameterInfo(NXPart part) {
               String filename = part.getPath();
               try {
                    String mathml =
((MString)engine.evaluate("MathML:-
ExportPresentation([seq([x,CAD:-NX:-
GetParameterValue(x,form=\"NX\")], x in CAD:-NX:-
GetParameterNames(\"" + filename.replace("\\", "\\\\") +
"\")) ]);")).stringValue();
                    Collection<NXExpression> params =
parseMathML( mathml );
                    return params;
               } catch (MapleException me) {
                    me.printStackTrace();
               }
               return null;
          }

          @Override
          public String getUniqueIdentifier(NXPart part) {
               Algebraic result;
               String filename = part.getPath();
               try {
                    result =
engine.evaluate("GetPartUID(\"" + filename.replace("\\",
"\\\\") + "\");");
                    return result.toString();
               } catch (MapleException me) {
                    me.printStackTrace();
```

```
            }
            return null;
        }

        @Override
        public boolean isConnected() {
            // TODO Auto-generated method stub
            return false;
        }

        @Override
        public NXPart openPart(File file) {
            return openPart(file, false);
        }

        @Override
        public NXPart openPart(File file, boolean
recurse) {
            NXPart result = null;
            System.out.println("opening part " + file);
            String filename = file.getAbsolutePath();
            try {
                engine.evaluate("CAD:-NX:-OpenPart(\""
+ filename.replace("\\", "\\\\") + "\"):");
                result = new NXPart( filename, filename
);
            } catch (MapleException me) {
                me.printStackTrace();
            }
            return result;
        }

        private Collection<NXExpression> parseMathML(
String mathml ) {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = null;
            try {
                db = dbf.newDocumentBuilder();
            } catch (ParserConfigurationException e1) {
                // TODO Auto-generated catch block
                e1.printStackTrace();
            }
            Document dom = null;
            try {
                dom = db.parse( new StringInputStream(
mathml ) );
```

```
                } catch (SAXException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                } catch (IOException e) {
                     // TODO Auto-generated catch block
                     e.printStackTrace();
                }

                org.w3c.dom.Element docElement =
dom.getDocumentElement();
                org.w3c.dom.Node rootNode =
docElement.getFirstChild();
                NodeList nl = rootNode.getChildNodes();
                if (nl == null) {
                     return new ArrayList<NXExpression>();
                }
                int parLen = 0;

                org.w3c.dom.Node  child;

                parLen = nl.getLength();
                String key, val;

                Collection<NXExpression> params = new
ArrayList<NXExpression>();
                for (int i=0; i < parLen; i++) {
                     org.w3c.dom.Node pair = nl.item(i);
                     child = pair.getFirstChild();
                     key = child.getTextContent();
                     val =
child.getNextSibling().getTextContent();
                     params.add( new NXExpression(key, val,
null) );
                }
                return params;
           }

           @Override
           public boolean renameFeature(NXPart part, String
oldName, String newName) {
                // TODO Auto-generated method stub
                return false;
           }

           @Override
           public boolean renameParameter(NXPart part,
String oldName, String newName) {
```

```
                // TODO Auto-generated method stub
                return false;
           }

           @Override
           public boolean savePart(NXPart file) {
                // TODO Auto-generated method stub
                return false;
           }

           @Override
           public boolean setParameterInfo(NXPart part,
Collection<NXExpression> params) {
                String name;
                try {
                     Algebraic result = null;
                     for (NXExpression param : params) {
                          name = param.getName();
                          result = engine.evaluate("CAD:-
NX:-SetParameterValue(\"" + name + "\", \"" +
param.getValue() + "\"):" );
                          //engine.evaluate( "[foo], [bar];"
);
                          //System.out.println("CAD:-NX:-
SetParameterValue(\"" + paramNames.get(i) + "\", \""
+paramValues.get(i) + "\"):" );
                     }
                     return true;
                } catch (MapleException me) {
                     me.printStackTrace();
                }
                return false;
           }

           @Override
           public boolean setParameterValue(NXPart part,
String param, String value) {
                // TODO Auto-generated method stub
                return false;
           }

           @Override
           public boolean setWorkPart(NXPart part) {
                // TODO Auto-generated method stub
                return false;
           }
     }
```

NXClientEngine:

```
     package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

     import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;

     import java.io.BufferedReader;
     import java.io.File;
     import java.io.IOException;
     import java.io.InputStreamReader;
     import java.io.StringReader;
     import java.util.ArrayList;
     import java.util.Collection;

     import javax.xml.parsers.DocumentBuilder;
     import javax.xml.parsers.DocumentBuilderFactory;
     import javax.xml.parsers.ParserConfigurationException;

     import org.w3c.dom.Document;
     import org.w3c.dom.Element;
     import org.w3c.dom.Node;
     import org.w3c.dom.NodeList;
     import org.xml.sax.InputSource;
     import org.xml.sax.SAXException;

     import com.maplesoft.externalcall.MapleException;
     import com.maplesoft.openmaple.Algebraic;
     import com.maplesoft.openmaple.Engine;
     import com.maplesoft.openmaple.EngineCallBacksDefault;
     import com.maplesoft.openmaple.MString;
     import com.nomagic.magicdraw.core.Application;

     /**
      *Implementation of NX connection using NX client
binary
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */

     public class NXClientEngine implements NXEngine {

         private static String mapleDir;
         private static Engine mapleEngine;
         protected final static boolean TESTINPUT = false;
         static {
```

```
          initializeMaple();
        }

        private static String[] buildCommandArray(String
cmd, String args) {
          String[] cmdArray = new String[4];
            cmdArray[0] =
String.format("%s\\run_managed.exe",
System.getenv("UGII_ROOT_DIR"));
            cmdArray[1] =
String.format("%s\\nxclient.exe", mapleDir);
            cmdArray[2] = cmd;
            cmdArray[3] = args;
            return cmdArray;
        }

        private static void initializeMaple() {
          if (TESTINPUT) return;
            String[] mapleEngineArgs ={"java"};
            Algebraic result;

            // Create a new Maple Engine object
            try {
                if (mapleEngine == null) {
                    mapleEngine = new
Engine(mapleEngineArgs, new EngineCallBacksDefault(), null,
null);
                    mapleEngine.restart();
                    result =
mapleEngine.evaluate("kernelopts(bindir);");
                    if (result instanceof MString) {
                        mapleDir =
((MString)result).stringValue();
                    }
                }
            }
            catch (MapleException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }

        private static final boolean isOkay( String result
) {
            return "okay".equals(result);
        }
```

```
        private static final String quote(String s) {
         return "\"" + s + "\"";
        }

        private boolean _isConnected = false;

        public NXClientEngine() throws
NXConnectionException {
              String result = runNXClient("ping", "");
              boolean connected = result.equals("NX
Okay");
              if (!connected) {
                 throw new NXConnectionException();
              }
         }

        private String buildExpression(Element element) {
            if
("expression".equalsIgnoreCase(element.getTagName())) {
                return
element.getAttribute("description");
            } else {
                return null;
            }
        }

        private NXFeature buildFeature(Element element) {
            if
("feature".equalsIgnoreCase(element.getTagName())) {
                String name =
element.getAttribute("name");
                String type =
element.getAttribute("type");
                String customName;
                if (element.hasAttribute("customname")) {
                   customName =
element.getAttribute("customname");
                    if (customName == "") customName =
null;
                } else {
                   customName = null;
                }
                NXFeature feature = new NXFeature(name,
type, customName);

                NodeList nodeList =
element.getChildNodes();
```

```
                for (int i = 0; i < nodeList.getLength();
i++) {
                    Node node = nodeList.item(i);
                    if (node.getNodeType() ==
Node.ELEMENT_NODE) {
                        Element child = (Element)node;
                        String childName =
child.getTagName();
                        if
("feature".equalsIgnoreCase(childName)) {
                            NXFeature childFeature =
buildFeature(child);

feature.addChild(childFeature);
                        } else if
("expression".equalsIgnoreCase(childName)) {
                            name =
child.getAttribute("name");
                            String val =
child.getAttribute("value");
                            feature.addExpression(name,
val);
                        }
                    }
                }
                return feature;
            }
            return null;
        }

        @Override
        public void closeConnection() {
            _isConnected = false;
        }

        @Override
        public boolean closePart(NXPart part) {
            return isOkay( runNXClient("closepart",
quote(part.getPath())) );
        }

        @Override
        public Collection<String> getComponentList(NXPart
part) {
            String result =
runNXClient("get_component_list", quote(part.getPath()));
            String[] components = result.split(";");
```

```
            ArrayList<String> arr = new
ArrayList<String>();
            String strim;
            for (String s : components) {
                strim = s.trim();
                if (strim.length() > 0) {
                    arr.add(strim);
                }
            }
            return arr;
        }

        @Override
        public Collection<NXExpression>
getExpressions(NXPart part) {
         return getParameterInfo(part);
        }

        @Override
        public Collection<NXFeature> getFeatures(NXPart
part) {
         String result;
         if (TESTINPUT) {
              result = returnTestInput();
         } else {
              result = runNXClient("get_features",
quote(part.getPath()));
         }

            if (PluginMain.DEBUG) {

    Application.getInstance().getGUILog().log("getFeatures
" + part.getPath() );


    Application.getInstance().getGUILog().log("[COMPONENTS
: input " + part.getName());

    Application.getInstance().getGUILog().log("[COMPONENTS
: output " + result);
            }


            Document doc = (result != null ? parseXML(
result ) : null);
            if (doc == null) {
                return null;
```

```
            }
            if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("DOCUMENT
OK");
            }
            NodeList nodeList = doc.getChildNodes();
            if (nodeList.getLength() == 0) {
                return null;
            }
            // The first node will be the part node
            Node partNode = nodeList.item(0);
            if (partNode.getNodeType() !=
Node.ELEMENT_NODE ||
!("part".equalsIgnoreCase(((Element)partNode).getTagName())
)) {
                return null;
            }
            // nodeList is the list of children of the
part node
            nodeList = partNode.getChildNodes();
            ArrayList<NXFeature> features = new
ArrayList<NXFeature>();
            for (int i = 0; i < nodeList.getLength(); i++)
{
                Node node = nodeList.item(i);
                if (node.getNodeType() ==
Node.ELEMENT_NODE) {
                    NXFeature feature = buildFeature(
(Element)node );
                    if (feature != null) {
                        features.add( feature );
                    }
                }
            }
            return features;
        }

        @Override
        public Collection<NXExpression>
getParameterInfo(NXPart part) {
            String result = runNXClient("expressions",
quote(part.getPath()));
            String[] lines = result.split(";");

            Collection<NXExpression> params = new
ArrayList<NXExpression>();
```

```
            String[] parampairs;
            String res, name, val, paramName, paramValue;
            int n;

            for (String line : lines) {
                line = line.replaceFirst("^[\\s]*Record.",
"").replaceFirst(".;[\\s]*$", "");
                if (line.trim().length() > 0) {
                    parampairs = line.split(", ");
                    paramName = paramValue = null;
                    for (String pair : parampairs) {
                        res = pair.replaceFirst(" *([A-Za-
z0-9]*) *= *\"([^\"]*)\"", "$1|$2");
                        //System.out.println( "pair: " +
pair );
                        //System.out.println( "relt: " +
res );
                        n = res.indexOf('|');
                        name = res.substring(0,
res.indexOf('|'));
                        val = res.substring(n+1);
                        if (name.equals("name")) {
                            paramName = val;
                        } else if (name.equals("value")) {
                            paramValue = val;
                        }
                    }
                    params.add( new NXExpression(
paramName, paramValue, null)  );
                }
            }

            return params;
        }

        @Override
        public String getUniqueIdentifier(NXPart part) {
            String result =
runNXClient("get_unique_identifier",
quote(part.getPath()));
            return result;
        }

        @Override
        public boolean isConnected() {
            return _isConnected;
```

```
        }

        private boolean isPartOpen(File file) {
            String result = runNXClient("isopen", quote(
file.getAbsolutePath() ) );
            return result.equals("yes");
        }

        @Override
        public NXPart openPart(File file) {
            return openPart(file, false);
        }

        @Override
        public NXPart openPart(File file, boolean recurse)
{
         NXPart part;
         if (TESTINPUT) {
              part = new NXPart( "testpart.prt",
"testpart.prt" );
                part._features    = getFeatures(part);
                return part;
         }
            boolean result;
            String filename = file.getAbsolutePath();

            if (isPartOpen(file)) {
                part = new NXPart( filename, filename );
                result = setWorkPart(part);
            } else {
                result = isOkay( runNXClient("openpart",
quote(filename)) );
                part = (result ? new NXPart( filename,
filename ) : null);
            }

            if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("open NX
part status: " + part );
            }

            if (part != null) {
                part._components  =
getComponentList(part);
                part._features    = getFeatures(part);
                part._expressions = getExpressions(part);
```

```
                part._uid         =
getUniqueIdentifier(part);
                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("open NX
part status2: " + part );
                }
                if (recurse) {
                    part._openComponents = new
ArrayList<NXPart>();
                    NXPart compPart;
                    for (String comp : part._components) {
                        compPart = openPart( new File(
comp ), true );

part._openComponents.add(compPart);
                    }
                }
            }
            return part;

        }

        private Document parseXML(String xml) {
            DocumentBuilderFactory dbf =
DocumentBuilderFactory.newInstance();
            DocumentBuilder db = null;
            try {
                db = dbf.newDocumentBuilder();
            } catch (ParserConfigurationException pce) {
                return null;
            }
            InputSource inStream = new InputSource();
            inStream.setCharacterStream(new
StringReader(xml));
            Document doc = null;
            try {
                doc = db.parse( inStream );
            } catch (SAXException se) {
            } catch (IOException io) {
            }
            return doc;
        }

        @Override
        public boolean renameFeature(NXPart part, String
oldName, String newName) {
```

```
            setWorkPart(part);
            return isOkay( runNXClient("rename_feature",
quote(oldName) + " " + quote(newName)) );
        }

        @Override
        public boolean renameParameter(NXPart part, String
oldName, String newName) {
            setWorkPart(part);
            return isOkay( runNXClient("rename_parameter",
quote(oldName) + " " + quote(newName)) );
        }

        private String returnTestInput() {
         return "<part><feature name=\"Extrude(0)\"
type=\"EXTRUDE\" tag=\"35850\">" +
                "<expression name=\"p8\" value=\"0\"
units=\"MilliMeter\" type=\"Number\" equation=\"p8=0\"
description=\"(Extrude(0) Start Limit)\"/>" +
                "<expression name=\"p9\" value=\"1.5\"
units=\"MilliMeter\" type=\"Number\" equation=\"p9=1.5\"
description=\"(Extrude(0) End Limit)\"/>" +
                "<feature name=\"SB Convert To Sheet
Metal(1)\" type=\"Convert To Sheetmetal\"
tag=\"45866\"></feature>" +
                "<feature name=\"Datum Coordinate
System(2)\" type=\"DATUM_CSYS\" tag=\"45865\"><feature
name=\"SKETCH_000:Sketch(2)\" type=\"SKETCH\"
tag=\"45863\">" +
                "<feature name=\"SB Bend(2)\" type=\"BEND\"
tag=\"45864\">" +
                "<expression name=\"p12\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p12=Sheet_Metal_Bend_Radius\" description=\"(SB
Bend(2) Bend Radius)\"/>" +
                "<expression name=\"p13\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p13=Sheet_Metal_Relief_Depth\" description=\"(SB
Bend(2) Bend Relief Depth)\"/>" +
                "<expression name=\"p14\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p14=Sheet_Metal_Relief_Width\" description=\"(SB
Bend(2) Bend Relief Width)\"/>" +
                "<expression name=\"p15\" value=\"0.33\"
type=\"Number\" equation=\"p15=Sheet_Metal_Neutral_Factor\"
description=\"(SB Bend(2) Neutral Factor)\"/>" +
```

```
                "<expression name=\"p16\" value=\"90\"
units=\"Degrees\" type=\"Number\" equation=\"p16=90\"
description=\"(SB Bend(2) Bend
Angle)\"/></feature></feature></feature>" +
                "<feature name=\"Datum Coordinate
System(3)\" type=\"DATUM_CSYS\" tag=\"35849\"><feature
name=\"SKETCH_001:Sketch(3)\" type=\"SKETCH\"
tag=\"45862\"><feature name=\"SB Bend(3)\" type=\"BEND\"
tag=\"35853\">" +
                "<expression name=\"p17\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p17=Sheet_Metal_Bend_Radius\" description=\"(SB
Bend(3) Bend Radius)\"/>" +
                "<expression name=\"p18\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p18=Sheet_Metal_Relief_Depth\" description=\"(SB
Bend(3) Bend Relief Depth)\"/>" +
                "<expression name=\"p19\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p19=Sheet_Metal_Relief_Width\" description=\"(SB
Bend(3) Bend Relief Width)\"/>" +
                "<expression name=\"p20\" value=\"0.33\"
type=\"Number\" equation=\"p20=Sheet_Metal_Neutral_Factor\"
description=\"(SB Bend(3) Neutral Factor)\"/>" +
                "<expression name=\"p21\" value=\"90\"
units=\"Degrees\" type=\"Number\" equation=\"p21=90\"
description=\"(SB Bend(3) Bend Angle)\"/>" +
                "<feature name=\"Datum Coordinate
System(4)\" type=\"DATUM_CSYS\" tag=\"35851\"><feature
name=\"SKETCH_002:Sketch(4)\" type=\"SKETCH\"
tag=\"45860\"><feature name=\"SB Bend(4)\" type=\"BEND\"
tag=\"45861\"><expression name=\"p22\" value=\"3\"
units=\"MilliMeter\" type=\"Number\"
equation=\"p22=Sheet_Metal_Bend_Radius\" description=\"(SB
Bend(4) Bend Radius)\"/><expression name=\"p23\"
value=\"3\" units=\"MilliMeter\" type=\"Number\"
equation=\"p23=Sheet_Metal_Relief_Depth\" description=\"(SB
Bend(4) Bend Relief Depth)\"/><expression name=\"p24\"
value=\"3\" units=\"MilliMeter\" type=\"Number\"
equation=\"p24=Sheet_Metal_Relief_Width\" description=\"(SB
Bend(4) Bend Relief Width)\"/><expression name=\"p25\"
value=\"0.33\" type=\"Number\"
equation=\"p25=Sheet_Metal_Neutral_Factor\"
description=\"(SB Bend(4) Neutral Factor)\"/><expression
name=\"p26\" value=\"15\" units=\"Degrees\" type=\"Number\"
equation=\"p26=15\" description=\"(SB Bend(4) Bend
Angle)\"/></feature></feature></feature><feature
```

```
name=\"Datum Coordinate System(5)\" type=\"DATUM_CSYS\"
tag=\"35852\"><feature name=\"SKETCH_003:Sketch(5)\"
type=\"SKETCH\" tag=\"35854\"><feature name=\"Split
Body(6)\" type=\"SPLIT BODY\"
tag=\"45859\"></feature></feature></feature></feature></fea
ture></feature></feature></part>";
         }

         private String runNXClient(String cmd, String
args) {
             Runtime r = Runtime.getRuntime();
             Process p = null;
             String[] cmdArray = buildCommandArray(cmd,
args);

             if (PluginMain.DEBUG) {
               Application.getInstance().getGUILog().log(
"PluginMain.DEBUG: " + cmdArray );
             }
             if (TESTINPUT) {
               return "okay";
             }
             try {
                 p = r.exec(cmdArray);
             } catch (IOException e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
             }
             StringBuilder sb = new StringBuilder();

             BufferedReader br = new BufferedReader( new
InputStreamReader (  p.getInputStream() ) );

             String line;
             try {
                 while (((line = br.readLine()) != null)) {
                     //System.out.println( line );
                     sb.append(line);
                 }
             } catch (IOException e) {
                 // TODO Auto-generated catch block
                 e.printStackTrace();
             }


             try {
                 if (p.waitFor() != 0) {
```

```
                    System.out.println("problem");
                }
            } catch (InterruptedException ie) {
                // TODO Auto-generated catch block
                ie.printStackTrace();
            }
            if (PluginMain.DEBUG) {
              Application.getInstance().getGUILog().log(
"PluginMain.DEBUG: " + sb.toString() );
            }
            return sb.toString();
        }

        @Override
        public boolean savePart(NXPart part) {
            String result = runNXClient("save",
quote(part.getPath()));
            System.out.println("Result of saving " +
part.getName() + ": " + result);
            return isOkay( result );
        }

        @Override
        public boolean setParameterInfo(NXPart part,
Collection<NXExpression> params) {
            String value;
            String result = null;
            String name;
            boolean overallSuccess = true;
            for (NXExpression expr : params) {
                name = expr.getName();
                value = expr.getValue();
                //currentName =
SysMLParameters.getSynchronizedParameterName(params, name);
                //if (!name.equals(currentName)) {
                //  boolean success = renameParameter(
part, currentName, name );
                //  overallSuccess = overallSuccess &&
success;
                //  if (success) {
                //
SysMLParameters.setSynchronizedParameterName(params, name);
                //  }
                //}
                result = runNXClient(
                    "set_parameter_value",
```

```
                    String.format("%s %s", quote(name),
quote(value))
                );
                overallSuccess = overallSuccess && (result
!= null && isOkay( result ));
            }
            return overallSuccess;
        }

        @Override
        public boolean setParameterValue(NXPart part,
String param, String value) {
            setWorkPart(part);
            return isOkay(
runNXClient("set_parameter_value", quote(param) + " " +
quote(value)) );
        }

        @Override
        public boolean setWorkPart(NXPart part) {
            return isOkay( runNXClient("setwork", quote(
part.getPath() ) ) );
        }

    }
```

NXConnectionException:

```
package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

public class NXConnectionException extends Exception {

}
```

NXEngine:

```
package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

import java.io.File;
import java.util.Collection;

/**
 * NX connection interface
 *
 * @author francisco.valdes@jpl.nasa.gov,
 */

public interface NXEngine {

     public void closeConnection();

     public boolean closePart(NXPart part);

     public Collection<String> getComponentList(NXPart
part);
     public Collection<NXExpression>
getExpressions(NXPart part);
     public Collection<NXFeature> getFeatures(NXPart
part);
     public Collection<NXExpression>
getParameterInfo(NXPart part);

     public String getUniqueIdentifier(NXPart part);

     boolean isConnected();
     public NXPart openPart(File file);
     public NXPart openPart(File file, boolean
recursive);

     public boolean renameFeature(NXPart part, String
oldName, String newName);

     public boolean renameParameter(NXPart part,
String oldName, String newName);
     public boolean savePart(NXPart file);
```

```
          public boolean setParameterInfo(NXPart part,
Collection<NXExpression> params);
          public boolean setParameterValue(NXPart part,
String param, String value);

          public boolean setWorkPart(NXPart part);

     }
```

NXExpression:

```
     package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;

     public class NXExpression {
          private String _name;
          private Property _prop;
          private String _value;

          public NXExpression(String name, String value,
Property prop) {
               _name = name;
               _value = value;
               _prop = prop;
          }

          public String getName() { return _name; }

          public Property getProperty() { return _prop; }

          public String getValue() { return _value; }
     }
```

NXFeature:

```
package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

import java.util.ArrayList;
import java.util.Collection;

public class NXFeature {
     private ArrayList<NXFeature> _children;
     private ArrayList<NXExpression> _expressions;
     private String _name;
     private String _type;
     private String _customName;

     public NXFeature(String name, String type, String
customName) {
          _name = name;
          _type = type;
          _customName = customName;
          _children = new ArrayList<NXFeature>();
          _expressions = new
ArrayList<NXExpression>();
     }

     public void addChild(NXFeature feature) {
          _children.add(feature);
     }

     public void addExpression(String name, String
value) {
          _expressions.add( new NXExpression(name,
value, null) );
     }

     public Collection<NXFeature> getChildren() {
          return _children;
     }

     public String getCustomName() {
          return _customName;
     }

     public Collection<NXExpression> getExpressions()
{
          return _expressions;
     }

     public String getName() {
```

```
            return _name;
        }

        public String getType() {
            return _type;
        }
    }
```

NXPart:

```
package gov.nasa.jpl.imce.sysmlnxsync.nxconnection;

import java.io.File;
import java.util.ArrayList;
import java.util.Collection;

public class NXPart {
     protected Collection<String> _components;
     protected Collection<NXExpression> _expressions;
     protected Collection<NXFeature> _features;
     private File _file;
     protected Collection<NXPart> _openComponents;
     protected String _uid;

     protected NXPart(File file) {
          _file = file;
          _components = new ArrayList<String>();
          _openComponents = new ArrayList<NXPart>();
          _features = new ArrayList<NXFeature>();
          _expressions = new
ArrayList<NXExpression>();
     }

     protected NXPart(String path, String name) {
          _file = new File( path );
     }

     public Collection<String> getComponents() {
          return _components;
     }

     public Collection<NXExpression> getExpressions()
{
          return _expressions;
     }

     public Collection<NXFeature> getFeatures() {
          return _features;
     }

     public String getName() {
          return _file.getName();
     }

     public File getFile() {
          return _file;
```

```
          }

          public Collection<NXPart> getOpenComponents() {
               return _openComponents;
          }

          public String getPath() {
               return _file.getAbsolutePath();
          }

          public String getUniqueIdentifier() {
               return _uid;
          }

          public boolean isAssembly() {
               return (_components.size() > 0);
          }

          @Override
          public String toString() {
               String c = (_components != null ?
Integer.toString(_components.size()) : "null");
               String f = (_features != null ?
Integer.toString(_features.size()) : "null");
               return String.format(
"NXPart[%s][c=%s][f=%s]", _file, c, f );
          }
     }
```

Actions:

Import NXPart:

```
     package gov.nasa.jpl.imce.sysmlnxsync.actions;

     import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
     import gov.nasa.jpl.imce.sysmlnxsync.ui.WaitDialog;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.PartFileFilter;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateFromNXResolver;

     import java.awt.Frame;
     import java.awt.event.ActionEvent;
     import java.io.File;
     import java.util.Collection;

     import javax.swing.JFileChooser;
     import javax.swing.JOptionPane;

     import com.nomagic.magicdraw.core.Application;
     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
     import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
     import
com.nomagic.magicdraw.openapi.uml.SessionManager;
     import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
     import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
     import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     /**
      * Simple action that allows for a file to be selected
using a file dialog - the file is then loaded
      * using the Maple API and a corresponding SysML block
with properties is created. Note that for reasons
      * of making this more readable, the logic should be
split into several classes, but, to get started,
      * let's keep everything in one file
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */
     public class ImportNXPart extends DefaultBrowserAction
{
          private File _file;
          private Package _package;

          /**   * Constructor - configures the action, in
this case our menu item
           */
          public ImportNXPart() {
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
               super("Import_NX_Part", "Import CAD Model",
null, null);
          }

          /**
           * This function (or action) will be fired
whenever a user clicks on the menu item that we are
           * describing in this class. I.e. whenever
someone right clicks in the containment browser and
           * selects our action, in this case "Import CAD
Part ...", this function will be called
           */
          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
               super.actionPerformed(actionEvent);
```

```
                Object userObject = getSelectedObject();

                if (!(userObject instanceof Package)) {
                     return;
                }
                Project project =
Application.getInstance().getProject();
                _package = (Package)userObject;

                // Create a new file chooser object
                JFileChooser fc = new JFileChooser();
                fc.setFileFilter( new PartFileFilter() );

                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
                // Show an "Open" dialog and check whether
the user has chosen to select a file (and has not
                // pressed "Cancel")
                if (fc.showOpenDialog(parentFrame) !=
JFileChooser.APPROVE_OPTION) return;

                // Get the file that was selected
                _file = fc.getSelectedFile();
                boolean check = confirmUpdate( parentFrame,
project, _file );
                if (!check) return;

                importNX( project );
           }

          private boolean confirmUpdate(Frame parentFrame,
Project project, File filePart) {
                Collection<Class> parts =
SysMLUtility.getAllParts(project, _package);
                Class res =
SysMLUtility.findPartByFilePath(project, parts, filePart);
                if (res != null) {
                     int response =
JOptionPane.showConfirmDialog( parentFrame,
                               "Model element file already exists
in SysML.\nOverride existing part model?",
                               "Model element already exists",
                               JOptionPane.OK_CANCEL_OPTION,
                               JOptionPane.WARNING_MESSAGE
                     );
                     return (response ==
JOptionPane.YES_OPTION);
```

```
                } else {
                    return true;
                }
            }

            public void importNX( Project project ) {
                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();

                NXPart part =
SysMLUtility.openPart(parentFrame, _file);
                if (part == null) return;

                WaitDialog waitDialog = new
WaitDialog(parentFrame, "Importing data from NX...",
"System is working");
                waitDialog.setVisible(true);

                String fileName = _file.getName();
                ElementsFactory elementsFactory =
project.getElementsFactory();

                ElementsFactory factory = elementsFactory;

                //Interaction interaction = (Interaction)
ModelHelper.findInParent(project.getModel(),
"Interaction1", Interaction.class, true);

                //Lifeline lifeline1 =
factory.createLifelineInstance();
                //Lifeline lifeline2 =
factory.createLifelineInstance();

                //Connector connector =
factory.createConnectorInstance();
                //connector.setOwner(interaction);

                //ModelHelper.setClientElement(connector,
lifeline1.getRepresents());
                //ModelHelper.setSupplierElement(connector,
lifeline2.getRepresents());

                //PresentationElementsManager manager =
PresentationElementsManager.getInstance();
                //DiagramPresentationElement diagramView =
getDiagramPresentationElement();
```

```
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                Class sysmlPart =
elementsFactory.createClassInstance();
                sysmlPart.setName( fileName );
                StereotypesHelper.addStereotype(sysmlPart,
nxPartStereotype);


     SessionManager.getInstance().createSession(project,
"CAD Plugin: add subtree");
                try {

     ModelElementsManager.getInstance().addElement(sysmlPar
t, _package);
                } catch (ReadOnlyElementException roee) {
                     throw new IllegalStateException("ROEE:
Cannot add package to subpackage");
                }

     SessionManager.getInstance().closeSession(project);

                SysMLModelTraverser traverser =
SysMLModelTraverser.launch( project, sysmlPart, part, new
UpdateFromNXResolver(null) );
                Class resolvedClass =
traverser.getResolvedClass();

                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "Resolved
class : " + resolvedClass );
                }
                //traverser = SysMLModelTraverser.launch(
project, _package, resolvedClass, part, null );
                waitDialog.setVisible(false);
           }
     }
```

ImportNXPartWithFilter:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.ui.StereotypeFilterDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.PartFileFilter;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.StereotypeFilterHandl
er;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateFromNXResolver;

import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.io.File;
import java.util.Collection;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.magicdraw.openapi.uml.SessionManager;
import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     public class ImportNXPartWithFilter extends
DefaultBrowserAction {
          private File _file;

          private Package _package;

          /**   * Constructor - configures the action, in
this case our menu item
           */
          public ImportNXPartWithFilter() {
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
               super("Import_NX_Part_Filtered", "Import CAD
Model with Feature Type Filter", null, null);
          }

          /**
           * This function (or action) will be fired
whenever a user clicks on the menu item that we are
           * describing in this class. I.e. whenever
someone right clicks in the containment browser and
           * selects our action, in this case "Import CAD
Part ...", this function will be called
           */
          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
               super.actionPerformed(actionEvent);

               Object userObject = getSelectedObject();

               if (!(userObject instanceof Package)) {
                    return;
               }
```

```
                Project project =
Application.getInstance().getProject();
                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();

                // Create a new file chooser object
                JFileChooser fc = new JFileChooser();
                fc.setFileFilter( new PartFileFilter() );

                // Show an "Open" dialog and check whether
the user has chosen to select a file (and has not
                // pressed "Cancel")
                if (fc.showOpenDialog(parentFrame) !=
JFileChooser.APPROVE_OPTION) return;

                // Get the file that was selected
                File file = fc.getSelectedFile();

                Collection<Stereotype> st =
SysMLUtility.getProfileStereotypes( project );
                Package pkg = (Package)userObject;

                boolean check = confirmUpdate( parentFrame,
project, pkg, file );
                if (!check) return;

                if (st == null) {
                     throw new IllegalStateException();
                }
                _package = pkg;
                _file = file;

                final StereotypeFilterDialog stFilter = new
StereotypeFilterDialog(parentFrame, st);
                stFilter.setVisible(true);

                Collection<Stereotype> filter =
stFilter.getFilter();
                if (filter != null) {

     Application.getInstance().getGUILog().log( "Showing
filtered stereotypes: " + filter.size() );
                     for (Stereotype s : filter) {

     Application.getInstance().getGUILog().log( "ST: " + s
);
                     }
```

```
     Application.getInstance().getGUILog().log( "End
Showing filtered stereotypes: " + filter.size() );
                    importWithFilter( project, filter );
               }
          }

          private boolean confirmUpdate(Frame parentFrame,
Project project, Package pkg, File filePart) {
               Collection<Class> parts =
SysMLUtility.getAllParts(project, pkg);
               Class res =
SysMLUtility.findPartByFilePath(project, parts, filePart);
               if (res != null) {
                    int response =
JOptionPane.showConfirmDialog( parentFrame,
                         "Model element file already exists
in SysML.\nOverride existing part model?",
                         "Model element already exists",
                         JOptionPane.OK_CANCEL_OPTION,
                         JOptionPane.WARNING_MESSAGE
                    );
                    return (response ==
JOptionPane.YES_OPTION);
               } else {
                    return true;
               }
          }

          public Property enterExpression( Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
               return sysmlExpression;
          }

          public void importWithFilter( Project project,
Collection<Stereotype> filter ) {
               Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();

               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "Size of
filtered set" + filter.size() );
               }
               NXPart part =
SysMLUtility.openPart(parentFrame, _file);
```

```
                if (part == null) return;

                String fileName = _file.getName();
                ElementsFactory elementsFactory =
project.getElementsFactory();

                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                Class sysmlPart =
elementsFactory.createClassInstance();
                sysmlPart.setName( fileName );
                StereotypesHelper.addStereotype(sysmlPart,
nxPartStereotype);


     SessionManager.getInstance().createSession(project,
"CAD Plugin: add subtree");
                try {

     ModelElementsManager.getInstance().addElement(sysmlPar
t, _package);
                } catch (ReadOnlyElementException roee) {
                     throw new IllegalStateException("ROEE:
Cannot add package to subpackage");
                }

     SessionManager.getInstance().closeSession(project);

                SysMLModelTraverser traverser =
SysMLModelTraverser.launch( project, sysmlPart, part, new
UpdateFromNXResolver(null) );
                Class resolvedClass =
traverser.getResolvedClass();

                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "Resolved
class : " + resolvedClass );
                }
                StereotypeFilterHandler handler = new
StereotypeFilterHandler(filter);
                Application.getInstance().getGUILog().log(
"Resolved class : " + handler );
                traverser = SysMLModelTraverser.launch(
project, resolvedClass, null, handler );
           }
     }
```

InternalUpdate:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.ui.StereotypeFilterDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.PartFileFilter;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.StereotypeFilterHandl
er;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateFromNXResolver;

import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.io.File;
import java.util.Collection;

import javax.swing.JFileChooser;
import javax.swing.JOptionPane;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.magicdraw.openapi.uml.SessionManager;
import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     public class ImportNXPartWithFilter extends
DefaultBrowserAction {
          private File _file;

          private Package _package;

          /**   * Constructor - configures the action, in
this case our menu item
           */
          public ImportNXPartWithFilter() {
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
               super("Import_NX_Part_Filtered", "Import CAD
Model with Feature Type Filter", null, null);
          }

          /**
           * This function (or action) will be fired
whenever a user clicks on the menu item that we are
           * describing in this class. I.e. whenever
someone right clicks in the containment browser and
           * selects our action, in this case "Import CAD
Part ...", this function will be called
           */
          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
               super.actionPerformed(actionEvent);

               Object userObject = getSelectedObject();

               if (!(userObject instanceof Package)) {
                    return;
               }
```

```
                Project project =
Application.getInstance().getProject();
                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();

                // Create a new file chooser object
                JFileChooser fc = new JFileChooser();
                fc.setFileFilter( new PartFileFilter() );

                // Show an "Open" dialog and check whether
the user has chosen to select a file (and has not
                // pressed "Cancel")
                if (fc.showOpenDialog(parentFrame) !=
JFileChooser.APPROVE_OPTION) return;

                // Get the file that was selected
                File file = fc.getSelectedFile();

                Collection<Stereotype> st =
SysMLUtility.getProfileStereotypes( project );
                Package pkg = (Package)userObject;

                boolean check = confirmUpdate( parentFrame,
project, pkg, file );
                if (!check) return;

                if (st == null) {
                     throw new IllegalStateException();
                }
                _package = pkg;
                _file = file;

                final StereotypeFilterDialog stFilter = new
StereotypeFilterDialog(parentFrame, st);
                stFilter.setVisible(true);

                Collection<Stereotype> filter =
stFilter.getFilter();
                if (filter != null) {

     Application.getInstance().getGUILog().log( "Showing
filtered stereotypes: " + filter.size() );
                     for (Stereotype s : filter) {

     Application.getInstance().getGUILog().log( "ST: " + s
);
                     }
```

```
     Application.getInstance().getGUILog().log( "End
Showing filtered stereotypes: " + filter.size() );
                    importWithFilter( project, filter );
               }
          }

          private boolean confirmUpdate(Frame parentFrame,
Project project, Package pkg, File filePart) {
               Collection<Class> parts =
SysMLUtility.getAllParts(project, pkg);
               Class res =
SysMLUtility.findPartByFilePath(project, parts, filePart);
               if (res != null) {
                    int response =
JOptionPane.showConfirmDialog( parentFrame,
                         "Model element file already exists
in SysML.\nOverride existing part model?",
                         "Model element already exists",
                         JOptionPane.OK_CANCEL_OPTION,
                         JOptionPane.WARNING_MESSAGE
                    );
                    return (response ==
JOptionPane.YES_OPTION);
               } else {
                    return true;
               }
          }

          public Property enterExpression( Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
               return sysmlExpression;
          }

          public void importWithFilter( Project project,
Collection<Stereotype> filter ) {
               Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();

               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "Size of
filtered set" + filter.size() );
               }
               NXPart part =
SysMLUtility.openPart(parentFrame, _file);
```

```
                if (part == null) return;

                String fileName = _file.getName();
                ElementsFactory elementsFactory =
project.getElementsFactory();

                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                Class sysmlPart =
elementsFactory.createClassInstance();
                sysmlPart.setName( fileName );
                StereotypesHelper.addStereotype(sysmlPart,
nxPartStereotype);


     SessionManager.getInstance().createSession(project,
"CAD Plugin: add subtree");
                try {

     ModelElementsManager.getInstance().addElement(sysmlPar
t, _package);
                } catch (ReadOnlyElementException roee) {
                     throw new IllegalStateException("ROEE:
Cannot add package to subpackage");
                }

     SessionManager.getInstance().closeSession(project);

                SysMLModelTraverser traverser =
SysMLModelTraverser.launch( project, sysmlPart, part, new
UpdateFromNXResolver(null) );
                Class resolvedClass =
traverser.getResolvedClass();

                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "Resolved
class : " + resolvedClass );
                }
                StereotypeFilterHandler handler = new
StereotypeFilterHandler(filter);
                Application.getInstance().getGUILog().log(
"Resolved class : " + handler );
                traverser = SysMLModelTraverser.launch(
project, resolvedClass, null, handler );
           }
     }
```

InternalValidate:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.ui.InstanceReportDialog;
import gov.nasa.jpl.imce.sysmlnxsync.ui.WaitDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.DefaultNodeHandler;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.InstanceReportResult;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;

import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.io.File;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
import
com.nomagic.magicdraw.openapi.uml.PresentationElementsManag
er;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.magicdraw.uml.symbols.DiagramPresentationElemen
t;
import
com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement;
```

```
     import com.nomagic.uml2.ext.jmi.helpers.ModelHelper;
     import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Association
;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Classifier;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpe
cification;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.StructuralF
eature;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     /**
      * Simple action that allows for a file to be selected
using a file dialog - the file is then loaded
      * using the Maple API and a corresponding SysML block
with properties is created. Note that for reasons
      * of making this more readable, the logic should be
split into several classes, but, to get started,
      * let's keep everything in one file
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */
     public class InternalValidate extends
DefaultBrowserAction {
```

```
          public static class UpdateResolver extends
DefaultNodeHandler {
               private InstanceSpecification
_featureInstance;
               private Collection<Stereotype> _filter;
               private HashMap<Class,ShapeElement> _hm =
new HashMap<Class,ShapeElement>();
               private Package _package;

               public UpdateResolver(Package pkg,
Collection<Stereotype> filter) {
                    _package = pkg;
                    _filter = filter;
               }

               private void addChildClass(Project project,
Class classB, Class classA) {
                    if (classA == null || classB == null) {
                         return;
                    }
                    Element model = project.getModel();
                    ElementsFactory f =
project.getElementsFactory();
                    ModelElementsManager
modelElementsManager = ModelElementsManager.getInstance();

                    try {
                         Association link =
f.createAssociationInstance();
                         //Dependency dependency =
f.createDependencyInstance();

     modelElementsManager.addElement(link, model);
                        ModelHelper.setClientElement(link,
classA);

     ModelHelper.setSupplierElement(link, classB);

                        DiagramPresentationElement
activeDiagram = project.getActiveDiagram();
                        PresentationElementsManager
presentationElementsManager =
PresentationElementsManager.getInstance();

                        if (activeDiagram != null) {
                          ShapeElement clientShape;
```

```
                                if (!_hm.containsKey(classA))
{
                                     clientShape =
presentationElementsManager.createShapeElement(classA,
activeDiagram);
                                     _hm.put(classA,
clientShape);
                                } else {
                                     clientShape =
_hm.get(classA);
                                }
                                ShapeElement supplierShape;
                                if (!_hm.containsKey(classB))
{
                                     supplierShape =
presentationElementsManager.createShapeElement(classB,
activeDiagram);
                                     _hm.put(classB,
supplierShape);
                                } else {
                                     supplierShape =
_hm.get(classB);
                                }

     presentationElementsManager.createPathElement(link,
clientShape, supplierShape);
                           } else {

     Application.getInstance().getGUILog().log("activeDiagr
am is NULL ");
                           }
                      } catch (ReadOnlyElementException roee)
{
                      }
                }

                @Override
                public Property enterExpression(Project
project, Class parent,
                                Property sysmlExpression,
NXExpression nxExpression) throws ReadOnlyElementException
{
                      if (nxExpression != null) {
                           String nxName =
nxExpression.getName();
                           ElementsFactory elementsFactory =
project.getElementsFactory();
```

```
                         Stereotype
sysmlNXValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
                         Stereotype
sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project, "ValueProperty");

                         if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log(
                                        "Visiting Feature:
SysML: " + (sysmlExpression != null ?
sysmlExpression.getName() : "[NULL] ")
                                        + " NX : " +
(nxExpression != null ? nxExpression.getName() : "[NULL] ")
                              );
                         }

                         Property resolvedExpression;
                         LiteralString blockSpec;
                         LiteralString instanceSpec;
                         Slot slot;

                         // So we have the parameter, now
set the value
                         instanceSpec =
elementsFactory.createLiteralStringInstance();

     instanceSpec.setValue(nxExpression.getValue());

                         if (sysmlExpression != null) {
                              resolvedExpression =
sysmlExpression;
                              blockSpec =
(LiteralString)resolvedExpression.getDefaultValue();
                              // So we have the parameter,
now set the value

     blockSpec.setValue(nxExpression.getValue());
                              slot =
resolvedExpression.get_slotOfDefiningFeature().iterator().n
ext();
                         } else {
                              resolvedExpression =
elementsFactory.createPropertyInstance();
```

```
                                blockSpec =
elementsFactory.createLiteralStringInstance();
                                // So we have the parameter,
now set the value

     blockSpec.setValue(nxExpression.getValue());


     resolvedExpression.setName(nxName);

     StereotypesHelper.addStereotype(resolvedExpression,
sysmlNXValuePropertyStereotype);

     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "currentName",
nxName);

     StereotypesHelper.addStereotype(resolvedExpression,
sysmlValuePropertyStereotype);
                           //
     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "Type", "Real");

     resolvedExpression.setDefaultValue(blockSpec);
                                slot =
elementsFactory.createSlotInstance();

     slot.setDefiningFeature(resolvedExpression);

     slot.setOwningInstance(_featureInstance);


     ModelElementsManager.getInstance().addElement(resolved
Expression, parent);
                           }

                           slot.getValue().add(instanceSpec);

                           // LiteralReal realSpec =
elementsFactory.createLiteralRealInstance();
                           // realSpec.setValue(
Double.parseDouble( nxExpression.getValue() ) );

                           // Set instance relationships

                           if (PluginMain.DEBUG) {
```

```
     Application.getInstance().getGUILog().log("Updated
expression: " + resolvedExpression.getName() + " child of "
+ parent.getName() );
                         }
                         return resolvedExpression;
                    } else if (sysmlExpression != null) {

     ModelElementsManager.getInstance().removeElement(sysml
Expression);
                    }
                    return null;
               }

               @Override
               public Class enterFeature(Project project,
Class parent, Class sysmlFeature, NXFeature nxFeature)
throws ReadOnlyElementException {
                    if (nxFeature != null) {
                         ElementsFactory elementsFactory =
project.getElementsFactory();
                         Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");

                         Stereotype additionalStereotype =
null;
                         Class resolvedFeature;
                         if (sysmlFeature != null) {
                              resolvedFeature =
sysmlFeature;
                         } else {
                              String nxName =
nxFeature.getName();
                              resolvedFeature =
elementsFactory.createClassInstance();

     resolvedFeature.setName(nxName);

     StereotypesHelper.addStereotype(resolvedFeature,
nxFeatureStereotype);

                              if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("NX Feature
Type: " + nxFeature.getType() );
                              }
```

```
                                // Set instance relationships
                                _featureInstance =
elementsFactory.createInstanceSpecificationInstance();

     _featureInstance.setName(nxName + " instance");


     _featureInstance.getClassifier().add( resolvedFeature
);

     //resolvedFeature.setAppliedStereotypeInstance(_featur
eInstance);

     //
     StereotypesHelper.addStereotype(_featureInstance,
nxFeatureStereotype);

                                String type =
nxFeature.getType();
                                if (type != null) {
                                     additionalStereotype =
SysMLUtility.featureTypeToStereotype(project, type);
                                }
                                if (additionalStereotype !=
null) {

     StereotypesHelper.addStereotype(resolvedFeature,
additionalStereotype);
                                //
     StereotypesHelper.addStereotype(_featureInstance,
additionalStereotype);
                                }

                           }

                           if (_filter != null &&
additionalStereotype != null &&
_filter.contains(additionalStereotype)) {
                                // skip this feature and
absorb any children into its parent
                                return null;
                           } else {
                                // Set stereotype property
values

     StereotypesHelper.setStereotypePropertyValue(resolvedF
```

```
eature, nxFeatureStereotype, "currentFeatureName",
nxFeature.getName() );

     StereotypesHelper.setStereotypePropertyValue(resolvedF
eature, nxFeatureStereotype, "featureType",
nxFeature.getType() );


     ModelElementsManager.getInstance().addElement(resolved
Feature, parent);
          //
     Application.getInstance().getGUILog().log("Updated
feature: " + resolvedFeature.getName() + " child of " +
parent.getName() );
                              //addChildClass(project,
resolvedFeature, parent);

                              return resolvedFeature;
                         }
                    } else if (sysmlFeature != null) {

     ModelElementsManager.getInstance().removeElement(sysml
Feature);
                    }
                    return null;
               }

               @Override
               public Class enterPart(Project project,
Class parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
                    if (nxPart != null) {
                         ElementsFactory elementsFactory =
project.getElementsFactory();
                         Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(project, "NXPart");
                         Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(project, "NXAssembly");

                         if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log(
                                        "Visiting Part:
SysML: " + (sysmlPart != null ? sysmlPart.getName() :
"[NULL] ")
                                        + " NX : " +
(nxPart != null ? nxPart.getName() : "[NULL] ")
```

```
                                        );
                        }

                        Class resolvedPart;
                        if (sysmlPart != null) {
                             resolvedPart = sysmlPart;
                        } else {
                             resolvedPart =
elementsFactory.createClassInstance();

     resolvedPart.setName(nxPart.getName());

     StereotypesHelper.addStereotype(resolvedPart,
nxPartStereotype);
                        }

                        // Now set the appropriate
stereotypes
                        // Set some special stereotype
properties, in this case the filename and unique ID
                        File file = new File(
nxPart.getPath() );
                        String uid =
nxPart.getUniqueIdentifier();

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "directory", file.getParent());

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "currentPartPath",
file.getAbsolutePath());

     StereotypesHelper.setStereotypePropertyValue(sysmlPart
, nxPartStereotype, "uniqueID", uid);

                        if (nxPart.isAssembly()) {

     StereotypesHelper.addStereotype(resolvedPart,
nxAssemblyStereotype);
                        }
                        if (parent != null) {

     ModelElementsManager.getInstance().addElement(resolved
Part, parent);
                        }
                        //addChildClass(project,
resolvedPart, parent);
```

```
                         return resolvedPart;
                    } else if (sysmlPart != null) {

     ModelElementsManager.getInstance().removeElement(sysml
Part);
                    }
                    return null;
               }

          }
          private static Class
getBlock(InstanceSpecification is) {
               List<Classifier> classifierList =
is.getClassifier();
               for (Classifier classifier : classifierList)
{
                    if (classifier instanceof Class) {
                         return (Class)classifier;
                    }
               }
               return null;
          }

          private static final boolean
isPackageWithInstanceChildren(Object object) {
               if (!(object instanceof Package)) {
                    return false;
               }
               Package userPackage = (Package)object;

               // Check if it has any children which are
properties
               Collection<Element> children =
userPackage.getOwnedElement();
               for (Element child : children) {
                    if (child instanceof
InstanceSpecification) {
                         return true;
                    }
               }
               return false;
          }

          private List<InstanceReportResult> _report;

          private HashSet<String> _uniqueID;
```

```
          private boolean _result;

          /**
           * Constructor - configures the action, in this
case our menu item
           */
          public InternalValidate() {
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
               super("Validate_Internal", "Instance Results
Report", null, null);
          }

          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
               super.actionPerformed(actionEvent);

               Object userObject = getSelectedObject();
               if
(isPackageWithInstanceChildren(userObject) == false) {
                    return;
               }
               //ValidateResolver resolver = new
ValidateResolver();


               Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
               WaitDialog waitDialog = new
WaitDialog(parentFrame, "Performing internal consistency
check...", "System is working");
               waitDialog.setVisible(true);
               _report =
InstanceReportResult.generateList(Application.getInstance()
.getProject(),(Package)userObject);
               waitDialog.setVisible(false);

               //for (StructuralFeature  key : hm.keySet())
{
          //
     Application.getInstance().getGUILog().log(
          //                  "Slot name: " + key.getName()
+ " type: " + key +
```

```
          //                          " size: " +
hm.get(key).size()
          //        );
          //
          //        updateBlockFromSlot( hm.get(key).get(0)
);
          //   }

               InstanceReportDialog instanceReport = new
InstanceReportDialog(parentFrame, _report);
               instanceReport.setVisible(true);
          }

          private void fail(String name, String fname,
String pname, String type, String report) {
               String s = String.format("%s||%s||%s||%s",
fname, pname, type, report);
               if (!_uniqueID.contains(s)) {
                    _report.add( new
InstanceReportResult(name, fname, pname, type, report ) );
                    _uniqueID.add(s);
               }
               _result = false;
          }

          private void updateBlockFromSlot(Slot slot) {
               ElementsFactory elementsFactory =
Application.getInstance().getProject().getElementsFactory()
;
               StructuralFeature sfeature =
slot.getDefiningFeature();
               if (sfeature instanceof Property) {
                    List<ValueSpecification> lit =
slot.getValue();
                    if (lit.isEmpty()) { return; }
                    ValueSpecification val = lit.get(0);
                    ValueSpecification defaultValue;
                    if (val instanceof LiteralString) {
                         String sysmlValue =
((LiteralString)val).getValue();
                         LiteralString instanceSpec =
elementsFactory.createLiteralStringInstance();
                         instanceSpec.setValue(sysmlValue);
                         defaultValue = instanceSpec;
                    } else {
                         defaultValue = val;
                    }
```

```
                // Propagate default value
                Property prop = (Property)sfeature;
                if (defaultValue != null) {

prop.setDefaultValue(defaultValue);
                }
            }
        }
}
```

LinkNXPart:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.PartFileFilter;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateFromNXResolver;

import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.Collection;

import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.WindowConstants;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.NamedElemen
t;
import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
import com.nomagic.uml2.impl.ElementsFactory;
```

```
     /**
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */
     public class LinkNXPart extends DefaultBrowserAction {

          class NameChooserDialog extends JDialog
implements ActionListener {
               private JButton _cancelButton;
               private JComboBox _combo;
               private JButton _confirmButton;
               private String _result;

               public NameChooserDialog(Frame parent,
String sysmlPartName, String nxPartName) {
                    super(parent, "Select part name",
true);

                    Box vbox = Box.createVerticalBox();
                    Box box = Box.createHorizontalBox();

                    vbox.add( Box.createVerticalStrut(20));


                    JLabel label = new JLabel( "Select part
name:");
                    _combo = new JComboBox();
                    _combo.setEditable(true);

                    box.add(label);
                    box.add(_combo);
                    if (sysmlPartName != null &&
sysmlPartName.trim().length() > 0) {
                         _combo.addItem( sysmlPartName );
                    }
                    if (nxPartName != null &&
nxPartName.trim().length() > 0) {
                         _combo.addItem( nxPartName );
                    }
                    vbox.add(box);

                    vbox.add( Box.createVerticalStrut(20));

                    box = Box.createHorizontalBox();
                    _cancelButton = new JButton("Cancel");
```

```
                    _confirmButton = new
JButton("Confirm");
                    box.add(_cancelButton);
                    box.add(_confirmButton);
                    vbox.add(box);

                    vbox.add( Box.createVerticalStrut(20));

                    add( vbox );
                    setPreferredSize( new Dimension(300,
150));
                    setLocationRelativeTo( parent );
                    pack();
                    setDefaultCloseOperation(
WindowConstants.HIDE_ON_CLOSE);

                    _cancelButton.addActionListener(this);
                    _confirmButton.addActionListener(this);
               }

               @Override
               public void actionPerformed(ActionEvent ae)
{
                    Object target = ae.getSource();
                    if (target == _cancelButton) {
                         setVisible(false);
                    } else if (target == _confirmButton) {
                         _result =
_combo.getSelectedItem().toString();
                         setVisible(false);
                    }
               }

               public String getResult() {
                    return _result;
               }

          }

          /**
           * Constructor - configures the action, in this
case our menu item
           */
          public LinkNXPart() {
               // In the containment browser context pop
the text that will be displayed in the menu item
```

```
                // is what we specify as the second
argument. The first argument is an ID
                super("Link_NX_Part", "Link CAD Model to
Existing SysML Model", null, null);

                // Configure the maple engine arguments -
must be a list of strings with the first element being
"java"

           }

           /**
            * This function (or action) will be fired
whenever a user clicks on the menu item that we are
            * describing in this class. I.e. whenever
someone right clicks in the containment browser and
            * selects our action, in this case "Link NX Part
...", this function will be called
            */
           @Override
           public void actionPerformed(ActionEvent
actionEvent) {
                super.actionPerformed(actionEvent);

                Object userObject = getSelectedObject();
                if (!(userObject instanceof Class)) {
                     return;
                }
                Class userClass = (Class)userObject;
                Project project =
Application.getInstance().getProject();
                ElementsFactory elementsFactory =
project.getElementsFactory();

                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                if
(!(StereotypesHelper.hasStereotype(userClass,
nxPartStereotype))) {
                     return;
                }

                JFileChooser fc = new JFileChooser();
                fc.setFileFilter( new PartFileFilter() );

                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
```

```
                // Show an "Open" dialog and check whether
the user has chosen to select a file (and has not pressed
"Cancel")
                if(fc.showOpenDialog(parentFrame) ==
JFileChooser.APPROVE_OPTION) {
                    // Get the file that was selected
                    File file = fc.getSelectedFile();

                    Collection<Class> parts =
SysMLUtility.getPartChildren(project, userClass);
                    Class search =
SysMLUtility.findPartByFilePath( project, parts, file );

                    if (search != null) {


     JOptionPane.showMessageDialog(parentFrame, "File
already linked", "Selected file already exists in SysML",
JOptionPane.ERROR_MESSAGE);
                         return;
                    }

                    String partName = choosePartName( file,
userClass );
                    userClass.setName( partName );

                    NXPart part =
SysMLUtility.openPart(parentFrame, file);
                    // DOES NOT WORK
                    SysMLModelTraverser.launch( project,
userClass, part, new UpdateFromNXResolver(null)  );
                }
           }

           private String choosePartName(File file,
NamedElement sysmlElement) {
                Frame parent =
MDDialogParentProvider.getProvider().getDialogParent();

                if (!file.exists()) {
                    JOptionPane.showMessageDialog(parent,
"File not found", "Selected file not found",
JOptionPane.ERROR_MESSAGE);
                    return null;
                }

                String partName = sysmlElement.getName();
```

```
                String nxPartName = file.getName();

                NameChooserDialog dialog = new
NameChooserDialog(parent, partName, nxPartName);

                dialog.setVisible(true);
                //System.out.println("Dialog Output: " +
partName);
                partName = dialog.getResult();

                if (partName != null &&
!partName.toLowerCase().endsWith(".prt")) {
                     partName = partName + ".prt";
                }
                return partName;
          }

     /*   private void updateComponent(Project project,
NamedElement sysmlElement, String partName, File file,
String uid) {
                if (partName == null) {
                     return;
                }
                sysmlElement.setName( partName );
                Stereotype nxPartStereotype =
NXStereotype.getStereotype(Application.getInstance().getPro
ject(), sysmlElement);
                if (nxPartStereotype != null) {

     StereotypesHelper.setStereotypePropertyValue(sysmlElem
ent, nxPartStereotype, "directory", file.getParent());

     StereotypesHelper.setStereotypePropertyValue(sysmlElem
ent, nxPartStereotype, "currentPartPath",
file.getAbsolutePath());

     StereotypesHelper.setStereotypePropertyValue(sysmlElem
ent, nxPartStereotype, "uniqueID", uid);
                }
          }

          /*private void linkPart (File file, NamedElement
userElement) {
                Project project =
Application.getInstance().getProject();

                String filename = file.getAbsolutePath();
```

```
                Stereotype st =
NXStereotype.getStereotype(project, userElement);
                System.out.println("Read stereotype " +
st.getName());

                NamedElement search =
SysMLUtility.findPartByName( project,
(Class)userElement.getOwner(), filename );
                Frame parent =
MDDialogParentProvider.getProvider().getDialogParent();
                if (search != null) {

                     JOptionPane.showMessageDialog(parent,
"File already linked", "Selected file already exists in
SysML", JOptionPane.ERROR_MESSAGE);
                     return;
                }

                WaitDialog waitDialog = new
WaitDialog(parent);
                waitDialog.setVisible(true);

                NXConnection engine =
NXClientEngine.getInstance();
                boolean success;
                success = engine.openConnection();
                System.out.println("Open connection: " +
success);
                NXPart part = engine.openPart( file );
                success = (part != null);
                System.out.println("Open part: " +
success);
                if (!success) {
                     return;
                }
                String uid = engine.getUniqueIdentifier(
part );
                System.out.println("Get UID: " + uid);
                NXExpressionList nxParams =
engine.getParameterInfo( part );
                Collection<String> components =
engine.getComponentList( part );
                success = engine.closeConnection();
                System.out.println("Close connection: " +
success);
                //int numParams = nxParams.size();
```

```
                //System.out.println("NumParams: " +
numParams);

                String partName = choosePartName( file,
userElement );

                //NamedElement otherElement =
SysMLUtility.getCorrespondingElement(project, userElement,
st);

                linkPartInternal( project, userElement,
partName, file, uid, nxParams, components );
                //if (otherElement != null) {
                //   linkPartInternal( project,
otherElement, partName, file, uid, nxParams, components );
                //}

                //SysMLUtility.linkComponents(project,
userElement, otherElement, components);

                waitDialog.setVisible(false);
          }

          private void linkPartInternal(Project project,
NamedElement element, String partName, File file, String
uid, NXExpressionList nxParams, Collection<String>
components) {

     SysMLUtility.clearAssemblyComponents(project,
element);
                updateComponent( project, element, partName,
file, uid );

                if (element instanceof Class) {
                     int numParams = nxParams.size();

     SysMLParameters.removeAllParameters(element);
                     for (NXExpression param : nxParams) {

     SysMLParameters.addParameterToClass( elementsFactory,
element, param );
                          System.out.printf( "%s -> %s\n",
param.getName(), param.getValue() );
                     }

                     if (components.size() > 0) {
```

```
                        Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(Application.getInstance().g
etProject(), "NXAssembly");

     StereotypesHelper.addStereotype(element,
nxAssemblyStereotype);
                    }
                }

          }*/

     }
```

ResolveNXPart:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXClientEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXConnectionExce
ption;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.ui.ConsistencyReportDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.ui.InteractiveConsistencyRepo
rtDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.ConsistencyResult;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.DefaultNodeHandler;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateObject;

import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;

import javax.swing.JDialog;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
```

```
     import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
     import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
     import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
     import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     public class ResolveNXPart extends
DefaultBrowserAction {

          static class FeatureNameUpdate extends
UpdateObject {
               private NXEngine _engine;
               private NXFeature _nxFeature;
               private NXPart _nxPart;
               private Project _project;
               private Class _sysmlFeature;

               public FeatureNameUpdate(NXEngine engine,
Project project, Class sysmlFeature, NXPart nxPart,
NXFeature nxFeature) {
                    _project = project;
                    _sysmlFeature = sysmlFeature;
                    _nxFeature = nxFeature;
                    _nxPart = nxPart;
               }
               @Override
               public boolean canUpdateNX() { return true;
}
               @Override
```

```
                public boolean canUpdateSysML() { return
true; }

                @Override
                public void updateNX() {
                     Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(_project, "NXPartFeature");
                     String newName =
_sysmlFeature.getName();
                     String oldName =
StereotypesHelper.getStereotypePropertyFirst(_sysmlFeature,
nxFeatureStereotype, "currentFeatureName").toString();

                     if (!oldName.equals(newName)) {
                          _engine.renameFeature( _nxPart,
oldName, newName );

     StereotypesHelper.setStereotypePropertyValue(_sysmlFea
ture, nxFeatureStereotype, "currentFeatureName", newName );
                     }
                }
                @Override
                public void updateSysML() {
                     Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(_project, "NXPartFeature");
                     String name = _nxFeature.getName();
                     _sysmlFeature.setName(name);

     StereotypesHelper.addStereotype(_sysmlFeature,
nxFeatureStereotype);
                }
          }

          static class FeatureTypeUpdate extends
UpdateObject {
                private NXFeature _nxFeature;
                private NXPart _nxPart;
                private Project _project;
                private Class _sysmlFeature;
                public FeatureTypeUpdate(Project project,
Class sysmlFeature, NXPart nxPart, NXFeature nxFeature) {
                     _project = project;
                     _sysmlFeature = sysmlFeature;
                     _nxFeature = nxFeature;
                     _nxPart = nxPart;
                }
                @Override
```

```
                public boolean canUpdateNX() { return false;
}
                @Override
                public boolean canUpdateSysML() { return
false; }
                @Override
                public void updateSysML() {
                     Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(_project, "NXPartFeature");
                     String type = _nxFeature.getType();
                     Stereotype additionalStereotype;
                     additionalStereotype = (type != null ?
SysMLUtility.featureTypeToStereotype(_project, type) :
null);
                     if (additionalStereotype != null) {

     StereotypesHelper.addStereotype(_sysmlFeature,
additionalStereotype);
                     }
                     if (additionalStereotype != null) {
                          // Set stereotype property values

     StereotypesHelper.setStereotypePropertyValue(_sysmlFea
ture, nxFeatureStereotype, "currentFeatureName",
_nxFeature.getName() );

     StereotypesHelper.setStereotypePropertyValue(_sysmlFea
ture, nxFeatureStereotype, "featureType", type );
                     }
                }
           }

           static class NewFeatureUpdate extends
UpdateObject {
                private NXFeature _nxFeature;
                private Project _project;
                private Class _sysmlParent;

                public NewFeatureUpdate(Project project,
Class parent, NXFeature nxFeature) {
                     _project = project;
                     _sysmlParent = parent;
                     _nxFeature = nxFeature;
                }

                @Override
```

```
                public boolean canUpdateNX() { return false;
}
                @Override
                public boolean canUpdateSysML() { return
true; }

                @Override
                public void updateSysML() {
                     ElementsFactory elementsFactory =
_project.getElementsFactory();
                     Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(_project, "NXPartFeature");

                     Stereotype additionalStereotype = null;
                     Class resolvedFeature =
elementsFactory.createClassInstance();
                     String name = _nxFeature.getName();
                     resolvedFeature.setName(name);

     StereotypesHelper.addStereotype(resolvedFeature,
nxFeatureStereotype);

                     String type = _nxFeature.getType();
                     if (type != null) {
                          additionalStereotype =
SysMLUtility.featureTypeToStereotype(_project, type);
                     }
                     if (additionalStereotype != null) {

     StereotypesHelper.addStereotype(resolvedFeature,
additionalStereotype);
                     }
                     // Set stereotype property values

     StereotypesHelper.setStereotypePropertyValue(resolvedF
eature, nxFeatureStereotype, "currentFeatureName", name );

     StereotypesHelper.setStereotypePropertyValue(resolvedF
eature, nxFeatureStereotype, "featureType",
_nxFeature.getType() );

                     try {

     ModelElementsManager.getInstance().addElement(resolved
Feature, _sysmlParent);
                     } catch (ReadOnlyElementException roe)
{
```

```
                    }
               }
          }

          static class NewParameterUpdate extends
UpdateObject {
               private NXExpression _nxExpression;
               private Project _project;
               private Class _sysmlParent;

               public NewParameterUpdate(Project project,
Class parent, NXExpression nxExpression) {
                    _project = project;
                    _sysmlParent = parent;
                    _nxExpression = nxExpression;
               }

               @Override
               public boolean canUpdateNX() { return false;
}
               @Override
               public boolean canUpdateSysML() { return
true; }

               @Override
               public void updateSysML() {
                    ElementsFactory elementsFactory =
_project.getElementsFactory();
                    Stereotype sysmlValuePropertyStereotype
= StereotypesHelper.getStereotype(_project,
"NXValueProperty");

                    String nxName =
_nxExpression.getName();
                    Property resolvedExpression =
elementsFactory.createPropertyInstance();
                    resolvedExpression.setName(nxName);

     StereotypesHelper.addStereotype(resolvedExpression,
sysmlValuePropertyStereotype);

     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "currentName",
nxName);

                    // So we have the parameter, now set
the value
```

```
                    LiteralString spec =
elementsFactory.createLiteralStringInstance();
                    spec.setValue( _nxExpression.getValue()
);

     resolvedExpression.setDefaultValue(spec);

                    try {

     ModelElementsManager.getInstance().addElement(resolved
Expression, _sysmlParent);
                    } catch (ReadOnlyElementException roe)
{
                    }
               }
          }

          static class NewPartUpdate extends UpdateObject {
               private NXEngine _engine;
               private NXPart _nxPart;
               private Project _project;
               private Class _sysmlParent;

               public NewPartUpdate(NXEngine engine,
Project project, Class parent, NXPart nxPart) {
                    _project = project;
                    _sysmlParent = parent;
                    _nxPart = nxPart;
               }

               @Override
               public boolean canUpdateNX() { return false;
}
               @Override
               public boolean canUpdateSysML() { return
true; }

               @Override
               public void updateSysML() {
                    ElementsFactory elementsFactory =
_project.getElementsFactory();
                    Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(_project, "NXPart");
                    Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(_project, "NXAssembly");
```

```
                    Class resolvedPart =
elementsFactory.createClassInstance();

     resolvedPart.setName(_nxPart.getName());

     StereotypesHelper.addStereotype(resolvedPart,
nxPartStereotype);

                    // Now set the appropriate stereotypes
                    // Set some special stereotype
properties, in this case the filename and unique ID
                    File file = new File( _nxPart.getPath()
);
                    String uid =
_nxPart.getUniqueIdentifier();

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "directory", file.getParent());

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "currentPartPath",
file.getAbsolutePath());

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "uniqueID", uid);

                    if (_nxPart.isAssembly()) {

     StereotypesHelper.addStereotype(resolvedPart,
nxAssemblyStereotype);
                    }
                    try {

     ModelElementsManager.getInstance().addElement(resolved
Part, _sysmlParent);
                    } catch (ReadOnlyElementException roe)
{
                    }
                    //addChildClass(project, resolvedPart,
parent);
               }
          }

          static class ParameterNameUpdate extends
UpdateObject {
               private NXEngine _engine;
               private NXExpression _nxExpression;
```

```
                private NXPart _nxPart;
                private Project _project;
                private Property _sysmlExpression;

                public ParameterNameUpdate(NXEngine engine,
Project project, Property sysmlExpression, NXPart nxPart,
NXExpression nxExpression) {
                     _project = project;
                     _sysmlExpression = sysmlExpression;
                     _nxExpression = nxExpression;
                     _nxPart = nxPart;
                     _engine = engine;
                }

                @Override
                public boolean canUpdateNX() { return true;
}
                @Override
                public boolean canUpdateSysML() { return
true; }
                @Override
                public void updateNX() {
                     Stereotype sysmlValuePropertyStereotype
= StereotypesHelper.getStereotype(_project,
"NXValueProperty");
                     String newName =
_sysmlExpression.getName();
                     String oldName =
StereotypesHelper.getStereotypePropertyFirst(_sysmlExpressi
on, sysmlValuePropertyStereotype,
"currentName").toString();

                     boolean result = true;
                     if (!oldName.equals(newName)) {
                          result = _engine.renameParameter(
_nxPart, oldName, newName );

     StereotypesHelper.setStereotypePropertyValue(_sysmlExp
ression, sysmlValuePropertyStereotype, "currentName",
newName );
                     }
                }

                @Override
                public void updateSysML() {
                     ElementsFactory elementsFactory =
_project.getElementsFactory();
```

```
                    Stereotype sysmlValuePropertyStereotype
= StereotypesHelper.getStereotype(_project,
"NXValueProperty");
                    String nxName =
_nxExpression.getName();
                    Property resolvedExpression =
elementsFactory.createPropertyInstance();
                    resolvedExpression.setName(nxName);

     StereotypesHelper.addStereotype(resolvedExpression,
sysmlValuePropertyStereotype);

     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "currentName",
nxName);
               }
          }

          static class ParameterValueUpdate extends
UpdateObject {
               private NXEngine _engine;
               private NXExpression _nxExpression;
               private NXPart _nxPart;
               private Project _project;
               private Property _sysmlExpression;

               public ParameterValueUpdate(NXEngine engine,
Project project, Property sysmlExpression, NXPart nxPart,
NXExpression nxExpression) {
                    _project = project;
                    _sysmlExpression = sysmlExpression;
                    _nxExpression = nxExpression;
                    _nxPart = nxPart;
                    _engine = engine;
               }
               @Override
               public boolean canUpdateNX() { return true;
}
               @Override
               public boolean canUpdateSysML() { return
true; }

               @Override
               public void updateNX() {
                    // Set stereotype property values
                    String newName =
_sysmlExpression.getName();
```

```
                    ValueSpecification spec =
_sysmlExpression.getDefaultValue();
                    String sysmlValue = (spec instanceof
LiteralString ? ((LiteralString)spec).getValue() : null );

     //Application.getInstance().getGUILog().log("Expr
sysmlValue: " + sysmlValue);
                    if (_nxExpression != null && sysmlValue
!= null && !sysmlValue.equals(_nxExpression.getValue())) {
                         NXExpression newExpr = new
NXExpression( newName, sysmlValue,
_nxExpression.getProperty() );
                         _engine.setParameterValue(
_nxPart, newName, sysmlValue );
                    }
               }

               @Override
               public void updateSysML() {
                    ElementsFactory elementsFactory =
_project.getElementsFactory();
                    // So we have the parameter, now set
the value
                    ValueSpecification spec =
_sysmlExpression.getDefaultValue();
                    if (spec instanceof LiteralString) {
                         ((LiteralString)
spec).setValue(_nxExpression.getValue());
                    } else {
                         throw new
IllegalStateException("illegal value specification type");
                    }
               }
          }

          static class PartNameUpdate extends UpdateObject
{
               private NXEngine _engine;
               private NXPart _nxPart;
               private Project _project;
               private Class _sysmlPart;

               public PartNameUpdate(NXEngine engine,
Project project, Class sysmlPart, NXPart nxPart) {
                    _project = project;
                    _sysmlPart = sysmlPart;
                    _nxPart = nxPart;
```

```
                    _engine = engine;
               }

               @Override
               public boolean canUpdateNX() { return true;
}
               @Override
               public boolean canUpdateSysML() { return
true; }

               @Override
               public void updateNX() {
                    Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(_project, "NXPart");
                    Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(_project, "NXAssembly");

                    // Now set the appropriate stereotypes
                    // Set some special stereotype
properties, in this case the filename and unique ID
                    String newName = _sysmlPart.getName();

                    String dir =
StereotypesHelper.getStereotypePropertyFirst(_sysmlPart,
nxPartStereotype, "directory").toString();
                    String filename = dir + File.separator
+ newName;

                    File newFile = new File( filename );

                    NXPart finalPart = _nxPart;
                    NXPart newPart =
SysMLUtility.renameNXPart( _engine, _nxPart, newFile );
                    if (newPart != null) {

     StereotypesHelper.setStereotypePropertyValue(
_sysmlPart, nxPartStereotype, "currentPartPath", filename
);
                         finalPart = newPart;
                    }
               }

               @Override
               public void updateSysML() {
                    ElementsFactory elementsFactory =
_project.getElementsFactory();
```

```
                    Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(_project, "NXPart");
                    Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(_project, "NXAssembly");

                    _sysmlPart.setName(_nxPart.getName());

     StereotypesHelper.addStereotype(_sysmlPart,
nxPartStereotype);

                    // Now set the appropriate stereotypes
                    // Set some special stereotype
properties, in this case the filename and unique ID
                    File file = new File( _nxPart.getPath()
);
                    String uid =
_nxPart.getUniqueIdentifier();

     StereotypesHelper.setStereotypePropertyValue(_sysmlPar
t, nxPartStereotype, "directory", file.getParent());

     StereotypesHelper.setStereotypePropertyValue(_sysmlPar
t, nxPartStereotype, "currentPartPath",
file.getAbsolutePath());

     StereotypesHelper.setStereotypePropertyValue(_sysmlPar
t, nxPartStereotype, "uniqueID", uid);

                    if (_nxPart.isAssembly()) {

     StereotypesHelper.addStereotype(_sysmlPart,
nxAssemblyStereotype);
                    }
               }
          }

          public class ValidateResolver extends
DefaultNodeHandler {
               private ArrayList<ConsistencyResult>
_consistencyReport;
               private NXEngine _engine;
               private ArrayList<NXPart> _partHierarchy;
               private boolean _result;
               private HashSet<String> _uniqueID;

               public ValidateResolver(NXEngine engine) {
```

```
                    _consistencyReport = new
ArrayList<ConsistencyResult>();
                    _uniqueID = new HashSet<String>();
                    _result = true;
                    _partHierarchy = new
ArrayList<NXPart>();
                    _engine = engine;
               }

               @Override
               public Property enterExpression(Project
project, Class parent,
                         Property sysmlExpression,
NXExpression nxExpression) {

                    String sysmlName = (sysmlExpression !=
null ? sysmlExpression.getName() : null);
                    NXPart nxPart = _partHierarchy.get(0);
                    if (nxExpression != null) {
                         String nxName =
nxExpression.getName();
                         if (sysmlExpression != null) {
                              if
(!nxName.equals(sysmlExpression.getName())) {
                                   fail(
                                        "Expression",
sysmlName,
                                        String.format("NX
parameter name %s differs from SysML parameter name",
nxName),
                                        new
ParameterNameUpdate(_engine, project, sysmlExpression,
nxPart, nxExpression)
                                   );
                              }
                         } else {
                              fail(
                                   "Expression", nxName,
"No SysML equivalent for NX parameter",
                                   new
NewParameterUpdate(project, parent, nxExpression)
                              );
                         }
                    } else if (sysmlExpression != null) {
                         fail("Expression", sysmlName, "No
NX equivalent for SysML parameter", errorUpdate );
                    }
```

```
                    if (sysmlExpression != null &&
nxExpression != null) {
                         ValueSpecification spec =
sysmlExpression.getDefaultValue();
                         String sysmlValue = (spec
instanceof LiteralString ? ((LiteralString)spec).getValue()
: null );
                         if (sysmlValue != null) {
                              if
(!sysmlValue.equals(nxExpression.getValue())) {
                                   fail(
                                        "Expression",
sysmlName,

     String.format("Value conflict: NX value %s differs
from SysML value %s", nxExpression.getValue(), sysmlValue),
                                        new
ParameterValueUpdate(_engine, project, sysmlExpression,
nxPart, nxExpression)
                                   );
                              }
                         } else {
                              fail("Expression",
nxExpression.getName(), "SysML parameter value cannot be
read", errorUpdate);
                         }
                    }
                    return sysmlExpression;
               }

               @Override
               public Class enterFeature(Project project,
Class parent, Class sysmlFeature, NXFeature nxFeature) {
                    Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");
                    String sysmlName = (sysmlFeature !=
null ? sysmlFeature.getName() : null);
                    NXPart nxPart = _partHierarchy.get(0);
                    if (nxFeature != null) {
                         String nxName =
nxFeature.getName();
                         if (sysmlFeature != null) {
                              if
(!nxName.equals(sysmlFeature.getName())) {
                                   fail(
```

```
                                        "Feature",
sysmlName,
                                        String.format("NX
feature name %s differs from SysML feature name", nxName),
                                        new
FeatureNameUpdate(_engine, project, sysmlFeature, nxPart,
nxFeature)
                                   );
                              }
                         } else {
                              fail(
                                   "Feature", nxName,
                                   "No SysML equivalent for
NX feature",
                                   new
NewFeatureUpdate(project, parent, nxFeature)
                              );
                         }
                    } else if (sysmlFeature != null) {
                         fail("Feature", sysmlName, "No NX
equivalent for SysML feature", errorUpdate );
                    }

                    //String sysmlFeatureName =
sysmlFeature.getName();
                    //String nxFeatureName =
nxFeature.getName();
                    //if
(!sysmlFeatureName.equals(nxFeatureName)) {
                    //   consistencyResult = new
ConsistencyResult( false, String.format("Feature name %s
differs from feature name %s", nxFeatureName,
sysmlFeatureName) );
                    //   return sysmlFeature;
                    //}

                    if (sysmlFeature != null && nxFeature
!= null) {
                         String nxFType =
nxFeature.getType();
                         if
(StereotypesHelper.hasStereotype(sysmlFeature,
nxFeatureStereotype) &&

     StereotypesHelper.getStereotypePropertyValue(sysmlFeat
ure, nxFeatureStereotype, "featureType") != null) {
```

```
                                Object childFtype =
StereotypesHelper.getStereotypePropertyFirst(sysmlFeature,
nxFeatureStereotype, "featureType");
                                String sysmlFType =
childFtype.toString();
                                if
(!sysmlFType.equals(nxFType)) {
                                     fail(
                                          "Feature",
sysmlName,
                                          String.format("NX
feature type %s differs from SysML feature type %s",
nxFType, sysmlFType),
                                          new
FeatureTypeUpdate(project, sysmlFeature, nxPart, nxFeature)
                                     );
                                     return sysmlFeature;
                                }
                           } else {
                                fail(
                                     "Feature", sysmlName,
                                     String.format("NX
feature type is %s, SysML has no feature type %s",
nxFType),
                                     new
FeatureTypeUpdate(project, sysmlFeature, nxPart, nxFeature)
                                );
                           }

                           Collection<NXExpression>
nxExpressions = nxFeature.getExpressions();
                           Collection<Property>
sysmlExpressions = SysMLUtility.getExpressions(project,
sysmlFeature);

                           if (nxExpressions.size() !=
sysmlExpressions.size()) {
                                fail("Feature", sysmlName,
String.format("%d parameters in NX, %d parameters in
SysML", nxExpressions.size(), sysmlExpressions.size()),
errorUpdate );
                                return sysmlFeature;
                           }

                           Collection<NXFeature>
nxSubfeatures = nxFeature.getChildren();
```

```
                         Collection<Class> sysmlSubfeatures
= SysMLUtility.getFeatures(project, sysmlFeature);

                         if (nxSubfeatures.size() !=
sysmlSubfeatures.size()) {
                              fail("Feature", sysmlName,
String.format("%d features in NX, %d features in SysML",
nxSubfeatures.size(), sysmlSubfeatures.size()), errorUpdate
);
                         }
                    }

                    return sysmlFeature;
               }

               @Override
               public Class enterPart(Project project,
Class parent, Class sysmlPart, NXPart nxPart) {
                    //ElementsFactory elementsFactory =
project.getElementsFactory();
     //             Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(project, "NXPart");
                    String sysmlName = (sysmlPart != null ?
sysmlPart.getName() : null);
                    _partHierarchy.add( 0, nxPart );
                    if (nxPart != null) {
                         String nxName = nxPart.getName();
                         if (sysmlPart != null) {
                              if
(!nxName.equals(sysmlPart.getName())) {
                                   fail(
                                        "Part", sysmlName,
                                        String.format("NX
part name %s differs from SysML part name %s", nxName),
                                        new
PartNameUpdate(_engine, project, sysmlPart, nxPart)
                                   );
                              }
                         } else {
                              fail(
                                   "Part", nxName,
                                   "No SysML equivalent for
NX part",
                                   new
NewPartUpdate(_engine, project, parent, nxPart)
                              );
                         }
```

```
                    } else if (sysmlPart != null) {
                         fail("Part", sysmlName, "No NX
equivalent for SysML part", errorUpdate );
                    }
                    return sysmlPart;
               }

               private void fail(String type, String id,
String report, UpdateObject obj) {
                    String s = String.format("%s||%s||%s",
type, id, report);
                    if (!_uniqueID.contains(s)) {
                         _consistencyReport.add( new
ConsistencyResult( type, id, report, obj ) );
                         _uniqueID.add(s);
                    }
                    _result = false;
               }

               public List<ConsistencyResult>
getInconsistencyList() {
                    return _consistencyReport;
               }

               public boolean getResult() {
                    return _result;
               }
          }

          ElementsFactory elementsFactory =
Application.getInstance().getProject().getElementsFactory()
;

          UpdateObject errorUpdate = new UpdateObject() {
               @Override
               public boolean canUpdateNX() { return true;
}
               @Override
               public boolean canUpdateSysML() { return
true; }
          };

          /**
           * Constructor - configures the action, in this
case our menu item
           */
          public ResolveNXPart() {
```

```
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
               super("Resolve_NX_Part_Maple", "Resolve
SysML-CAD Inconsistencies", null, null);
          }

          /**
           * This function (or action) will be fired
whenever a user clicks on the menu item that we are
           * describing in this class. I.e. whenever
someone right clicks in the containment browser and
           * selects our action, in this case "Import NX
Part ...", this function will be called
           */
          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
               super.actionPerformed(actionEvent);

               NXEngine engine;
               try {
                    engine = new NXClientEngine();
               } catch (NXConnectionException nxce) {
                    engine = null;
               }
               if (engine == null ) { return; }

               Object userObject = getSelectedObject();
               if (!(userObject instanceof Class)) {
                    return;
               }
               Class userClass = (Class)userObject;
               Project project =
Application.getInstance().getProject();
               Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
               if
(!(StereotypesHelper.hasStereotype(userClass,
nxPartStereotype))) {
                    return;
               }

               Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
```

```
     //        WaitDialog waitDialog = new
WaitDialog(parent);
     //        waitDialog.setVisible(true);
               //ProgressMonitor pm = new
ProgressMonitor(parentFrame, JOptionPane.PLAIN_MESSAGE,
"Please wait", 0, 10);

               String currentPartPath =
StereotypesHelper.getStereotypePropertyFirst(userClass,
nxPartStereotype, "currentPartPath").toString();
               File file = new File( currentPartPath );
               ValidateResolver resolver = new
ValidateResolver(engine);

               NXPart part =
SysMLUtility.openPart(parentFrame, file);
               SysMLModelTraverser.launch( project,
userClass, part, resolver );

               JDialog consistencyReport;
               List<ConsistencyResult> res =
resolver.getInconsistencyList();
               if (res == null) { res = new
ArrayList<ConsistencyResult>(); };
               if (res.size() > 0) {
                    consistencyReport = new
InteractiveConsistencyReportDialog(parentFrame, res,
project, userClass, file, part);
               } else {
                    consistencyReport = new
ConsistencyReportDialog(parentFrame, res);
               }
               consistencyReport.setVisible(true);
          }

     }
```

UpdateFromNXPart:

```
     package gov.nasa.jpl.imce.sysmlnxsync.actions;

     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
     import gov.nasa.jpl.imce.sysmlnxsync.ui.WaitDialog;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.DefaultNodeHandler;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
```

```
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
     import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateFromNXResolver;

     import java.awt.Frame;
     import java.awt.event.ActionEvent;
     import java.io.File;

     import com.nomagic.magicdraw.core.Application;
     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
     import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
     import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

     /**
      * Simple action that allows for a file to be selected
using a file dialog - the file is then loaded
      * using the Maple API and a corresponding SysML block
with properties is created. Note that for reasons
      * of making this more readable, the logic should be
split into several classes, but, to get started,
      * let's keep everything in one file
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */
     public class UpdateFromNXPart extends
DefaultBrowserAction {

          /**
           * Constructor - configures the action, in this
case our menu item
           */
          public UpdateFromNXPart() {
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
```

```
                super("Update_Part_from_NX", "Update SysML
Model from CAD Model", null, null);
          }


          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
                super.actionPerformed(actionEvent);

                Object userObject = getSelectedObject();
                if (!(userObject instanceof Class)) {
                     return;
                }
                Class userClass = (Class)userObject;
                Project project =
Application.getInstance().getProject();
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                if
(!(StereotypesHelper.hasStereotype(userClass,
nxPartStereotype))) {
                     return;
                }
                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
                WaitDialog waitDialog = new
WaitDialog(parentFrame, "Updating from NX...", "System is
working");
                waitDialog.setVisible(true);

                Object currentPartPath =
StereotypesHelper.getStereotypePropertyFirst(userClass,
nxPartStereotype, "currentPartPath");
                File nxFile = new File(
currentPartPath.toString() );

     //         ProgressMonitor pm = new
ProgressMonitor(parentFrame, JOptionPane.PLAIN_MESSAGE,
"Please wait", 0, 10);

                DefaultNodeHandler resolver = new
UpdateFromNXResolver(null);
                NXPart part =
SysMLUtility.openPart(parentFrame, nxFile);
                SysMLModelTraverser.launch( project,
userClass, part, resolver );
```

```
            resolver = null;
            // Read parameter and components

//          pm.close();
            waitDialog.setVisible(false);
        }
}
```

UpdateToNXPart:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXClientEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXConnectionExce
ption;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import gov.nasa.jpl.imce.sysmlnxsync.ui.WaitDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateToNXResolver;

import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.io.File;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
import com.nomagic.uml2.impl.ElementsFactory;


/**
 * Simple action that allows for a file to be selected
using a file dialog - the file is then loaded
 * using the Maple API and a corresponding SysML block
with properties is created. Note that for reasons
 * of making this more readable, the logic should be
split into several classes, but, to get started,
```

```
      * let's keep everything in one file
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */
     public class UpdateToNXPart extends
DefaultBrowserAction {



          ElementsFactory elementsFactory =
Application.getInstance().getProject().getElementsFactory()
;

          /**
           * Constructor - configures the action, in this
case our menu item
           */
          public UpdateToNXPart() {
               // In the containment browser context pop
the text that will be displayed in the menu item
               // is what we specify as the second
argument. The first argument is an ID
               super("Update_NX_Part_Maple", "Update CAD
Model from SysML Model", null, null);
          }

          /**
           * This function (or action) will be fired
whenever a user clicks on the menu item that we are
           * describing in this class. I.e. whenever
someone right clicks in the containment browser and
           * selects our action, in this case "Import CAD
Part ...", this function will be called
           */
          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
               super.actionPerformed(actionEvent);

               NXEngine engine;
               try {
                    engine = new NXClientEngine();
               } catch (NXConnectionException nxce) {
                    engine = null;
               }
               if (engine == null ) { return; }
```

```
                Object userObject = getSelectedObject();
                if (!(userObject instanceof Class)) {
                     return;
                }
                Class userClass = (Class)userObject;
                Project project =
Application.getInstance().getProject();
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                if
(!(StereotypesHelper.hasStereotype(userClass,
nxPartStereotype))) {
                     return;
                }
                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
                WaitDialog waitDialog = new
WaitDialog(parentFrame, "Updating to NX...", "System is
working");
                waitDialog.setVisible(true);
           //        ProgressMonitor pm = new
ProgressMonitor(parentFrame, JOptionPane.PLAIN_MESSAGE,
"Please wait", 0, 10);

                Object currentPartPath =
StereotypesHelper.getStereotypePropertyFirst(userClass,
nxPartStereotype, "currentPartPath");
                File nxFile = new File(
currentPartPath.toString() );

                NXPart part =
SysMLUtility.openPart(parentFrame, nxFile);
                SysMLModelTraverser.launch( project,
userClass, part, new UpdateToNXResolver(engine) );

                waitDialog.setVisible(false);
     //                   pm.close();

           }

     }
```

ValidateAgainstNXPart:

```
package gov.nasa.jpl.imce.sysmlnxsync.actions;

import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.ui.ConsistencyReportDialog;
import gov.nasa.jpl.imce.sysmlnxsync.ui.WaitDialog;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.ConsistencyResult;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.DefaultNodeHandler;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;

import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashSet;
import java.util.List;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.ui.browser.actions.DefaultBrowserActi
on;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     public class ValidateAgainstNXPart extends
DefaultBrowserAction {

          public class ValidateResolver extends
DefaultNodeHandler {
               private ArrayList<ConsistencyResult>
_consistencyReport;
               private boolean _result;
               private HashSet<String> _uniqueID;

               public ValidateResolver() {
                    _consistencyReport = new
ArrayList<ConsistencyResult>();
                    _uniqueID = new HashSet<String>();
                    _result = true;
               }

               @Override
               public Property enterExpression(Project
project, Class parent,
                         Property sysmlExpression,
NXExpression nxExpression) {

                    String sysmlName = (sysmlExpression !=
null ? sysmlExpression.getName() : null);
                    if (nxExpression != null) {
                         String nxName =
nxExpression.getName();
                         if (sysmlExpression != null) {
                              if
(!nxName.equals(sysmlExpression.getName())) {
                                   fail("Expression",
sysmlName, String.format("NX parameter name %s differs from
SysML parameter name", nxName) );
                              }
                         } else {
                              fail("Expression", nxName,
"No SysML equivalent for NX parameter" );
                         }
                    } else if (sysmlExpression != null) {
```

```
                        fail("Expression", sysmlName, "No
NX equivalent for SysML parameter" );
                    }

                    if (sysmlExpression != null &&
nxExpression != null) {
                        ValueSpecification spec =
sysmlExpression.getDefaultValue();
                        String sysmlValue = (spec
instanceof LiteralString ? ((LiteralString)spec).getValue()
: null );
                        if (sysmlValue != null) {
                            if
(!sysmlValue.equals(nxExpression.getValue())) {
                                fail("Expression",
sysmlName, String.format("Value conflict: NX value %s
differs from SysML value %s", nxExpression.getValue(),
sysmlValue) );
                            }
                        } else {
                            fail("Expression",
nxExpression.getName(), "SysML parameter value cannot be
read");
                        }
                    }

                    return sysmlExpression;
                }

                @Override
                public Class enterFeature(Project project,
Class parent, Class sysmlFeature, NXFeature nxFeature) {
                    Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");
                    String sysmlName = (sysmlFeature !=
null ? sysmlFeature.getName() : null);
                    if (nxFeature != null) {
                        String nxName =
nxFeature.getName();
                        if (sysmlFeature != null) {
                            if
(!nxName.equals(sysmlFeature.getName())) {
                                fail("Feature",
sysmlName, String.format("NX feature name %s differs from
SysML feature name", nxName) );
                            }
                        } else {
```

```
                                fail("Feature", nxName, "No
SysML equivalent for NX feature" );
                            }
                        } else if (sysmlFeature != null) {
                            fail("Feature", sysmlName, "No NX
equivalent for SysML feature" );
                        }

                        //String sysmlFeatureName =
sysmlFeature.getName();
                        //String nxFeatureName =
nxFeature.getName();
                        //if
(!sysmlFeatureName.equals(nxFeatureName)) {
                        //   consistencyResult = new
ConsistencyResult( false, String.format("Feature name %s
differs from feature name %s", nxFeatureName,
sysmlFeatureName) );
                        //   return sysmlFeature;
                        //}

                        if (sysmlFeature != null && nxFeature
!= null) {
                            String nxFType =
nxFeature.getType();
                            if
(StereotypesHelper.hasStereotype(sysmlFeature,
nxFeatureStereotype) &&

     StereotypesHelper.getStereotypePropertyValue(sysmlFeat
ure, nxFeatureStereotype, "featureType") != null) {
                                Object childFtype =
StereotypesHelper.getStereotypePropertyFirst(sysmlFeature,
nxFeatureStereotype, "featureType");
                                String sysmlFType =
childFtype.toString();
                                if
(!sysmlFType.equals(nxFType)) {
                                    fail("Feature",
sysmlName, String.format("NX feature type %s differs from
SysML feature type %s", nxFType, sysmlFType) );
                                    return sysmlFeature;
                                }
                            } else {
                                fail("Feature", sysmlName,
String.format("NX feature type is %s, SysML has no feature
type %s", nxFType) );
```

```
                        }

                        Collection<NXExpression>
nxExpressions = nxFeature.getExpressions();
                        Collection<Property>
sysmlExpressions = SysMLUtility.getExpressions(project,
sysmlFeature);

                        if (nxExpressions.size() !=
sysmlExpressions.size()) {
                             fail("Feature", sysmlName,
String.format("%d parameters in NX, %d parameters in
SysML", nxExpressions.size(), sysmlExpressions.size()) );
                             return sysmlFeature;
                        }

                        Collection<NXFeature>
nxSubfeatures = nxFeature.getChildren();
                        Collection<Class> sysmlSubfeatures
= SysMLUtility.getFeatures(project, sysmlFeature);

                        if (nxSubfeatures.size() !=
sysmlSubfeatures.size()) {
                             fail("Feature", sysmlName,
String.format("%d features in NX, %d features in SysML",
nxSubfeatures.size(), sysmlSubfeatures.size()) );
                        }
                   }

                   return sysmlFeature;
              }

              @Override
              public Class enterPart(Project project,
Class parent, Class sysmlPart, NXPart nxPart) {
                   //ElementsFactory elementsFactory =
project.getElementsFactory();
     //             Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(project, "NXPart");
                   String sysmlName = (sysmlPart != null ?
sysmlPart.getName() : null);
                   if (nxPart != null) {
                        String nxName = nxPart.getName();
                        if (sysmlPart != null) {
                             if
(!nxName.equals(sysmlPart.getName())) {
```

```
                                        fail("Part", sysmlName,
String.format("NX part name %s differs from SysML part name
%s", nxName) );
                                   }
                              } else {
                                   fail("Part", nxName, "No
SysML equivalent for NX part");
                              }
                         } else if (sysmlPart != null) {
                              fail("Part", sysmlName, "No NX
equivalent for SysML part" );
                         }
                         return sysmlPart;
                    }

                    private void fail(String type, String id,
String report) {
                         String s = String.format("%s||%s||%s",
type, id, report);
                         if (!_uniqueID.contains(s)) {
                              _consistencyReport.add( new
ConsistencyResult( type, id, report, null ) );
                              _uniqueID.add(s);
                         }
                         _result = false;
                    }

                    public List<ConsistencyResult>
getInconsistencyList() {
                         return _consistencyReport;
                    }

                    public boolean getResult() {
                         return _result;
                    }
               }

               ElementsFactory elementsFactory =
Application.getInstance().getProject().getElementsFactory()
;

               /**
                * Constructor - configures the action, in this
case our menu item
                */
               public ValidateAgainstNXPart() {
```

```
                // In the containment browser context pop
the text that will be displayed in the menu item
                // is what we specify as the second
argument. The first argument is an ID
                super("Validate_NX_Part_Maple", "Execute CAD
- SysML Consistency Analysis", null, null);

          }

          /**
           * This function (or action) will be fired
whenever a user clicks on the menu item that we are
           * describing in this class. I.e. whenever
someone right clicks in the containment browser and
           * selects our action, in this case "Import NX
Part ...", this function will be called
           */
          @Override
          public void actionPerformed(ActionEvent
actionEvent) {
                super.actionPerformed(actionEvent);

                Object userObject = getSelectedObject();
                if (!(userObject instanceof Class)) {
                     return;
                }
                Class userClass = (Class)userObject;
                Project project =
Application.getInstance().getProject();
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                if
(!(StereotypesHelper.hasStereotype(userClass,
nxPartStereotype))) {
                     return;
                }

                Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
                WaitDialog waitDialog = new
WaitDialog(parentFrame, "Checking consistency...", "System
is working");
                waitDialog.setVisible(true);
                //ProgressMonitor pm = new
ProgressMonitor(parentFrame, JOptionPane.PLAIN_MESSAGE,
"Please wait", 0, 10);
```

```
                String currentPartPath =
StereotypesHelper.getStereotypePropertyFirst(userClass,
nxPartStereotype, "currentPartPath").toString();
                File file = new File( currentPartPath );
                ValidateResolver resolver = new
ValidateResolver();

                NXPart part =
SysMLUtility.openPart(parentFrame, file);
                SysMLModelTraverser.launch( project,
userClass, part, resolver );

                List<ConsistencyResult> res =
resolver.getInconsistencyList();
                if (res == null) { res = new
ArrayList<ConsistencyResult>(); };

                waitDialog.setVisible(false);

                ConsistencyReportDialog consistencyReport =
new ConsistencyReportDialog(parentFrame, res);
                consistencyReport.setVisible(true);
          }

     }
```

Model:

ExpressionModel:

```
package gov.nasa.jpl.imce.sysmlnxsync.model;

import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;

public class ExpressionModel {
     private Property _sysmlProperty;
     private LiteralString _instanceValue;

     public ExpressionModel(Property sysmlProperty,
LiteralString instanceValue) {
          _sysmlProperty = sysmlProperty;
          _instanceValue = instanceValue;
     }

     public Property getBlock() {
          return _sysmlProperty;
     }

     public LiteralString getInstance() {
          return _instanceValue;
     }
}
```

FeatureModel:

```
package gov.nasa.jpl.imce.sysmlnxsync.model;

import java.util.LinkedList;

import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;

public class FeatureModel {
     private Class _sysmlClass;
     private Package _instancePackage;
     private LinkedList<FeatureModel> _subFeatures;

     public FeatureModel(Class sysmlClass, Package
instancePackage) {
          _sysmlClass = sysmlClass;
          _instancePackage = instancePackage;
          _subFeatures = new
LinkedList<FeatureModel>();
     }

     public void addFeature(FeatureModel fm) {
          _subFeatures.add(fm);
     }

     public Class getBlock() {
          return _sysmlClass;
     }

     public FeatureModel getFeatures(int i) {
          return _subFeatures.get(i);
     }

     public Package getInstancePackage() {
          return _instancePackage;
     }

}
```

PartModel:

```
package gov.nasa.jpl.imce.sysmlnxsync.model;

import java.util.LinkedList;

import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;

public class PartModel {
     private Class _sysmlClass;
     private LinkedList<PartModel> _subParts;
     private LinkedList<FeatureModel> _subFeatures;

     public PartModel(Class sysmlClass) {
          _sysmlClass = sysmlClass;
          _subParts = new LinkedList<PartModel>();
          _subFeatures = new
LinkedList<FeatureModel>();
     }

     public void addFeature(FeatureModel fm) {
          _subFeatures.add(fm);
     }

     public void addPart(PartModel pm) {
          _subParts.add(pm);
     }

     public Class getBlock() {
          return _sysmlClass;
     }

     public FeatureModel getFeatures(int i) {
          return _subFeatures.get(i);
     }

     public PartModel getPart(int i) {
          return _subParts.get(i);
     }
}
```

UserInterface:

ConsistencyReportDialog:

```
package gov.nasa.jpl.imce.sysmlnxsync.ui;

import
gov.nasa.jpl.imce.sysmlnxsync.utility.ConsistencyResult;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.List;

import javax.swing.AbstractCellEditor;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
import javax.swing.WindowConstants;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableCellEditor;

public class ConsistencyReportDialog extends JDialog
implements ActionListener {
      static class ConsistencyTableModel extends
AbstractTableModel {

                  private List<ConsistencyResult>
_consistencyResult;

            public ConsistencyTableModel(
List<ConsistencyResult> consistencyResult ) {
                  _consistencyResult = consistencyResult;
            }

            @Override
            public int getColumnCount() {
                  return 3;
            }

            @Override
```

```
                public String getColumnName(int column) {
                     switch(column) {
                     case 0:
                          return "Type";
                     case 1:
                          return "Name";
                     case 2:
                          return "Message";
                     default:
                          return
super.getColumnName(column);
                     }
                }

                @Override
                public int getRowCount() {
                     return _consistencyResult.size();
                }


                @Override
                public Object getValueAt(int rowIndex, int
columnIndex) {
                     if (rowIndex <
_consistencyResult.size()) {
                          switch(columnIndex) {
                          case 0:
                               return
_consistencyResult.get(rowIndex).getType();
                          case 1:
                               return
_consistencyResult.get(rowIndex).getIdentifier();
                          case 2:
                               return
_consistencyResult.get(rowIndex).getMessage();
                          default:
                               return "";
                          }
                     } else {
                          return "";
                     }
                }

          }
          static private class MyCellEditor extends
AbstractCellEditor implements TableCellEditor {
                private final JTextArea _ta;
```

```
                public MyCellEditor() {
                     _ta = new JTextArea();
                     _ta.setEditable(false);
                }

                @Override
                public Object getCellEditorValue() {
                     return _ta.getText();
                }

                @Override
                public Component
getTableCellEditorComponent(JTable table,
                          Object value, boolean isSelected,
int row, int column) {
                     _ta.setText(value.toString());
                     return _ta;
                }
           }
           /**
            *
            */
           private static final long serialVersionUID =
1140107010316106563L;
           private JButton _ok;

           private JTable _report;

           private JLabel _result;


           public ConsistencyReportDialog(Frame parent,
List<ConsistencyResult> consistencyResult) {
                super(parent, "Consistency Report");
                Box vbox = Box.createVerticalBox();

                Box hbox = Box.createHorizontalBox();
                JLabel label = new JLabel(
(consistencyResult.isEmpty() ? "Model element is
consistent" :  "Model element is inconsistent" ) );

                hbox.add(label);

                vbox.add(Box.createVerticalStrut(10));
                vbox.add(hbox);
```

```
              if (!consistencyResult.isEmpty()) {
                   _report = new JTable( new
ConsistencyTableModel(consistencyResult) );
                   JScrollPane scroll = new
JScrollPane(_report);

     scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.
VERTICAL_SCROLLBAR_ALWAYS);
                   scroll.setPreferredSize( new
Dimension(450, 250) );
                   vbox.add(Box.createVerticalStrut(10));
                   vbox.add(scroll);


     _report.getColumnModel().getColumn(0).setPreferredWidt
h(40);

     _report.getColumnModel().getColumn(1).setPreferredWidt
h(60);

     _report.getColumnModel().getColumn(2).setPreferredWidt
h(400);


     _report.getColumnModel().getColumn(2).setCellEditor(
new MyCellEditor() );
              }

            _ok = new JButton("OK");

              vbox.add(Box.createVerticalStrut(10));
              vbox.add(_ok);
              vbox.add(Box.createVerticalStrut(10));

              add( vbox );

              setMinimumSize( new Dimension(500, 300));

              setLocationRelativeTo( parent );

              invalidate();
              setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
              pack();

              _ok.addActionListener(this);
         }
```

```
        @Override
        public void actionPerformed(ActionEvent ae) {
             Object target = ae.getSource();
             if (target == _ok) {
                  setVisible(false);
                  dispose();
             }
        }
}
```

InstanceReportDialog:

```
package gov.nasa.jpl.imce.sysmlnxsync.ui;

import
gov.nasa.jpl.imce.sysmlnxsync.utility.InstanceReportResult;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import javax.swing.AbstractCellEditor;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
import javax.swing.WindowConstants;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableCellEditor;

public class InstanceReportDialog extends JDialog
implements ActionListener {
        private List<InstanceReportResult> _resultList;
        private List<InstanceReportResult>
_filteredResultList;

        private static final long serialVersionUID =
1140107010316106563L;
        private JButton _ok;

        private AbstractTableModel _model;
        private JTable _report;
        private JLabel _result;

        private JCheckBox _filterStructuralValues,
_filterValidationValues, _filterTargetValues,
_filterPerformanceValues;
```

```
          class InstanceReportTableModel extends
AbstractTableModel {

               @Override
               public int getColumnCount() {
                    return 5;
               }

               @Override
               public String getColumnName(int column) {
                    switch(column) {
                    case 0:
                         return "Parameter Name";
                    case 1:
                         return "Feature Name";
                    case 2:
                         return "Part Name";
                    case 3:
                         return "Value Type";
                    case 4:
                         return "Value";
                    default:
                         return
super.getColumnName(column);
                    }
               }

               @Override
               public int getRowCount() {
                    return _filteredResultList.size();
               }


               @Override
               public Object getValueAt(int rowIndex, int
columnIndex) {
                    if (rowIndex <
_filteredResultList.size()) {
                         InstanceReportResult res =
_filteredResultList.get(rowIndex);
                         switch(columnIndex) {
                         case 0:
                              return res.getName();
                         case 1:
                              return res.getFeatureName();
                         case 2:
```

```
                              return res.getPartName();
                         case 3:
                              return res.getType();
                         case 4:
                              return res.getValue();
                         default:
                              return "";
                         }
                    } else {
                         return "";
                    }
               }

          }
          static private class MyCellEditor extends
AbstractCellEditor implements TableCellEditor {
               private final JTextArea _ta;

               public MyCellEditor() {
                    _ta = new JTextArea();
                    _ta.setEditable(false);
               }

               @Override
               public Object getCellEditorValue() {
                    return _ta.getText();
               }

               @Override
               public Component
getTableCellEditorComponent(JTable table,
                         Object value, boolean isSelected,
int row, int column) {
                    _ta.setText(value.toString());
                    return _ta;
               }
          }
          /**
           *
           */

          public List<InstanceReportResult>
getFilteredList(List<InstanceReportResult> result) {
               ArrayList<InstanceReportResult> arr = new
ArrayList<InstanceReportResult>();
               for (InstanceReportResult row : result) {
```

```
                    if ("Part
Property".equals(row.getType()) || "Constraint
Parameter".equals(row.getType()) || "Constraint
Property".equals(row.getType())) {
                        // Ignore and never include in
result list
                    } else if
("NXValueProperty".equals(row.getType()) || "Value
Property".equals(row.getType())) {
                        if
(_filterStructuralValues.isSelected()) arr.add(row);
                    } else if ("Validation Value
Property".equals(row.getType())) {
                        if
(_filterValidationValues.isSelected()) arr.add(row);
                    } else if ("Performance Value
Property".equals(row.getType())) {
                        if
(_filterPerformanceValues.isSelected()) arr.add(row);
                    } else if ("Targert Value
Property".equals(row.getType())) {
                        if
(_filterTargetValues.isSelected()) arr.add(row);
                    } else {
                        arr.add(row);
                    }
                }
                Comparator<InstanceReportResult> cmp = new
Comparator<InstanceReportResult>() {
                    public int compare(InstanceReportResult
c1, InstanceReportResult c2) {
                        return
c1.getName().compareTo(c2.getName());
                    }
                };
                Collections.sort(arr, cmp );
                return arr;
            }

            public InstanceReportDialog(Frame parent,
List<InstanceReportResult> result) {
                super(parent, "Instance Results Report");

                _filterStructuralValues  = new
JCheckBox("Structural Values", true);
                _filterValidationValues  = new
JCheckBox("Validation Values", true);
```

```
                _filterTargetValues      = new
JCheckBox("Target Values", true);
                _filterPerformanceValues = new
JCheckBox("Performance Values", true);

                Box vbox = Box.createVerticalBox();
                Box hbox = Box.createHorizontalBox();

                JLabel label = new JLabel( "View:" );
                hbox.add(label);
                hbox.add(_filterStructuralValues);
                hbox.add(_filterValidationValues );
                hbox.add(_filterTargetValues);
                hbox.add(_filterPerformanceValues);
                vbox.add(hbox);

                vbox.add(Box.createVerticalStrut(10));

                _resultList = result;
                _filteredResultList = getFilteredList(
_resultList );

                _model = new InstanceReportTableModel();
                _report = new JTable( _model );
                JScrollPane scroll = new
JScrollPane(_report);

     scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.
VERTICAL_SCROLLBAR_ALWAYS);
                scroll.setPreferredSize( new Dimension(600,
250) );
                vbox.add(Box.createVerticalStrut(10));
                vbox.add(scroll);


     _report.getColumnModel().getColumn(0).setPreferredWidt
h(125);

     _report.getColumnModel().getColumn(1).setPreferredWidt
h(125);

     _report.getColumnModel().getColumn(2).setPreferredWidt
h(125);

     _report.getColumnModel().getColumn(3).setPreferredWidt
h(150);
```

```
     _report.getColumnModel().getColumn(4).setPreferredWidt
h(125);


     _report.getColumnModel().getColumn(0).setCellEditor(
new MyCellEditor() );

     _report.getColumnModel().getColumn(1).setCellEditor(
new MyCellEditor() );

     _report.getColumnModel().getColumn(2).setCellEditor(
new MyCellEditor() );

     _report.getColumnModel().getColumn(3).setCellEditor(
new MyCellEditor() );

     _report.getColumnModel().getColumn(4).setCellEditor(
new MyCellEditor() );

             _ok = new JButton("OK");

               vbox.add(Box.createVerticalStrut(10));
               vbox.add(_ok);
               vbox.add(Box.createVerticalStrut(10));

               add( vbox );

               setMinimumSize( new Dimension(600, 300));

               setLocationRelativeTo( parent );

               invalidate();
               setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
               pack();


     _filterStructuralValues.addActionListener(this);

     _filterValidationValues.addActionListener(this);
               _filterTargetValues.addActionListener(this);

     _filterPerformanceValues.addActionListener(this);
               _ok.addActionListener(this);
          }
```

```
          @Override
          public void actionPerformed(ActionEvent ae) {
               Object target = ae.getSource();
               if (target == _ok) {
                    setVisible(false);
                    dispose();
               } else if (target == _filterStructuralValues
|| target == _filterValidationValues ||
                                   target == _filterTargetValues
|| target == _filterPerformanceValues) {
                    _filteredResultList = getFilteredList(
_resultList );
                    _model.fireTableDataChanged();
                    _report.invalidate();
               }
          }
     }
```

InteractiveConsistencyReportDialog:

```
package gov.nasa.jpl.imce.sysmlnxsync.ui;

import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXClientEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXConnectionExce
ption;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.ConsistencyResult;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.DefaultNodeHandler;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLModelTraverser;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateFromNXResolver;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.UpdateToNXResolver;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.util.ArrayList;
import java.util.List;

import javax.swing.AbstractCellEditor;
import javax.swing.Box;
import javax.swing.DefaultCellEditor;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
import javax.swing.WindowConstants;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellEditor;
import javax.swing.table.TableCellRenderer;
```

```
     import javax.swing.table.TableColumn;

     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;

     public class InteractiveConsistencyReportDialog
extends JDialog implements ActionListener {
          static private class ComboBoxCellEditor extends
DefaultCellEditor {
               public ComboBoxCellEditor(String[] items) {
                    super( new JComboBox(items) );
               }
          }
          static private class ComboBoxCellRenderer extends
JComboBox implements TableCellRenderer {
               private List<ConsistencyResult> _cr;
               public
ComboBoxCellRenderer(List<ConsistencyResult> cr) {
                    super();
                    _cr = cr;
               }

              @Override
               public Component
getTableCellRendererComponent(JTable table, Object value,
                      boolean isSelected, boolean hasFocus,
int row, int column) {

               super.removeAllItems();
               if (_cr.get(row).canUpdateNX())
super.addItem("NX");
               if (_cr.get(row).canUpdateSysML())
super.addItem("SysML");

               if (isSelected) {

setForeground(table.getSelectionForeground());

super.setBackground(table.getSelectionBackground());
                  } else {
                      setForeground(table.getForeground());
                      setBackground(table.getBackground());
                  }

                  // Select the current value
                  setSelectedItem(value);
```

```
                setEnabled(true);
                return this;
            }
        }
        static class ConsistencyTableModel extends
DefaultTableModel {

                        private List<ConsistencyResult>
_consistencyResult;
                        private List<Boolean> _res;

            public ConsistencyTableModel(
List<ConsistencyResult> consistencyResult ) {
                super(new
String[]{"Type","Name","Message","Resolve Using"},
consistencyResult.size());
                _consistencyResult = consistencyResult;
                _res = new ArrayList<Boolean>();
            }

            @Override
            public Object getValueAt(int rowIndex, int
columnIndex) {
                if (rowIndex <
_consistencyResult.size()) {
                        switch(columnIndex) {
                        case 0:
                            return
_consistencyResult.get(rowIndex).getType();
                        case 1:
                            return
_consistencyResult.get(rowIndex).getIdentifier();
                        case 2:
                            return
_consistencyResult.get(rowIndex).getMessage();
                        case 3:
                        default:
                            return
super.getValueAt(rowIndex, columnIndex);
                        }
                } else {
                        return super.getValueAt(rowIndex,
columnIndex);
                }
            }

        }
```

```
          static private class MyCellEditor extends
AbstractCellEditor implements TableCellEditor {
               private final JTextArea _ta;

               public MyCellEditor() {
                    _ta = new JTextArea();
                    _ta.setEditable(false);
               }

               @Override
               public Object getCellEditorValue() {
                    return _ta.getText();
               }

               @Override
               public Component
getTableCellEditorComponent(JTable table,
                         Object value, boolean isSelected,
int row, int column) {
                    _ta.setText(value.toString());
                    return _ta;
               }
          }
          /**
           *
           */
          private static final long serialVersionUID =
1140107010316106563L;
          private File _nxFile;
          private NXPart _part;
          private Project _project;

          private JTable _report;

          private JButton _resolveNX, _resolveSysML,
_resolve, _cancel;

          private JLabel _result;

          private Class _userClass;


          public InteractiveConsistencyReportDialog(Frame
parent,
                    List<ConsistencyResult>
consistencyResult,
```

```
                    Project project, Class userClass, File
nxFile, NXPart part) {
               super(parent, "Resolve Model");
               _project = project;
               _userClass = userClass;
               _part = part;
               _nxFile = nxFile;
               Box vbox = Box.createVerticalBox();

               Box hbox = Box.createHorizontalBox();
               JLabel label = new JLabel(
(consistencyResult.isEmpty() ? "Model element is
consistent" :  "Model element is inconsistent" ) );

               hbox.add(label);

               vbox.add(Box.createVerticalStrut(10));
               vbox.add(hbox);

               if (!consistencyResult.isEmpty()) {
                    _report = new JTable( new
ConsistencyTableModel(consistencyResult) );
                    JScrollPane scroll = new
JScrollPane(_report);

     scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.
VERTICAL_SCROLLBAR_ALWAYS);
                    scroll.setPreferredSize( new
Dimension(700, 300) );
                    vbox.add(Box.createVerticalStrut(10));
                    vbox.add(scroll);

                    TableColumn col =
_report.getColumnModel().getColumn(0);
                    col.setCellEditor( new MyCellEditor()
);
                    col.setPreferredWidth(100);

                    col =
_report.getColumnModel().getColumn(1);
                    col.setCellEditor( new MyCellEditor()
);
                    col.setPreferredWidth(60);

                    col =
_report.getColumnModel().getColumn(2);
```

```
                    col.setCellEditor( new MyCellEditor()
);
                    col.setPreferredWidth(400);

                    col =
_report.getColumnModel().getColumn(3);
                    ComboBoxCellRenderer cbcr = new
ComboBoxCellRenderer(consistencyResult);
                    ComboBoxCellEditor cbce = new
ComboBoxCellEditor( new String[]{"NX", "SysML"});
                    col.setCellEditor( cbce );
                    cbcr.setSelectedItem("NX");
                    col.setCellRenderer( cbcr );
                    col.setPreferredWidth(140);
                }

                hbox = Box.createHorizontalBox();
              _resolve = new JButton("Resolve Using Above
Settings");
              _resolveNX = new JButton("Resolve using CAD");
              _resolveSysML = new JButton("Resolve using
SysML");
              _cancel = new JButton("Cancel");
              hbox.add(_resolve);
              hbox.add(Box.createHorizontalStrut(10));
              hbox.add(_resolveNX);
              hbox.add(Box.createHorizontalStrut(10));
              hbox.add(_resolveSysML);
              hbox.add(Box.createHorizontalStrut(10));
              hbox.add(_cancel);

                vbox.add(Box.createVerticalStrut(10));
                vbox.add(hbox);
                vbox.add(Box.createVerticalStrut(10));

                add( vbox );

                setMinimumSize( new Dimension(500, 300));

                setLocationRelativeTo( parent );

                invalidate();
                setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
              pack();

              _cancel.addActionListener(this);
```

```
            _resolve.addActionListener(this);
            _resolveNX.addActionListener(this);
            _resolveSysML.addActionListener(this);
        }

        @Override
        public void actionPerformed(ActionEvent ae) {

            NXEngine engine;
            try {
                engine = new NXClientEngine();
            } catch (NXConnectionException nxce) {
                engine = null;
            }
            if (engine == null ) { return; }

            Object target = ae.getSource();
            if (target == _cancel) {
                setVisible(false);
                dispose();
            } else if (target == _resolve || target ==
_resolveSysML) {
                DefaultNodeHandler resolver = new
UpdateToNXResolver(engine);
                SysMLModelTraverser.launch( _project,
_userClass, _part, resolver );
                setVisible(false);
                dispose();

            } else if (target == _resolveNX) {

    //              Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
    //              ProgressMonitor pm = new
ProgressMonitor(parentFrame, JOptionPane.PLAIN_MESSAGE,
"Please wait", 0, 10);

                DefaultNodeHandler resolver = new
UpdateFromNXResolver(null);
                SysMLModelTraverser.launch( _project,
_userClass, _part, resolver );
                resolver = null;

                setVisible(false);
                dispose();

            } else if (target == _resolveSysML) {
```

```
                    DefaultNodeHandler resolver = new
UpdateToNXResolver(engine);
                    SysMLModelTraverser.launch( _project,
_userClass, _part, resolver );
                    resolver = null;

                    setVisible(false);
                    dispose();
                }
            }
        }
```

InteractiveInstanceReportDialog:

```
package gov.nasa.jpl.imce.sysmlnxsync.ui;

import
gov.nasa.jpl.imce.sysmlnxsync.utility.InstanceReportResult;
import
gov.nasa.jpl.imce.sysmlnxsync.utility.InteractiveInstanceRe
portResult;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;

import javax.swing.AbstractCellEditor;
import javax.swing.Box;
import javax.swing.DefaultCellEditor;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
import javax.swing.WindowConstants;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.DefaultTableModel;
import javax.swing.table.TableCellEditor;
import javax.swing.table.TableColumnModel;
import javax.swing.table.TableModel;

import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.StructuralF
eature;

public class InteractiveInstanceReportDialog extends
JDialog implements ActionListener {
```

```
          private List<InteractiveInstanceReportResult>
_resultList;
          private ArrayList<JComboBox> _valueSelector;
          private ArrayList<TableCellEditor> _editors;

          class MyTable extends JTable {

               public MyTable (DefaultTableModel tm) {
                    super(tm);
               }

               /*@Override
               public TableCellEditor getCellEditor(int
row, int column) {
                    int modelColumn =
convertColumnIndexToModel( column );
                    if (modelColumn == 3) {
                         return _editors.get(row);
                    }
                    return super.getCellEditor(row,
column);
               }*/

          }

          class InteractiveInstanceReportTableModel extends
DefaultTableModel {

               @Override
               public int getColumnCount() {
                    return 5;
               }

               @Override
               public String getColumnName(int column) {
                    switch(column) {
                    case 0:
                         return "Parameter Name";
                    case 1:
                         return "Feature Name";
                    case 2:
                         return "Part Name";
                    case 3:
                         return "Default Value";
                    case 4:
                         return "Value";
                    default:
```

```
                    return
super.getColumnName(column);
               }
          }

          @Override
          public int getRowCount() {
               return _resultList.size();
          }


          @Override
          public Object getValueAt(int rowIndex, int
columnIndex) {
               if (rowIndex < _resultList.size()) {
                    InteractiveInstanceReportResult
res = _resultList.get(rowIndex);
                    switch(columnIndex) {
                    case 0:
                         return res.getName();
                    case 1:
                         return res.getFeatureName();
                    case 2:
                         return res.getPartName();
                    case 3:
                         return res.getDefaultValue();
                    case 4:
                         return res.getValue().get(0);
                    default:
                         return
super.getValueAt(rowIndex, columnIndex);
                    }
               } else {
                    return super.getValueAt(rowIndex,
columnIndex);
               }
          }

     }

     static private class MyCellEditor extends
AbstractCellEditor implements TableCellEditor {
          private final JTextArea _ta;

          public MyCellEditor() {
               _ta = new JTextArea();
               _ta.setEditable(false);
```

```
                }

                @Override
                public Object getCellEditorValue() {
                     return _ta.getText();
                }

                @Override
                public Component
getTableCellEditorComponent(JTable table,
                          Object value, boolean isSelected,
int row, int column) {
                     _ta.setText(value.toString());
                     return _ta;
                }
           }

           public class CustomComboBoxEditor extends
DefaultCellEditor implements TableCellEditor {

                // Decalre a model that is used for adding
the elements to the `Combo box`
                private DefaultComboBoxModel _cbModel;

                public CustomComboBoxEditor() {
                     super(new JComboBox());
                     _cbModel =
(DefaultComboBoxModel)((JComboBox)getComponent()).getModel(
);
                }

                @Override
                public Component
getTableCellEditorComponent(JTable table, Object value,
boolean isSelected, int row, int column) {
                    // Add the elements which you want to
the model.
                    // Here I am adding elements from the
orderList(say).
                     TableModel model = table.getModel();

                     _cbModel.removeAllElements();
                     List<String> valList =
_resultList.get(row).getValue();
                     for (String val : valList) {
                          _cbModel.addElement(val);
                     }
```

```
                    Object val = table.getValueAt(row,
column);
                    _cbModel.setSelectedItem(val);

                   //model.setValueAt(valList.get(0), row,
column);

                    //finally return the component.
                    return
super.getTableCellEditorComponent(table, value, isSelected,
row, column);
               }

          }

          /**
           *
           */
          private static final long serialVersionUID =
1140107010316106563L;
          private JButton _cancelButton, _updateButton;

          private JTable _report;
          private JLabel _result;

          public InteractiveInstanceReportDialog(Frame
parent, List<InteractiveInstanceReportResult> resultList) {
               super(parent, "Update Block Value Properties
from Instance");

               _resultList = resultList;
               _valueSelector = new ArrayList<JComboBox>();
               _editors = new ArrayList<TableCellEditor>();
               JComboBox cb;

               for (InteractiveInstanceReportResult result
: resultList) {
                    cb = new
JComboBox(result.getValue().toArray());
                    _valueSelector.add(cb);
                    _editors.add( new DefaultCellEditor(cb)
);
               }

               Box vbox = Box.createVerticalBox();
               Box hbox = Box.createHorizontalBox();
```

```
                DefaultTableModel tm = new
InteractiveInstanceReportTableModel();
                _report = new MyTable(tm);
                TableColumnModel cm =
_report.getColumnModel();

                JScrollPane scroll = new
JScrollPane(_report);

     scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.
VERTICAL_SCROLLBAR_ALWAYS);
                scroll.setPreferredSize( new Dimension(600,
250) );
                vbox.add(Box.createVerticalStrut(10));
                vbox.add(scroll);


                cm.getColumn(0).setPreferredWidth(120);
                cm.getColumn(1).setPreferredWidth(120);
                cm.getColumn(2).setPreferredWidth(120);
                cm.getColumn(3).setPreferredWidth(120);
                cm.getColumn(4).setPreferredWidth(120);

                cm.getColumn(1).setCellEditor( new
MyCellEditor() );
                cm.getColumn(2).setCellEditor( new
MyCellEditor() );
                cm.getColumn(3).setCellEditor( new
MyCellEditor() );
                cm.getColumn(4).setCellEditor(new
CustomComboBoxEditor());

                _cancelButton = new JButton("Cancel");
              _updateButton = new JButton("Update");

                vbox.add(Box.createVerticalStrut(10));

                hbox = Box.createHorizontalBox();
                hbox.add(_cancelButton);
                hbox.add(Box.createHorizontalStrut(10));
                hbox.add(_updateButton);
                vbox.add(hbox);

                vbox.add(Box.createVerticalStrut(10));

                add( vbox );
```

```
                setMinimumSize( new Dimension(500, 300));

                setLocationRelativeTo( parent );

                invalidate();
                setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
                pack();

                _updateButton.addActionListener(this);
           }

           @Override
           public void actionPerformed(ActionEvent ae) {
                Object target = ae.getSource();
                if (target == _updateButton) {
                     setVisible(false);
                     dispose();
                } else if (target == _cancelButton) {
                     setVisible(false);
                     dispose();
                }
           }
      }
```

StereotypeFilterDialog:

```
package gov.nasa.jpl.imce.sysmlnxsync.ui;

import
gov.nasa.jpl.imce.sysmlnxsync.utility.SysMLUtility;

import java.awt.Component;
import java.awt.Dimension;
import java.awt.Frame;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

import javax.swing.AbstractCellEditor;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
import javax.swing.WindowConstants;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableCellEditor;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

public class StereotypeFilterDialog extends JDialog
implements ActionListener {

      static private class MyCellEditor extends
AbstractCellEditor implements TableCellEditor {
            private final JTextArea _ta;

            public MyCellEditor() {
                  _ta = new JTextArea();
                  _ta.setEditable(false);
            }

            @Override
```

```
                public Object getCellEditorValue() {
                     return _ta.getText();
                }

                @Override
                public Component
getTableCellEditorComponent(JTable table,
                          Object value, boolean isSelected,
int row, int column) {
                     _ta.setText(value.toString());
                     return _ta;
                }
           }
           static class StereotypeTableModel extends
AbstractTableModel {
                private List<Stereotype>
_excludedStereotypes;
                private List<Stereotype>
_profileStereotypes;

                public StereotypeTableModel(
Collection<Stereotype> profileStereotypes ) {
                     _profileStereotypes = new
ArrayList<Stereotype>( profileStereotypes );
                     Collections.sort( _profileStereotypes,
new Comparator<Stereotype>() {
                       @Override
                          public int compare(Stereotype st1,
Stereotype st2) {
                           return
st1.getName().compareTo(st2.getName());
                       }
                   });
                     // list of excluded stereotypes is
initially empty
                     _excludedStereotypes = new
ArrayList<Stereotype>();
                }

                @Override
                public Class<?> getColumnClass(int
columnIndex) {
                     if (columnIndex == 0) {
                          return Boolean.class;
                     }
                     return
super.getColumnClass(columnIndex);
```

```
                }

                @Override
                public int getColumnCount() {
                     return 2;
                }

                @Override
                public String getColumnName(int column) {
                     switch(column) {
                     case 0:
                          return "Include?";
                     default:
                     case 1:
                          return "Name";
                     }
                }

                public Collection<Stereotype>
getExcludedStereotypes() {
                     return _excludedStereotypes;
                }

                @Override
                public int getRowCount() {
                     return _profileStereotypes.size();
                }


                @Override
                public Object getValueAt(int rowIndex, int
columnIndex) {
                     Stereotype st =
_profileStereotypes.get(rowIndex);
                     switch(columnIndex) {
                     case 0:
                          return
!_excludedStereotypes.contains(st);
                     default:
                     case 1:
                          return st.getName();
                     }
                }

                @Override
                public boolean isCellEditable(int rowIndex,
int columnIndex) {
```

```
     //             if (columnIndex == 0) {
     //                  String sn =
_profileStereotypes.get(rowIndex).getName();
     //                  if ("NXSketch".equals(sn) ||
"NXCoordinateSystem".equals(sn)) {
     //                       return true;
     //                  }
     //             }
                    return false;
                    return true;
               }

               @Override
               public void setValueAt(Object aValue, int
rowIndex, int columnIndex) {
                    Stereotype st1 =
_profileStereotypes.get(rowIndex);
                    if (columnIndex == 0) {
                    Stereotype st =
_profileStereotypes.get(rowIndex);
                    Boolean checked = (Boolean)aValue;
                    if (checked &&
_excludedStereotypes.contains(st)) {
                         _excludedStereotypes.remove(st);
                    } else if (!checked &&
!_excludedStereotypes.contains(st)) {
                         _excludedStereotypes.add(st);
                    }
                  } else {
                    super.setValueAt(aValue, rowIndex,
columnIndex);
                  }
               }
          }
          private JButton _cancel;
          private JButton _import;
          private JTable _filter_table;
          private StereotypeTableModel _filter_table_model;
          private Collection<Stereotype> _filter;

          public StereotypeFilterDialog(Frame parent,
Collection<Stereotype> profileStereotypes) {
               super(parent, "Set Stereotype Filter",
true);
               Box vbox = Box.createVerticalBox();

               Box hbox = Box.createHorizontalBox();
```

```
                //JLabel label = new JLabel(
(consistencyResult.isEmpty() ? "Model element is
consistent" :  "Model element is inconsistent" ) );

                //hbox.add(label);

                vbox.add(Box.createVerticalStrut(10));
                vbox.add(hbox);

                //if (!consistencyResult.isEmpty()) {
                     _filter_table_model = new
StereotypeTableModel(profileStereotypes);
                    _filter_table = new JTable(
_filter_table_model );
                    JScrollPane scroll = new
JScrollPane(_filter_table);

     scroll.setVerticalScrollBarPolicy(ScrollPaneConstants.
VERTICAL_SCROLLBAR_ALWAYS);
                    scroll.setPreferredSize( new
Dimension(300, 250) );
                    vbox.add(Box.createVerticalStrut(10));
                    vbox.add(scroll);


     _filter_table.getColumnModel().getColumn(0).setPreferr
edWidth(50);

     _filter_table.getColumnModel().getColumn(1).setPreferr
edWidth(250);
                //}

              _import = new JButton("Import");
              _cancel = new JButton("Cancel");

                vbox.add(Box.createVerticalStrut(10));

                hbox = Box.createHorizontalBox();
                hbox.add(_import);
                hbox.add(Box.createHorizontalStrut(50));
                hbox.add(_cancel);
                vbox.add(hbox);

                vbox.add(Box.createVerticalStrut(10));

                add( vbox );
```

```
                setMinimumSize( new Dimension(400, 300));

                setLocationRelativeTo( parent );

                invalidate();
                setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
                pack();

                _import.addActionListener(this);
                _cancel.addActionListener(this);
           }

           @Override
           public void actionPerformed(ActionEvent ae) {
                Object target = ae.getSource();
                if (target == _import) {
                     setVisible(false);

                     Collection<Stereotype> excl =
_filter_table_model.getExcludedStereotypes();
                     Project project =
Application.getInstance().getProject();
                     Collection<Stereotype> bst =
SysMLUtility.getBasicStereotypes( project );
                     excl.removeAll(bst);
                     _filter = excl;

                     dispose();
                } else if (target == _cancel) {
                     setVisible(false);
                     dispose();
                }
           }

           public Collection<Stereotype> getFilter() {
                return _filter;
           }

     }
```

WaitDialog:

```
package gov.nasa.jpl.imce.sysmlnxsync.ui;

import java.awt.Dimension;
import java.awt.Frame;

import javax.swing.Box;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.WindowConstants;

/**
 * Display window during synchronization
 *
 * @author francisco.valdes@jpl.nasa.gov,
 */

public class WaitDialog extends JDialog {

      public WaitDialog(Frame parent, String title,
String message) {
            super(parent, title);

            Box box = Box.createHorizontalBox();

            JLabel label = new JLabel( message );

            box.add(label);

            add( box );

            setPreferredSize( new Dimension(300, 150));
            setLocationRelativeTo( parent );

            invalidate();
            setDefaultCloseOperation(
WindowConstants.DISPOSE_ON_CLOSE);
            pack();
      }

}
```

Utility:

ConsistencyResult:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

public class ConsistencyResult {
     private String _id;
     private String _message;
     private String _type;
     private UpdateObject _updateObject;

     public ConsistencyResult (String type, String id,
String message, UpdateObject obj) {
          _type = type;
          _id = id;
          _message = message;
          _updateObject = obj;
     }

     public boolean canUpdateNX() {
          return _updateObject.canUpdateNX();
     }

     public boolean canUpdateSysML() {
          return _updateObject.canUpdateSysML();
     }

     public String getIdentifier() {
          return _id;
     }

     public String getMessage() {
          return _message;
     }

     public String getType() {
          return _type;
     }

     public void updateNX() {
          _updateObject.updateNX();
     }

     public void updateSysML() {
          _updateObject.updateSysML();
     }
}
```

DefaultNodeHandler:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;

public abstract class DefaultNodeHandler {

      public Property enterExpression( Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
            return null;
      }

      public Class enterFeature( Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
            return null;
      }

      public Class enterPart( Project project, Class
parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
            return null;
      }

      public Property exitExpression( Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
            return null;
      }
```

```
          public Class exitFeature( Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "222" +
sysmlFeature );
               }
               return null;
          }
          public Class exitPart( Project project, Class
parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
               return null;
          }

     }
```

IdentityNodeHandler:

```
     package gov.nasa.jpl.imce.sysmlnxsync.utility;

     import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;
     import com.nomagic.magicdraw.core.Application;
     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     public abstract class IdentityNodeHandler extends
DefaultNodeHandler {
          @Override
          public Property enterExpression( Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
               return sysmlExpression;
          }
          @Override
```

```
          public Class enterFeature( Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
               return sysmlFeature;
          }
          @Override
          public Class enterPart( Project project, Class
parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
               return sysmlPart;
          }
          @Override
          public Property exitExpression( Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
               return sysmlExpression;
          }
          @Override
          public Class exitFeature( Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "111" +
sysmlFeature );
               }
               return sysmlFeature;
          }
          @Override
          public Class exitPart( Project project, Class
parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
               return sysmlPart;
          }

     }
```

InstanceReportResult:

```
     package gov.nasa.jpl.imce.sysmlnxsync.utility;

     import java.util.ArrayList;
     import java.util.List;
     import java.util.Map;

     import com.nomagic.magicdraw.core.Application;
     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.StructuralF
eature;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

     public class InstanceReportResult {
          private String _name, _fname, _pname, _value,
_type;

          public InstanceReportResult (String name, String
fname, String pname, String type, String value) {
               _name = name;
               _fname = fname;
               _pname = pname;
               _type = type;
               _value = value;
          }

          public String getName() {
               return _name;
          }

          public String getFeatureName() {
               return _fname;
          }

          public String getPartName() {
               return _pname;
          }

          public String getValue() {
               return _value;
          }

          public String getType() {
               return _type;
          }

          public static List<InstanceReportResult>
generateList(Project project, Package pkg) {
```

```
               Stereotype targetValuePropertyStereotype =
StereotypesHelper.getStereotype(project, "Targert Value
Property");
               ArrayList<InstanceReportResult> arr = new
ArrayList<InstanceReportResult>();
               Map<StructuralFeature, List<Slot>> hm =
SysMLUtility.getInstanceMap(project, pkg);
               for (StructuralFeature key : hm.keySet()) {
                    String name = key.getName();
                    Slot sl = hm.get(key).get(0);
                    Property df =
((Property)sl.getDefiningFeature());

                    String type = df.getHumanType();
                    String val =
SysMLUtility.getValueSpecificationValue(
sl.getValue().get(0) );
     //             String val = df.getDefault();


     /*Application.getInstance().getGUILog().log(
                                "Slot name: " + key.getName()
+ " type: "
                                          + key +
                                " size: " +
hm.get(key).size() + " val: " + hm.get(key).get(0) +
                                " val2: " +
hm.get(key).get(0).getHumanName() +
                                " val3: " +
hm.get(key).get(0).getDefiningFeature() +
                                " val4: " + df.getDefault() +
                                " val5: " + df.getHumanName()
+
                                " val6: " +
df.getDefaultValue() +
                                " val7: " +
df.getUpperValue()
                    );*/
                    Property target = df;
                    if (StereotypesHelper.hasStereotype(df,
targetValuePropertyStereotype)) {
                         Object o =
StereotypesHelper.getStereotypePropertyFirst(df,
targetValuePropertyStereotype, "Original Classifier");
                         if (o instanceof Property) {
                              target = (Property)o;
                         }
```

```
                    }

                    String fname =
SysMLUtility.getFeatureName(project, target);
                    String pname =
SysMLUtility.getPartName(project, target);
                    arr.add( new InstanceReportResult(name,
fname, pname, type, val) );
                }
                return arr;
          }

     }
```

InteractiveInstanceReportResult:

```
     package gov.nasa.jpl.imce.sysmlnxsync.utility;

     import java.util.ArrayList;
     import java.util.List;
     import java.util.Map;

     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.StructuralF
eature;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;

     public class InteractiveInstanceReportResult {
          private String _name, _fname, _pname,
_defaultValue, _type;
          private List<String> _value;

          public InteractiveInstanceReportResult (String
name, String fname, String pname, String type, String
defaultValue, List<String> value) {
                _name = name;
                _fname = fname;
                _pname = pname;
                _type = type;
                _defaultValue = defaultValue;
```

```
            _value = value;
        }

        public String getName() {
            return _name;
        }

        public String getFeatureName() {
            return _fname;
        }

        public String getPartName() {
            return _pname;
        }

        public String getDefaultValue() {
            return _defaultValue;
        }

        public List<String> getValue() {
            return _value;
        }

        public String getType() {
            return _type;
        }

        public static
List<InteractiveInstanceReportResult> generateList(Project
project, Package pkg) {
            ArrayList<InteractiveInstanceReportResult>
arr = new ArrayList<InteractiveInstanceReportResult>();
            Map<StructuralFeature, List<Slot>> hm =
SysMLUtility.getInstanceMap(project, pkg);
            ArrayList<String> values;
            String defaultValue, val = "", type = "";
            for (StructuralFeature  key : hm.keySet()) {
                defaultValue = "";
                values = new ArrayList<String>();
                for (Slot sl : hm.get(key)) {
                    Property df =
((Property)sl.getDefiningFeature());
                    type = df.getHumanType();
                    if (df.getDefaultValue() != null)
{
```

```
                                defaultValue =
SysMLUtility.getValueSpecificationValue(df.getDefaultValue(
));
                            }
                            for (ValueSpecification vs :
sl.getValue()) {
                                val =
SysMLUtility.getValueSpecificationValue(vs);
                                values.add(val);
                            }
                        }
                        String name = key.getName();
                        String fname =
SysMLUtility.getFeatureName(project, key);
                        String pname =
SysMLUtility.getPartName(project, key);

                        if ("Part Property".equals(type) ||
"Constraint Parameter".equals(type) || "Constraint
Property".equals(type)) {
                            // Ignore and never include in
result list
                        } else if ("Validation Value
Property".equals(type)) {
                            // Ignore and never include in
result list
                        } else if ("Performance Value
Property".equals(type)) {
                            // Ignore and never include in
result list
                        } else {
                            arr.add( new
InteractiveInstanceReportResult(name, fname, pname, type,
defaultValue, values) );
                        }
                    }
                    return arr;
                }

        }
```

PartFileFilter:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import java.io.File;

import javax.swing.filechooser.FileFilter;
/**
```

```
 * File filter for file.prt
 *
 * @author francisco.valdes@jpl.nasa.gov,
 */

public class PartFileFilter extends FileFilter {

     @Override
     public boolean accept(File file) {
          return file.isDirectory() ||
file.getName().toLowerCase().endsWith(".prt");
     }

     @Override
     public String getDescription() {
          // TODO Auto-generated method stub
          return "NX Part Files";
     }

}
```

StereotypeFilterHandler:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;

import java.util.Collection;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

public class StereotypeFilterHandler extends
IdentityNodeHandler {
        private Collection<Stereotype> _filter;

        public
StereotypeFilterHandler(Collection<Stereotype> filter) {
            _filter = filter;
        }

        @Override
        public Class exitFeature(Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
            if (PluginMain.DEBUG) {

    Application.getInstance().getGUILog().log( "000" +
sysmlFeature );
                if (parent != null) {

    Application.getInstance().getGUILog().log( "001" +
parent + " | " + parent.getName() );
                } else {
```

```
     Application.getInstance().getGUILog().log( "001 parent
is null " );
                    }
               }
               boolean included = true;

               if (sysmlFeature != null) {
                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "AAA
Checking feature : " + sysmlFeature.getName() );
                    }
                    for (Stereotype st : _filter) {

     Application.getInstance().getGUILog().log( "BBB
Checking stereotype : " + st.getName() );
                         if
(StereotypesHelper.hasStereotype(sysmlFeature, st)) {
                              included = false;
                              break;
                         }
                    }
                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log( "CCC Is
Included?" + sysmlFeature.getName() + " " + included );
                    }
                    if (included) {
                         return sysmlFeature;
                    } else {
                         Collection<Class> featureList =
SysMLUtility.getFeatures(project, sysmlFeature);
                         ModelElementsManager msm =
ModelElementsManager.getInstance();
                         msm.removeElement(sysmlFeature);
                         for (Class childElement :
featureList) {

     msm.removeElement(childElement);
                              msm.addElement(childElement,
parent);
                         }
                         return parent;

                    }
               }
```

```
            return parent;
        }
    }
```

SysMLModelTraverser:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;

import java.awt.Frame;
import java.util.Collection;

import javax.swing.JOptionPane;
import javax.swing.ProgressMonitor;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.magicdraw.openapi.uml.SessionManager;
import
com.nomagic.magicdraw.ui.dialogs.MDDialogParentProvider;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;

public class SysMLModelTraverser extends Thread {

     public static SysMLModelTraverser launch(Project
project, Class rootPartClass, NXPart nxPart,
DefaultNodeHandler resolver) {
          // Now send some commands to Maple
          SessionManager sm =
SessionManager.getInstance();
          sm.createSession("NX Plugin");

          Frame parentFrame =
MDDialogParentProvider.getProvider().getDialogParent();
          ProgressMonitor pm = new
ProgressMonitor(parentFrame, JOptionPane.PLAIN_MESSAGE,
"Please wait", 0, 10);
```

```
                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Test! 123 "
+ resolver );
                }


                SysMLModelTraverser traverser = new
SysMLModelTraverser( project, rootPartClass, nxPart,
resolver, pm );
                //traverser._sm = sm;
                //traverser.start();

                //try {
                //   traverser.join();
                //} catch (InterruptedException ie) {
                //   // TODO Auto-generated catch block
                //   ie.printStackTrace();
                //}
                traverser.run();

                sm.closeSession();

           //   waitDialog.setVisible(false);

                return traverser;
           }
           private DefaultNodeHandler _handler;
           private NXPart _nxPart;
           private ProgressMonitor _pm;
           private Project _project;
           private Class _resolvedClass;
           private boolean _result;
           private Class _partClass;

           private SysMLModelTraverser (Project project,
Class partClass, NXPart nxPart,
                           DefaultNodeHandler handler,
ProgressMonitor pm) {
                _project = project;
                _partClass = partClass;
                _nxPart = nxPart;
                _handler = handler;
                _pm = pm;
                _result = true;
           }
```

```
          public Class getResolvedClass() {
               return _resolvedClass;
          }

          public boolean getResult() {
               return _result;
          }

          /**
           * Creates a SysML representation of a part. The
mapping that we want in this case is:
           * (NX)Part => (SysML)Block[stereotype=NXOpen]
           *        where:
           *
     (SysML)Block[stereotype=NXOpen].name     =
(NX)Part.name
           *
     (SysML)Block[stereotype=NXOpen].filename =
(NX)Part.filename
           *
           * @param part The part to work with
           * @param filename The filename of the part
           * @throws ReadOnlyElementException
           */
          @Override
          public void run() {

               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Test!
sysmlPart: " + (_partClass) );

     Application.getInstance().getGUILog().log("Test!  NX
part status: " + _nxPart );
               }

               Class result = null;
               try {
                    result = traversePart( null,
_partClass, _nxPart );
               } catch (ReadOnlyElementException roee) {
                    roee.printStackTrace();
               }

               _resolvedClass = result;
               _result = true;
          }
```

```
          private Property traverseExpression( Class
parent, Property sysmlExpression, NXExpression nxExpression
) throws ReadOnlyElementException {
               Property res = _handler.enterExpression(
_project, parent, sysmlExpression, nxExpression );
               _handler.exitExpression( _project, parent,
sysmlExpression, nxExpression );
               return res;
          }

          private Class traverseFeature( Class parent,
Class sysmlFeature, NXFeature nxFeature ) throws
ReadOnlyElementException {
               // Call resolver on it
               Class resolvedFeature =
_handler.enterFeature( _project, parent, sysmlFeature,
nxFeature );
               if (resolvedFeature == null) {
                    resolvedFeature = parent;
               }
               // Traverse expressions
               Collection<Property> sysmlExpressions =
SysMLUtility.getExpressions(_project, resolvedFeature);
               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("traverseFea
ture: [SysML:" + sysmlFeature + ", NX:" + nxFeature +"]" );

     Application.getInstance().getGUILog().log("traverseFea
ture exprs: " + resolvedFeature + ": " +
sysmlExpressions.size() );
               }
               if (nxFeature != null) {
                    Collection<NXExpression> nxExpressions
= nxFeature.getExpressions();
                    Property childExpression;
                    for (NXExpression nxExpression :
nxExpressions) {
                         // Traverse the expression
                         childExpression =
SysMLUtility.findExpressionByName( _project,
sysmlExpressions, nxExpression.getName() );
                         traverseExpression(
resolvedFeature, childExpression, nxExpression );
                         // Remove the expression from our
collection
```

```
     sysmlExpressions.remove(childExpression);
                    }
               }
               // sysmlExpressions now is a list of
rejects, i.e. stuff in SysML with no analogue in NX
               for (Property childExpression :
sysmlExpressions) {
                    traverseExpression(resolvedFeature,
childExpression, null);
               }

               // Traverse child features
               Collection<Class> sysmlFeatures =
SysMLUtility.getFeatures(_project, resolvedFeature);
               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("traverseFea
ture children: " + resolvedFeature + ": " +
sysmlFeatures.size() );
               }
               if (nxFeature != null) {
                    Collection<NXFeature> childFeatures =
nxFeature.getChildren();
                    Class childFeature;
                    for (NXFeature nxChildFeature :
childFeatures) {
                         // Traverse the feature
                         childFeature =
SysMLUtility.findFeatureByName( _project, sysmlFeatures,
nxChildFeature.getName(), nxChildFeature.getType() );
                         traverseFeature( resolvedFeature,
childFeature, nxChildFeature );
                         // Remove the expression from our
collection

     sysmlFeatures.remove(childFeature);
                    }
               }
               // sysmlFeatures now is a list of rejects,
i.e. stuff in SysML with no analogue in NX
               for (Class childFeature : sysmlFeatures) {
                    traverseFeature( resolvedFeature,
childFeature, null );
     //             _handler.enterFeature(_project, parent,
cls, null);
```

```
     //             _handler.exitFeature(_project, parent,
cls, null);
               }

               _handler.exitFeature( _project, parent,
sysmlFeature, nxFeature );

               return resolvedFeature;
          }

          private Class traversePart( Class parent, Class
sysmlPart, NXPart nxPart ) throws ReadOnlyElementException
{
               // if sysmlPart is null, search for part
under current parent

               // Perform resolver action on part itself
               Class resolvedPart = _handler.enterPart(
_project, parent, sysmlPart, nxPart );

               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("traversePar
t: [Parent:" + parent + " SysML:" + sysmlPart + ", NX:" +
nxPart + " resolvedPart " + resolvedPart +"]" );
               }

               // Decide on name for part
               String name = resolvedPart.getName();
               if (!name.endsWith(".prt")) {
                    name = name + ".prt";
                    resolvedPart.setName(name);
               }

               // First traverse feature children
               Collection<Class> sysmlFeatures =
SysMLUtility.getFeatures(_project, resolvedPart);
               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("traversePar
t: SysML feature count: " + sysmlFeatures.size());
               }
               if (nxPart != null) {
                    Collection<NXFeature> nxFeatures =
nxPart.getFeatures();
                    if (PluginMain.DEBUG) {
```

```
     Application.getInstance().getGUILog().log("traversePar
t: NX feature count: " + nxFeatures.size());
                    }
                    Class childFeature;
                    for (NXFeature nxFeature : nxFeatures)
{
                         childFeature =
SysMLUtility.findFeatureByName( _project, sysmlFeatures,
nxFeature.getName(), nxFeature.getType() );
                         traverseFeature( resolvedPart,
childFeature, nxFeature );

     sysmlFeatures.remove(childFeature);
                    }
               }
               // sysmlFeatures now is a list of rejects,
i.e. stuff in SysML with no analogue in NX
               for (Class childFeature : sysmlFeatures) {
                    traverseFeature( resolvedPart,
childFeature, null );
               }

               // Now traverse part children
               Collection<Class> sysmlComponents =
SysMLUtility.getPartChildren(_project, resolvedPart);
               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Part: SysML
component count: " + sysmlComponents.size());
               }
               // Traverse components
               if (nxPart != null) {
                    Collection<NXPart> nxComponents =
nxPart.getOpenComponents();
                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("traversePar
t: NX feature count: " + nxComponents.size());
                    }
                    Class childPart;
                    for (NXPart nxChildPart : nxComponents)
{
                         childPart =
SysMLUtility.findPartByFilePath( _project, sysmlComponents,
nxChildPart.getFile() );
```

```
                        traversePart( resolvedPart,
childPart, nxChildPart );
                        sysmlComponents.remove(childPart);
                    }
                }
                // sysmlComponents now is a list of rejects,
i.e. stuff in SysML with no analogue in NX
                for (Class childPart : sysmlComponents) {
                    traversePart( resolvedPart, childPart,
null );
                }

                _handler.exitPart( _project, parent,
sysmlPart, nxPart );

                return resolvedPart;
            }

    }
```

SysMLParameters:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;

import java.util.Collection;

import com.nomagic.magicdraw.core.Project;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;
import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
import com.nomagic.uml2.impl.ElementsFactory;
/**
 * Class for managing parameter within SysML
 *
 * @author francisco.valdes@jpl.nasa.gov,
 */

public class SysMLParameters {

     public static void addParameterToClass( Project
project, Class sysmlClass, NXExpression param ) {
          String name = param.getName();
          String value = param.getValue();
          ElementsFactory factory =
project.getElementsFactory();
          Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
          Property myProperty =
factory.createPropertyInstance();
          // New property step 1
          myProperty.setName( name );
```

```
                ValueSpecification spec =
factory.createLiteralStringInstance();
                ((LiteralString)spec).setValue( value );
                myProperty.setDefaultValue(spec);

                StereotypesHelper.addStereotype(myProperty,
sysmlValuePropertyStereotype);


     StereotypesHelper.setStereotypePropertyValue(myPropert
y, sysmlValuePropertyStereotype, "currentName",
param.getName());

                if (sysmlClass != null) {
                     myProperty.setUMLClass(sysmlClass);

     sysmlClass.getAttribute().add(myProperty);
                }
          }

          public static String
getSynchronizedParameterName( Project project,
Collection<NXExpression> params, String name ) {
                Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
                NXExpression found = null;
                for (NXExpression expr : params) {
                     if (name.equals(expr.getName())) {
                          found = expr;
                          break;
                     }
                }
                if (found != null) {
                     return
StereotypesHelper.getStereotypePropertyFirst(found.getPrope
rty(), sysmlValuePropertyStereotype,
"currentName").toString();
                } else {
                     return null;
                }
          }

          public static boolean removeAllParameters(Element
sysmlElement) {
                if (sysmlElement instanceof Class) {
```

```
                    Class sysmlPart = (Class)sysmlElement;
                    sysmlPart.getAttribute().clear();
               }
               return true;
          }

          public static boolean removeParameter(Element
sysmlElement, String parameterName) {
               if (sysmlElement instanceof Class) {
                    Class sysmlPart = (Class)sysmlElement;
                    for (Property prop :
sysmlPart.getAttribute() ) {
                         if (parameterName.equals(
prop.getName() )) {

     sysmlPart.getAttribute().remove(prop);
                              return true;
                         }
                    }
               }
               return false;
          }

          public static boolean
setSynchronizedParameterName( Project project,
Collection<NXExpression> params, String name ) {
               Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
               NXExpression found = null;
               for (NXExpression expr : params) {
                    if (name.equals(expr.getName())) {
                         found = expr;
                         break;
                    }
               }
               if (found != null) {

     StereotypesHelper.setStereotypePropertyValue(found.get
Property(), sysmlValuePropertyStereotype, "currentName",
name);
                    return true;
               } else {
                    return false;
               }
          }
```

```
          public static boolean
updateParameter(ElementsFactory elementsFactory, Element
sysmlElement, NXExpression param) {
               if (sysmlElement instanceof Class) {
                    Class sysmlPart = (Class)sysmlElement;
                    Property targetProperty = null;
                    for (Property prop:
sysmlPart.getAttribute() ) {
                         if
(param.getName().equals(prop.getName())) {
                              targetProperty = prop;
                              break;
                         }
                    }

                    if (targetProperty != null) {
                         LiteralString spec =
elementsFactory.createLiteralStringInstance();
                         spec.setValue(param.getValue());

     targetProperty.setDefaultValue(spec);
                         return true;
                    } else {
                         return false;
                    }
               }
               // we don't currently add parameters to
properties so we don't update them
               return false;
          }

     }
```

SysMLUtility:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXClientEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXConnectionExce
ption;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXEngine;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;

import java.awt.Frame;
import java.io.File;
import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.HashSet;
import java.util.List;
import java.util.Map;

import javax.swing.JOptionPane;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.auxiliaryconstructs.mdmodels
.Model;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpe
cification;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralBool
ean;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralInte
ger;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralReal
;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.NamedElemen
t;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Package;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Packageable
Element;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.StructuralF
eature;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Profile;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

     /**
      * Class to interoperate between NX and SysML
      *
      * @author francisco.valdes@jpl.nasa.gov,
      */

     public class SysMLUtility {

          public static Stereotype
featureTypeToStereotype(Project project, String type) {

     Application.getInstance().getGUILog().log("Type is: "
+ type );
               if (type.equals("DATUM_CSYS")) {
```

```
                    return
StereotypesHelper.getStereotype(project,
"NXCoordinateSystem");
               } else if (type.equals("DATUM_PLANE")) {
                    return
StereotypesHelper.getStereotype(project, "NXDatumPlane");
               } else if (type.equals("EXTRACT_BODY")) {
               } else if (type.equals("SIMPLE HOLE")) {
                    return
StereotypesHelper.getStereotype(project, "NXSimpleHole");
               } else if (type.equals("HOLE PACKAGE")) {
               } else if (type.equals("SB_FLAT_SOLID")) {
               } else if (type.equals("FLAT_PATTERN")) {
               } else if (type.equals("BALL_END_SLOT")) {
                    return
StereotypesHelper.getStereotype(project, "NXBallEndSlot");
               } else if (type.equals("MIRROR")) {
                    return
StereotypesHelper.getStereotype(project, "NXMirrorBody");
               } else if (type.equals("MIRROR_SET")) {
                    return
StereotypesHelper.getStereotype(project,
"NXMirrorFeature");
               } else if (type.equals("CHAMFER")) {
                    return
StereotypesHelper.getStereotype(project, "NXChamfer");
               } else if (type.equals("SKETCH") ||
type.equals("Sketch")) {
                    return
StereotypesHelper.getStereotype(project, "NXSketch");
               } else if (type.equals("Base Tab")) {
                    return
StereotypesHelper.getStereotype(project, "NXSMBaseFlange");
               } else if (type.equals("Break Corner")) {
                    return
StereotypesHelper.getStereotype(project,
"NXSMCornerBreak");
               } else if (type.equals("Flange")) {
                    return
StereotypesHelper.getStereotype(project, "NXSMFlange");
               } else if (type.equals("Normal Cutout")) {
                    return
StereotypesHelper.getStereotype(project,
"NXSMNormalCutOut");
               }
               return null;
          }
```

```
          public static Property findExpressionByName(
Project project, Collection<Property> exprs, String
paramName ) {
               Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
               Object currentName;
               for (Property expr : exprs) {
                    currentName =
StereotypesHelper.getStereotypePropertyFirst(expr,
sysmlValuePropertyStereotype, "currentName");
                    if (currentName != null &&
paramName.equals(currentName.toString())) {
                         return expr;
                    }
               }
               return null;
          }

          public static Class findFeatureByName( Project
project, Collection<Class> features, String featureName,
String featureType ) {
               Stereotype nxPartFeature =
StereotypesHelper.getStereotype(project, "NXPartFeature");
               Object childFtype;
               String childFNewname, childFOldname;
               for (Class childFeature : features) {
                    childFtype =
StereotypesHelper.getStereotypePropertyFirst(childFeature,
nxPartFeature, "featureType");
                    childFNewname = childFeature.getName();
                    childFOldname =
StereotypesHelper.getStereotypePropertyFirst(childFeature,
nxPartFeature, "currentFeatureName").toString();

                    if ((featureName.equals(childFOldname)
|| featureName.equals(childFNewname)) && childFtype != null
&& featureType.equals(childFtype.toString())) {
                         if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Feature
Search: [NX=" + featureName+"][SysML="+childFOldname+"]" );
                         }
                         return childFeature;
                    }
               }
```

```
                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Feature
Search: [NX=" + featureName+"][SysML=null]" );
                }
                return null;
           }

           public static Class findPartByFilePath( Project
project, Collection<Class> parts, File path ) {
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                Object currentPartPath;
                File currentPartFile;
                for (Class elem : parts) {
                     currentPartPath =
StereotypesHelper.getStereotypePropertyFirst(elem,
nxPartStereotype, "currentPartPath");
                     if (currentPartPath == null) continue;
                     currentPartFile = new File(
currentPartPath.toString() );

     Application.getInstance().getGUILog().log("SYSml Part
name: " + currentPartPath.toString() );
                     if (currentPartFile.equals(path)) {
                          return elem;
                     }
                }
                return null;
           }

           public static Collection<Class> getAllParts(
Project project, Package pkg ) {
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
                Collection<Class> parts = new
HashSet<Class>();
                Collection<PackageableElement> children =
pkg.getPackagedElement();
                for (PackageableElement child : children) {
                     if
(StereotypesHelper.hasStereotype(child, nxPartStereotype))
{
                          parts.add( (Class)child );
                     }
                }
                return parts;
```

```
          }

          /*public static NamedElement
getCorrespondingElement(Project project, NamedElement
userElement, Stereotype st) {
               Object val =
StereotypesHelper.getStereotypePropertyFirst(userElement,
st, "currentPartPath");
               String originalName = (val != null ?
val.toString() : null);
               Stereotype otherStereotype =
getCorrespondingStereotype(project, st);
               if (otherStereotype != null && originalName
!= null) {
                    return getClassByPathname( project,
userElement.getOwner(), otherStereotype, originalName );
               } else {
                    return null;
               }
          }*/
          private static Collection<NamedElement>
getAssemblyComponents(Element queryElement, Project
project) {
               Stereotype nxPartStereotype         =
StereotypesHelper.getStereotype(project, "NXPart");
               Stereotype nxPartPropertyStereotype =
StereotypesHelper.getStereotype(project, "NXPartProperty");

               Collection<NamedElement> parts = new
HashSet<NamedElement>();
               Collection<Element> children =
queryElement.getOwnedElement();
               for (Element child : children) {
                    if
(StereotypesHelper.hasStereotype(queryElement,
nxPartStereotype)) {
                         parts.add( (NamedElement)child );
                    } else if
(StereotypesHelper.hasStereotype(queryElement,
nxPartPropertyStereotype)) {
                         parts.add( (NamedElement)child );
                    }
               }
               return parts;
          }
```

```
          public static Collection<Stereotype>
getBasicStereotypes( Project project ) {
               String nm;
               Profile nxProfile =
StereotypesHelper.getProfile( project, "NXProfile" );

     Application.getInstance().getGUILog().log("NX Profile
is " + (nxProfile != null) + "|" + nxProfile.getName() );
               Collection<Stereotype> stc =
StereotypesHelper.getStereotypesByProfile( nxProfile );
               ArrayList<Stereotype> list = new
ArrayList<Stereotype>();
               for (Stereotype st : stc) {
                    nm = st.getName();
                    if (nm.equals("NXPart") ||
nm.equals("NXAssembly") || nm.equals("NXFeature") ||
                         nm.equals("NXPartFeature") ||
nm.equals("NXProject") ||
                         nm.equals("NXSheetMetalFeatures")
||
                         nm.equals("NXPartProperty") ||
nm.equals("NXValueProperty")) {
                         list.add(st);
                    }
               }
               return list;
          }

          public static Collection<Property>
getExpressions( Project project, Class parent ) {
               Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
               Collection<Property> params = new
HashSet<Property>();
               for (Property prop : parent.getAttribute())
{
                    if
(StereotypesHelper.hasStereotype(prop,
sysmlValuePropertyStereotype)) {
                         params.add( prop );
                    }
               }
               return params;
          }
```

```
          public static Collection<NXExpression>
getExpressionsNX( Project project, Class parent ) {
               Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
               Collection<NXExpression> params = new
HashSet<NXExpression>();
               ValueSpecification spec;
               String paramName = null;
               String paramValue = null;

               for (Property prop : parent.getAttribute())
{
                    if
(StereotypesHelper.hasStereotype(parent,
sysmlValuePropertyStereotype)) {
                         paramName = prop.getName();
                         spec = prop.getDefaultValue();
                         paramValue =
SysMLUtility.getValueSpecificationValue(spec);
                    }
                    params.add( new NXExpression(
paramName, paramValue, prop ) );
               }
               return params;
          }

          public static Collection<Class> getFeatures(
Project project, Class parent ) {
               Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");

               Collection<Class> features = new
HashSet<Class>();
               Collection<Element> children =
parent.getOwnedElement();
               for (Element child : children) {
                    if
(StereotypesHelper.hasStereotype(child,
nxFeatureStereotype)) {
                         features.add( (Class)child );
                    }
               }
               return features;
          }
```

```
          public static String getFeatureName(Project
project, StructuralFeature element) {
               Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");
               Element parent = element.getOwner();
               while (parent != null) {
                    if
(StereotypesHelper.hasStereotype(parent,
nxFeatureStereotype)) {
                         return
((NamedElement)parent).getName();
                    }
                    parent = parent.getOwner();
               }
               return "";
          }

          public static String getPartName(Project project,
StructuralFeature element) {
               Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
               Element parent = element.getOwner();
               while (parent != null) {
                    if
(StereotypesHelper.hasStereotype(parent,
nxFeatureStereotype)) {
                         return
((NamedElement)parent).getName();
                    }
                    parent = parent.getOwner();
               }
               return "";
          }

          public static List<String>
getQualifiedFeatureName(Project project, StructuralFeature
element) {
               ArrayList<String> arr = new
ArrayList<String>();
               Element parent = element.getOwner();
               while (parent != null && parent instanceof
NamedElement) {
                    if (parent instanceof Model) { break; }
                    arr.add(0,
((NamedElement)parent).getName());
                    parent = parent.getOwner();
               }
```

```
                return arr;
          }


          public static Map<StructuralFeature, List<Slot>>
getInstanceMap(Project project, Package pkg) {
                Collection<Element> children =
pkg.getOwnedElement();
                HashMap<StructuralFeature,List<Slot>> hm =
new HashMap<StructuralFeature ,List<Slot>>();
                for (Element child : children) {
                     if (child instanceof
InstanceSpecification) {
                          InstanceSpecification is =
(InstanceSpecification)child;
                          //Class block = getBlock(is);
                          Collection<Element> grandchildren
= child.getOwnedElement();
                          for (Element slotElement :
grandchildren) {
                               if (slotElement instanceof
Slot) {
                                    Slot slot1 =
(Slot)slotElement;
                                    StructuralFeature str =
slot1.getDefiningFeature();
                                    if
(!(hm.containsKey(str))) {
                                         hm.put(str, new
ArrayList<Slot>());
                                    }
                                    hm.get(str).add(slot1);
                               }
                          }
                     }
                }
                return hm;
          }

          public static String
getValueSpecificationValue(ValueSpecification val) {
                if (val instanceof LiteralReal) {
                     LiteralReal lr = (LiteralReal)val;
                     return Double.toString(lr.getValue());
                } else if (val instanceof LiteralInteger) {
                     LiteralInteger lr =
(LiteralInteger)val;
```

```
                    return Integer.toString(lr.getValue());
               } else if (val instanceof LiteralBoolean) {
                    LiteralBoolean lb =
(LiteralBoolean)val;
                    return Boolean.toString(lb.isValue());
               } else if (val instanceof LiteralString) {
                    LiteralString ls = (LiteralString)val;
                    return ls.getValue();
               } else {
                    return val.toString();
               }
          }

          public static Collection<Class> getPartChildren(
Project project, Class partClass ) {
               Stereotype nxPartStereotype =
StereotypesHelper.getStereotype(project, "NXPart");
               Collection<Class> parts = new
HashSet<Class>();
               for (Element child :
partClass.getOwnedElement()) {
                    if
(StereotypesHelper.hasStereotype(child, nxPartStereotype))
{
                         parts.add( (Class)child );
                    }
               }
               return parts;
          }

          /*public static Class getPartClass( Package
partPackage ) {
               for (Object o :
partPackage.getOwnedElement()) {
                    if (o instanceof Class) {
                         return (Class)o;
                    }
               }
               return null;
          }*/

          public static Collection<Stereotype>
getProfileStereotypes( Project project ) {
               String nm;
               Profile nxProfile =
StereotypesHelper.getProfile( project, "NXProfile" );
```

```
     Application.getInstance().getGUILog().log("NX Profile
is " + (nxProfile != null) + "|" + nxProfile.getName() );
               Collection<Stereotype> stc =
StereotypesHelper.getStereotypesByProfile( nxProfile );
               ArrayList<Stereotype> list = new
ArrayList<Stereotype>();
               for (Stereotype st : stc) {
                    nm = st.getName();
                    if (!(nm.equals("NXPart") ||
nm.equals("NXAssembly") || nm.equals("NXFeature") ||
                         nm.equals("NXPartFeature") ||
nm.equals("NXProject") ||
                         nm.equals("NXSheetMetalFeatures")
||
                         nm.equals("NXPartProperty") ||
nm.equals("NXValueProperty"))) {
                         list.add(st);
                    }
               }
               return list;
          }

          public static NXPart openPart(Frame parentFrame,
File nxFile) {
               boolean success = true;
               NXPart nxPart = null;
               NXEngine engine;
               // Read parameter and components

               if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("NX Feature
Type: " + nxFile.toString() );
               }

               if (!(nxFile != null && nxFile.exists() &&
nxFile.canRead())) {
                    throw new IllegalStateException();
               }

               try {
                    engine = new NXClientEngine();
               } catch (NXConnectionException nce) {
                    JOptionPane.showMessageDialog(
parentFrame,
```

```
                         "Cannot connect to Siemens
NX.\nEnsure that Execute->NX Open->nx_maple_server.dll has
been run",
                         "NX connection error",
                         JOptionPane.ERROR_MESSAGE
                    );
                    return null;
               }


     Application.getInstance().getGUILog().log("NX Open
Connection: " + success );

               nxPart = engine.openPart( nxFile, true );

               engine.closeConnection();

               if (nxPart != null) {
                    return nxPart;
               } else {
                    return null;
               }
          }

          public static NXPart renameNXPart( NXEngine
engine, NXPart part, File newFile ) {
               File originalFile = new File( part.getPath()
);
               if (newFile.equals(originalFile)) {
                    return null;
               }
               // do nothing, no change needed
               if (newFile.exists()) {
                    JOptionPane.showMessageDialog(null,
                         "New part file name already
exists.\nRemove existing file or choose a different name.",
                         "updating NX file error",
                         JOptionPane.ERROR_MESSAGE
                    );
                    return null;
               }

               // RENAME is attempted
               // Renames are allowed, the rename is
nontrivial (i.e. oldname != newname) and newname doesn't
exist yet
```

```
                //String originalName =
originalFile.getAbsolutePath();
                boolean closeResult = engine.closePart( part
);

                originalFile.renameTo(newFile);

                if (!newFile.exists()) {
                     JOptionPane.showMessageDialog(null,
                          "Could not rename NX part
file.\nPlease ensure the target directory is writable.",
                          "Part file error",
                          JOptionPane.ERROR_MESSAGE
                     );
                     return null;
                }

                NXPart newPart = engine.openPart( newFile );

                return newPart;
                //List<String> qualifiedPath =
getSysMLQualifiedName(
Application.getInstance().getProject(), queryElement,
nxStereotype );

     //                   Stereotype otherStereotype =
getCorrespondingStereotype(Application.getInstance().getPro
ject(), nxPartStereotype);
     //                   if (otherStereotype != null) {
     //                        NamedElement otherElem =
getClassByPathname( project, queryElement.getOwner(),
otherStereotype, originalName );
     //                        otherElem.setName(name);
     //
     StereotypesHelper.setStereotypePropertyValue(
otherElem, otherStereotype, "dir", dir );
     //
     StereotypesHelper.setStereotypePropertyValue(
otherElem, otherStereotype, "currentPartPath", filename );
     //                   }
                    // At this point, the Class and the
PartProperty should have the same name, dir, and
currentPartPath,
                    // which will all correspond to the
present state of the corresponding part file in NX.

          /*   } else {
```

```
                    // if renames are not allowed then
display warning to user
                    JOptionPane.showMessageDialog(null,
                         "Cannot change name of part in
assembly.\nPlease change part name within NX.",
                         "Cannot change part name",
                         JOptionPane.WARNING_MESSAGE
                    );
               }*/
          }

          private static boolean updateFeatureToNX(Project
project, NXEngine engine, NXPart part, NamedElement
featureElement) {
               Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");
               if
(StereotypesHelper.hasStereotype(featureElement,
nxFeatureStereotype)) {
                    boolean result = true;
                    String newName =
featureElement.getName();
                    String oldName =
StereotypesHelper.getStereotypePropertyFirst(featureElement
, nxFeatureStereotype, "currentFeatureName").toString();

                    if (!oldName.equals(newName)) {
                         result =
engine.renameFeature(part, oldName, newName);
                         for (NamedElement subfeature :
getFeatures( project, (Class)featureElement) ) {
                              if (!result) break;
                              result = result &&
updateFeatureToNX(project, engine, part, subfeature);
                         }
                    }
                    return result;
               } else {
                    return false;
               }
          }

          private static boolean
updatePartToNXInternal(Project project, NXEngine engine,
                         NamedElement contextElement,
boolean renamesAllowed) {
```

```
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype( project, "NXPart" );
                Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(project, "NXAssembly");

     //         Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");

                Collection<NXExpression> params =
SysMLUtility.getExpressionsNX( project,
(Class)contextElement );

                boolean success = true;
                String name = contextElement.getName();

                if (!name.endsWith(".prt")) {
                     name = name + ".prt";
                     contextElement.setName(name);
                }

                Element queryElement = null;
                queryElement = contextElement;

                // Now send some commands to Maple
                String filename, dir, originalName,
uniqueID;

                dir =
StereotypesHelper.getStereotypePropertyFirst(queryElement,
nxPartStereotype, "directory").toString();
                originalName =
StereotypesHelper.getStereotypePropertyFirst(queryElement,
nxPartStereotype, "currentPartPath").toString();
                uniqueID =
StereotypesHelper.getStereotypePropertyFirst(queryElement,
nxPartStereotype, "uniqueID").toString();
                filename = dir + File.separator + name;

                File newFile = new File( filename );
                File originalFile = new File( originalName
);
                NXPart part = engine.openPart(originalFile);
     //         List<String> featureList =
getFeatureNames(engine, part);

                //filename = dir + File.separator + name;
```

```
                NXPart newPart = renameNXPart( engine, part,
newFile );
                if (newPart != null) {

     StereotypesHelper.setStereotypePropertyValue(
queryElement, nxPartStereotype, "currentPartPath", filename
);
                     part = newPart;
                }

     //         File file = new File( filename );
     //
     //         NXPart part = engine.openPart( file );

                if (part != null) {
                     //overallSuccess = overallSuccess &&
engine.setWorkPart( filename );
                     success = engine.setParameterInfo(
part, params );

                     Collection<Class> childFeatures =
SysMLUtility.getFeatures( project, (Class)queryElement);
                     for (NamedElement featureElement :
childFeatures) {
                          success =
updateFeatureToNX(project, engine, part, featureElement);
                     }

                     success = engine.savePart( part );
                } else {
                     JOptionPane.showMessageDialog(null,
                          "Could not open NX part
file.\nPlease ensure the file exists and is readable.",
                          "Part file error",
                          JOptionPane.ERROR_MESSAGE
                     );
                }

                Collection<NamedElement> childParts =
getAssemblyComponents(queryElement, project);
                for (NamedElement childPart : childParts) {
                     success =
updatePartToNXInternal_recurse(project, engine, childPart,
renamesAllowed );
                }

                return success;
```

```
          }

          /*public static Stereotype
getCorrespondingStereotype(Project project, Stereotype st)
{
               Stereotype nxPartStereotype         =
StereotypesHelper.getStereotype(project, "NXPart");
               Stereotype nxPartPropertyStereotype =
StereotypesHelper.getStereotype(project, "NXPartProperty");
               if (st == nxPartStereotype) {
                    return nxPartPropertyStereotype;
               } else if (st == nxPartPropertyStereotype) {
                    return nxPartStereotype;
               } else {
                    return null;
               }
          }*/

          private static boolean
updatePartToNXInternal_recurse(Project project, NXEngine
engine,
                         NamedElement queryElement, boolean
renamesAllowed) {
               Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");

               boolean success = false;

               //Stereotype nxStereotype =
NXStereotype.getStereotype( project, queryElement );

               // Perform any feature-level renames that
are required

               Collection<NamedElement> childParts =
getAssemblyComponents(queryElement, project);
               for (NamedElement child : childParts) {
                    success =
updatePartToNXInternal_recurse(project, engine, child,
renamesAllowed);
               }

               return success;
          }

          public static boolean updateToNX(Project project,
NamedElement userElement) {
```

```
                boolean success;
                Stereotype nxPartStereotype =
StereotypesHelper.getStereotype( project, "NXPart" );

                if (nxPartStereotype == null ||
!StereotypesHelper.hasStereotype(userElement,
nxPartStereotype)) {
                     return false;
                }

                if (userElement instanceof Class) {
                     NXEngine engine;
                     try {
                          engine = new NXClientEngine();
                     } catch (NXConnectionException ne) {
                          engine = null;
                     }
                     if (engine != null) {
                          updatePartToNXInternal(null,
engine, userElement, true);
                          engine.closeConnection();
                          success = true;
                     }

                     return true;
                } else {
                     return true;
                }
           }
      }
```

UpdateFromNXResolver:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;

import java.io.File;
import java.util.Collection;
import java.util.HashMap;

import com.nomagic.magicdraw.core.Application;
import com.nomagic.magicdraw.core.Project;
import
com.nomagic.magicdraw.openapi.uml.ModelElementsManager;
import
com.nomagic.magicdraw.openapi.uml.PresentationElementsManag
er;
import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
import
com.nomagic.magicdraw.uml.symbols.DiagramPresentationElemen
t;
import
com.nomagic.magicdraw.uml.symbols.shapes.ShapeElement;
import com.nomagic.uml2.ext.jmi.helpers.ModelHelper;
import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Association
;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Element;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.InstanceSpe
cification;
import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
```

```
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Slot;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;
     import com.nomagic.uml2.impl.ElementsFactory;

     public class UpdateFromNXResolver extends
DefaultNodeHandler {

          private InstanceSpecification _featureInstance;
          private Collection<Stereotype> _filter;
          private HashMap<Class,ShapeElement> _hm = new
HashMap<Class,ShapeElement>();

          public
UpdateFromNXResolver(Collection<Stereotype> filter) {
               _filter = filter;
          }

          private void addChildClass(Project project, Class
classB, Class classA) {
               if (classA == null || classB == null) {
                    return;
               }
               Element model = project.getModel();
               ElementsFactory f =
project.getElementsFactory();
               ModelElementsManager modelElementsManager =
ModelElementsManager.getInstance();

               try {
                    Association link =
f.createAssociationInstance();
                    //Dependency dependency =
f.createDependencyInstance();
                    modelElementsManager.addElement(link,
model);
                    ModelHelper.setClientElement(link,
classA);
                    ModelHelper.setSupplierElement(link,
classB);

                    DiagramPresentationElement
activeDiagram = project.getActiveDiagram();
```

```
                    PresentationElementsManager
presentationElementsManager =
PresentationElementsManager.getInstance();

                    if (activeDiagram != null) {
                         ShapeElement clientShape;
                         if (!_hm.containsKey(classA)) {
                              clientShape =
presentationElementsManager.createShapeElement(classA,
activeDiagram);
                              _hm.put(classA, clientShape);
                         } else {
                              clientShape =
_hm.get(classA);
                         }
                         ShapeElement supplierShape;
                         if (!_hm.containsKey(classB)) {
                              supplierShape =
presentationElementsManager.createShapeElement(classB,
activeDiagram);
                              _hm.put(classB,
supplierShape);
                         } else {
                              supplierShape =
_hm.get(classB);
                         }

     presentationElementsManager.createPathElement(link,
clientShape, supplierShape);
                    } else {

     Application.getInstance().getGUILog().log("activeDiagr
am is NULL ");
                    }
               } catch (ReadOnlyElementException roee) {
               }
          }

          @Override
          public Property enterExpression(Project project,
Class parent,
                         Property sysmlExpression,
NXExpression nxExpression) throws ReadOnlyElementException
{
               if (nxExpression != null) {
                    String nxName = nxExpression.getName();
```

```
                    ElementsFactory elementsFactory =
project.getElementsFactory();
                    Stereotype
sysmlNXValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");
                    Stereotype sysmlValuePropertyStereotype
= StereotypesHelper.getStereotype(project,
"ValueProperty");

                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log(
                                   "Visiting Feature:
SysML: " + (sysmlExpression != null ?
sysmlExpression.getName() : "[NULL] ")
                                   + " NX : " +
(nxExpression != null ? nxExpression.getName() : "[NULL] ")
                         );
                    }

                    Property resolvedExpression;
                    LiteralString blockSpec;
                    //LiteralString instanceSpec;
                    Slot slot;

                    // So we have the parameter, now set
the value
                    //instanceSpec =
elementsFactory.createLiteralStringInstance();
          //
     instanceSpec.setValue(nxExpression.getValue());

                    if (sysmlExpression != null) {
                         resolvedExpression =
sysmlExpression;
                        blockSpec =
(LiteralString)resolvedExpression.getDefaultValue();
                         // So we have the parameter, now
set the value

     blockSpec.setValue(nxExpression.getValue());
                         //slot =
resolvedExpression.get_slotOfDefiningFeature().iterator().n
ext();
                    } else {
```

```
                        resolvedExpression =
elementsFactory.createPropertyInstance();
                        blockSpec =
elementsFactory.createLiteralStringInstance();
                        // So we have the parameter, now
set the value

     blockSpec.setValue(nxExpression.getValue());


     resolvedExpression.setName(nxName);

     StereotypesHelper.addStereotype(resolvedExpression,
sysmlNXValuePropertyStereotype);

     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "currentName",
nxName);

     StereotypesHelper.addStereotype(resolvedExpression,
sysmlValuePropertyStereotype);
                    //
     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "Type", "Real");

     resolvedExpression.setDefaultValue(blockSpec);
                        slot =
elementsFactory.createSlotInstance();
               //
     slot.setDefiningFeature(resolvedExpression);
               //
     slot.setOwningInstance(_featureInstance);


     ModelElementsManager.getInstance().addElement(resolved
Expression, parent);
                    }

               //   slot.getValue().add(instanceSpec);


                    // LiteralReal realSpec =
elementsFactory.createLiteralRealInstance();
                    // realSpec.setValue(
Double.parseDouble( nxExpression.getValue() ) );

                    // Set instance relationships
```

```
                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Updated
expression: " + resolvedExpression.getName() + " child of "
+ parent.getName() );
                    }
                    return resolvedExpression;
               } else if (sysmlExpression != null) {

     ModelElementsManager.getInstance().removeElement(sysml
Expression);
               }
               return null;
          }

          @Override
          public Class enterFeature(Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
               if (nxFeature != null) {
                    ElementsFactory elementsFactory =
project.getElementsFactory();
                    Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");

                    Stereotype additionalStereotype = null;
                    Class resolvedFeature;
                    if (sysmlFeature != null) {
                         resolvedFeature = sysmlFeature;
                    } else {
                         String nxName =
nxFeature.getCustomName() != null ?
nxFeature.getCustomName() : nxFeature.getName();
                         resolvedFeature =
elementsFactory.createClassInstance();
                         resolvedFeature.setName(nxName);

     StereotypesHelper.addStereotype(resolvedFeature,
nxFeatureStereotype);


                         String type = nxFeature.getType();
                         if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("NX Feature
Type: " + type );
```

```
                        }

                        if (type != null) {
                             additionalStereotype =
SysMLUtility.featureTypeToStereotype(project, type);

     Application.getInstance().getGUILog().log("SysML
Stereotype for " + type + " is non-null: " +
(additionalStereotype != null) );
                             if (additionalStereotype !=
null) {

     StereotypesHelper.addStereotype(resolvedFeature,
additionalStereotype);
                             }
                        }
                   }

                   if (_filter != null &&
additionalStereotype != null &&
_filter.contains(additionalStereotype)) {
                        // skip this feature and absorb
any children into its parent
                        return null;
                   } else {
                        // Set stereotype property values

     StereotypesHelper.setStereotypePropertyValue(resolvedF
eature, nxFeatureStereotype, "currentFeatureName",
nxFeature.getName() );

     StereotypesHelper.setStereotypePropertyValue(resolvedF
eature, nxFeatureStereotype, "nxName", nxFeature.getName()
);

     StereotypesHelper.setStereotypePropertyValue(resolvedF
eature, nxFeatureStereotype, "featureType",
nxFeature.getType() );


     ModelElementsManager.getInstance().addElement(resolved
Feature, parent);
     //
     Application.getInstance().getGUILog().log("Updated
feature: " + resolvedFeature.getName() + " child of " +
parent.getName() );
```

```
                        //addChildClass(project,
resolvedFeature, parent);

                        return resolvedFeature;
                    }
                } else if (sysmlFeature != null) {

     ModelElementsManager.getInstance().removeElement(sysml
Feature);
                }
                return null;
            }

            @Override
            public Class enterPart(Project project, Class
parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
                if (nxPart != null) {
                    ElementsFactory elementsFactory =
project.getElementsFactory();
                    Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(project, "NXPart");
                    Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(project, "NXAssembly");

                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log(
                                    "Visiting Part: SysML: "
+ (sysmlPart != null ? sysmlPart.getName() : "[NULL] ")
                                    + " NX : " + (nxPart !=
null ? nxPart.getName() : "[NULL] ")
                                    );
                    }

                    Class resolvedPart;
                    if (sysmlPart != null) {
                        resolvedPart = sysmlPart;
                    } else {
                        resolvedPart =
elementsFactory.createClassInstance();

     resolvedPart.setName(nxPart.getName());

     StereotypesHelper.addStereotype(resolvedPart,
nxPartStereotype);
                    }
```

```
                    // Now set the appropriate stereotypes
                    // Set some special stereotype
properties, in this case the filename and unique ID
                    File file = new File( nxPart.getPath()
);
                    String uid =
nxPart.getUniqueIdentifier();

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "directory", file.getParent());

     StereotypesHelper.setStereotypePropertyValue(resolvedP
art, nxPartStereotype, "currentPartPath",
file.getAbsolutePath());

     StereotypesHelper.setStereotypePropertyValue(sysmlPart
, nxPartStereotype, "uniqueID", uid);

                    if (nxPart.isAssembly()) {

     StereotypesHelper.addStereotype(resolvedPart,
nxAssemblyStereotype);
                    }
                    if (parent != null) {

     ModelElementsManager.getInstance().addElement(resolved
Part, parent);
                    }
                    //addChildClass(project, resolvedPart,
parent);
                    return resolvedPart;
               } else if (sysmlPart != null) {

     ModelElementsManager.getInstance().removeElement(sysml
Part);
               }
               return null;
          }
     }
```

UpdateObject:

```
package gov.nasa.jpl.imce.sysmlnxsync.utility;

public abstract class UpdateObject {
     public abstract boolean canUpdateNX();
     public abstract boolean canUpdateSysML();
     public void updateNX() {
     }
     public void updateSysML() {
     }
}
```

UpdateToNXResolver:

```
     package gov.nasa.jpl.imce.sysmlnxsync.utility;

     import
gov.nasa.jpl.imce.sysmlnxsync.controller.PluginMain;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXEngine;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXExpression;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXFeature;
     import
gov.nasa.jpl.imce.sysmlnxsync.nxconnection.NXPart;

     import java.io.File;
     import java.util.ArrayList;

     import com.nomagic.magicdraw.core.Application;
     import com.nomagic.magicdraw.core.Project;
     import
com.nomagic.magicdraw.openapi.uml.ReadOnlyElementException;
     import
com.nomagic.uml2.ext.jmi.helpers.StereotypesHelper;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Class;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.LiteralStri
ng;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.Property;
     import
com.nomagic.uml2.ext.magicdraw.classes.mdkernel.ValueSpecif
ication;
     import
com.nomagic.uml2.ext.magicdraw.mdprofiles.Stereotype;

     public class UpdateToNXResolver extends
DefaultNodeHandler {
          private NXEngine _engine;
          private ArrayList<NXExpression> _paramsToSet;
          private ArrayList<NXPart> _partHierarchy;

          public UpdateToNXResolver(NXEngine engine) {
               _engine = engine;
               _paramsToSet = new
ArrayList<NXExpression>();
               _partHierarchy = new ArrayList<NXPart>();
```

```
          }

          @Override
          public Property enterExpression(Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
               Stereotype sysmlValuePropertyStereotype =
StereotypesHelper.getStereotype(project,
"NXValueProperty");

               if (sysmlExpression == null) {
                    throw new IllegalStateException("Empty
MagicDraw expression");
               } else if(nxExpression == null) {
                    throw new IllegalStateException("Empty
NX expression");
               }

               NXPart currentPart = _partHierarchy.get(0);

               Property resolvedExpression;
               resolvedExpression = sysmlExpression;

               String newName = sysmlExpression.getName();
               String oldName =
StereotypesHelper.getStereotypePropertyFirst(sysmlExpressio
n, sysmlValuePropertyStereotype, "currentName").toString();

               if (!oldName.equals(newName)) {
                    boolean result =
_engine.renameParameter( currentPart, oldName, newName );
                    if (!result) {
                         throw new
IllegalStateException("Engine: cannot rename expression");
                    }

     StereotypesHelper.setStereotypePropertyValue(resolvedE
xpression, sysmlValuePropertyStereotype, "currentName",
newName );
               }

               // Set stereotype property values
               ValueSpecification spec =
sysmlExpression.getDefaultValue();
               String sysmlValue = (spec instanceof
LiteralString ? ((LiteralString)spec).getValue() : null );
```

```
                if (nxExpression != null && sysmlValue !=
null && !sysmlValue.equals(nxExpression.getValue())) {
                    NXExpression newExpr = new
NXExpression( newName, sysmlValue,
nxExpression.getProperty() );
                    _paramsToSet.add( newExpr );
                    _engine.setParameterValue( currentPart,
newName, sysmlValue );
                    if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Updated
expression: " + resolvedExpression.getName() + " child of "
+ parent.getName() );
                    }
                }
                return resolvedExpression;
          }

          @Override
          public Class enterFeature(Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
                if (sysmlFeature == null) {
                    // We couldn't find a sysmlFeature with
this NX name, suggesting the feature's been renamed
                    // on the SysML side
                    throw new IllegalStateException("Empty
MagicDraw feature");
                } else if (nxFeature == null) {
                    throw new IllegalStateException("Empty
NX feature");
                }
                Stereotype nxFeatureStereotype =
StereotypesHelper.getStereotype(project, "NXPartFeature");
                String newName = sysmlFeature.getName();
                String internalName =
StereotypesHelper.getStereotypePropertyFirst(sysmlFeature,
nxFeatureStereotype, "currentFeatureName").toString();

                NXPart part = _partHierarchy.get(0);
                _engine.renameFeature( part, internalName,
newName );
                if (PluginMain.DEBUG) {

     Application.getInstance().getGUILog().log("Set feature
name: " + sysmlFeature.getName() + " child of " +
parent.getName() );
```

```
                }
                return sysmlFeature;
            }

            @Override
            public Class enterPart(Project project, Class
parent, Class sysmlPart, NXPart nxPart) throws
ReadOnlyElementException {
                    Stereotype nxPartStereotype     =
StereotypesHelper.getStereotype(project, "NXPart");
                Stereotype nxAssemblyStereotype =
StereotypesHelper.getStereotype(project, "NXAssembly");

                // Now set the appropriate stereotypes
                // Set some special stereotype properties,
in this case the filename and unique ID
                String newName = sysmlPart.getName();

                String dir =
StereotypesHelper.getStereotypePropertyFirst(sysmlPart,
nxPartStereotype, "directory").toString();
                String filename = dir + File.separator +
newName;

                File newFile = new File( filename );

                NXPart finalPart = nxPart;
                NXPart newPart = SysMLUtility.renameNXPart(
_engine, nxPart, newFile );
                if (newPart != null) {

     StereotypesHelper.setStereotypePropertyValue(
sysmlPart, nxPartStereotype, "currentPartPath", filename );
                    finalPart = newPart;
                }

                // Push the part stack
                _partHierarchy.add( 0, finalPart );

                //_engine.savePart( nxPart );

     //         if (PluginMain.DEBUG) {
     //
     Application.getInstance().getGUILog().log("Updated
part: " + finalPart.getName()+ " child of " +
parent.getName() );
     //         }
```

```
                return sysmlPart;
          }

          @Override
          public Property exitExpression(Project project,
Class parent, Property sysmlExpression, NXExpression
nxExpression) throws ReadOnlyElementException {
                return sysmlExpression;
          }

          @Override
          public Class exitFeature(Project project, Class
parent, Class sysmlFeature, NXFeature nxFeature) throws
ReadOnlyElementException {
                return sysmlFeature;
          }

          @Override
          public Class exitPart(Project project, Class
parent, Class sysmlPart,
                      NXPart nxPart) throws
ReadOnlyElementException {
                // Pop the part stack
                //NXPart part = _partHierarchy.get(0);

                _partHierarchy.remove(0);
                return sysmlPart;
          }

     }
```

REFERENCES

[1] A. S. o. C. Contractors, "Reducing the Cost of Tolerances compatibility Problems," Missouri, 2006.

[2] American Concrete Institute, *Commentary on Standard Specifications for Tolerances for Concrete Construction and Materials (ACI 117R-90),*, Farmington Hills, MI: American Concrete Institute, 1990.

[3] P. C. Albert and P. L. Chan an Ada, "Key performance indicators for measuring construction success," *Benchmarking: An International Journal,* vol. 11, no. 2, pp. 203-221, 2004.

[4] R. Pogrebin and K. Zezima, "M.I.T. Sues Frank Gehry, Citing Flaws in Center He Designed," New York Times, 7 Nov 2007. [Online]. Available: http://www.nytimes.com/2007/11/07/us/07mit.html?_r=0. [Accessed June 2015].

[5] W. Dalgliesh, A. Little and C. Tucker, "Variations in Position of Columns and Slabs," in *11th Canadian Conference on Building Science and Technology*, Banff, Alberta, 2007.

[6] K. Ballast, "Tolerances In Construction: Dimensional Tolerances for Surface Accessibility," Architectural Research Consulting, 2011.

[7] P. E. D. Love, "Forensic project management: the underlying causes of rework in construction projects," *Civil Environmental Engineering Systems*, vol. 12, no. 3, pp. 207-228, 2004.

[8] P. Love, J. Edwards, J. Smith and D. Walker , "Congruence or divergence? A path model of rework in building and civil engineering projects," *ASCE Journal of Perfromance Construction Facilities,* vol. 23, no. 6, pp. 480-488, 2009.

[9] P. Love, P. Davis, J. Ellis and Cheung SO, "Dispute causation: identification of pathogenic influences," *Engineering, Construction and Architecture Management,* vol. 17, no. 4, pp. 404-423, 2010.

[10] Wikipedia, "Engineering Tolerance," May 2006. [Online]. Available: https://en.wikipedia.org/wiki/Engineering_tolerance. [Accessed 21 3 2015].

[11] P. Birkeland and L. Westhoff, "Dimensional Tolerances in a Tall Concrete Building," *ACI Journal*, vol. 68, pp. 600-607, 1971.

[12] P. Love, D. Edwards, S. Han and Y. Goh, "Design error reduction: toward the effective utilization of building information modeling," *Reserach in Engineering Design*, vol. 22, pp. 173-187, 2011.

[13] R. K. Allen, S. N. Pollalis and B. R. Schwegler, "Promise and barriers to technology enabled and open project collaboration," *ASCE Journal Prof Issues Engineering Education Practice*, vol. 131, no. 4, pp. 301-311, 2005.

[14] C. Juliana and A. Ramirez, "Construction Management/Design-Build," Lorman Seminar, 2005.

[15] F. Valdes, "Geometric Deviations and Tolerances in Construction.," Georgia Institute of Technology, Atlanta Ga, 2012.

[16] Y. Li, "The Impact of Design Rework on Construction Project Performance," University of Kentucky, College of Engineering, Lexington.

[17] N. Banaitiene and A. Banaitis, "Risk Management in Construction Projects," Business, Management and Economics, 2012.

[18] N. Banaitiene and A. Banaiti, "Risk Management in Construction Projects," in *Risk Management - Current Issues and Challenges*, 2012.

[19] National Defense Industrial Association, "21st Century Advanced Manufacturing Modeling & Simulation Roadmap Focus Area Recommendations," Advanced Manufacturing Engineering Capabilities (AMEC) Committee, 2013.

[20] E. Yares, "3D CAD World," Siemems, 7 August 2012. [Online]. Available: http://www.3dcadworld.com/how-was-the-mars-rover-curiosity-designed/. [Accessed 21 8 2015].

[21] Institution of Mechanical Engineers, "Begginers guide to measurement in mechanical engineering," National Physical Laboratory, Teddington , 2014.

[22] International Organization for Standardization, "ISO," 11 8 2015. [Online]. Available: http://www.iso.org/iso/home.html. [Accessed 14 1 2016].

[23] C. Eastman, R. Sacks, P. Teicholz and Liston Kathleen, BIM handbook: A guide to building information modeling for owners, managers, designers, engineers and contractors, John Wiley & Son, 2007.

[24] Object Direct, *sysML Modeling Language explained,* Object Direct.

[25] No Magic, *SysML Plugin User Guide 18.1,* No Magic, 2015.

[26] R. Peak, S. Friedenthal, M. Wilson, M. Bajaj and KIm, "Simulation-based design using sysml," in *INCOSE Internations Symposium* , San Diego, 2007.

[27] L. Delligatti, SysML Distilled: A Brief Guide to the System Modeling Language, Addison-Wesley Professional, Nov. 2013.

[28] J. Brown, Interviewee, *design errors vs modeling mistakes.* [Interview]. 15 2 2016.

[29] D. Ballast, Handook of Construction Tolerances, 2007.

[30] M. Bernal, J. Haymaker and C. Eastman, "On the role of computational support for designers in action," *Design Studies,* vol. 41, no. B, pp. 163-182, 2015.

[31] V. Gane and J. Haymaker, "Design Scenarios: Enabling transparent parametric design spaces," *Advanced Engineering Informatics,* pp. 618-640, 2012.

[32] R. Lopez and P. Love, "Design Errors Costs in Construction Projects," *J. Constr. Eng. Manage.,* vol. 5, pp. 585-593, 2012.

[33] J. Hymaker, P. Keel, E. Ackerman and W. Porter, "Filter mediated design: generating coherence in collaborative design," *Design Studies,* vol. 21, no. 2, pp. 205-220, 2000.

[34] G. Ameta, S. Serge and M. Giordano , "Comparison of Spatial Math Models for Tolerance Analysis: Tolerance-Maps, Deviation Domain, and TTRS," *J. Comput. Inf. Sci. Eng* , vol. 11, no. 2, p. 8, 2011.

[35] J. N.P., "Modelling and Representation of Dimensions and Tolerances: a survey," *Computer-Aided Design,* vol. 24(1), pp. 3-17, 1992.

[36] T. Kandikjan, J. Shah and J. Davidson, "A Mechanism for Validating Dimensioning and Tolerancing Schemes in CAD Systems," *Computer-Aided Design*, pp. 721-737, 2001.

[37] A. Requicha, "Toward a theory of geometric tolerancing," *Journal of Robotics Research*, vol. 2, no. 4, pp. 45-60, 1983.

[38] G. Lee, R. Sacks and C. Eastman, "Specifying parametric building object behavior (BOB) for a building information modeling system," *Knowledge Enabled Information System Applications in Construction*, vol. 15, no. 6, pp. 758-776, 2006.

[39] A. Requicha, "Part and Assembly Description Languages. 1. Dimensioning and Tolerancing," *Production Automation Project*, 1970.

[40] A. Requicha and R. Tilove, "Mathematical Foundations of Constructive Solid Geometry: General Topology of Closed Regular Sets," *Production Automation Project*, 1978.

[41] V. Srinivasan and R. Jaramaran, "Issues in Conditional Tolerances in CAD systems," in *IEEE*, St. Louis, 1985.

[42] C. Zhang and H. P. Wang, "Simultaneous optimization of design and manufacturing—tolerances with process (machine) selection," *Cirp IRP*, vol. 41(4), pp. 569-572, 1992.

[43] R. Soderberg, "Tolerances Allocation Considering Customer and Manufacturer Objectives," *Advances in Design Automation*, vol. 2, no. 65, pp. 149-157, 1993.

[44] P. M. Martin, Y. Fathi, R. O. Mittal and J. L. Cline, "Alternative Manufacturing Sequence and Tolerance build Up: A Point of View and a Case Study," *Int. J. Pord. Res*, vol. 2, no. 35, pp. 123-136, 1997.

[45] B. P. Fraticelli, E. A. Lehtihet and T. M. Cavalier, "Tool Wear Effect Compensation Sequential Tolerance Control," *Int. J. Prod. Res.*, vol. 3, no. 37, pp. 639-651, 1999.

[46] M. H. Gaddalah and H. A. EIMaraghy, "The Tolerance Optimization Problem Using an Experimental Design," *Advances in Design Automation*, 1994.

[47] V. Skowronski and J. Turner, "Using Monte-Carlo variance reduction in statistical tolerance synthesis," *Computer-Aided Design*, vol. 29, no. 1, pp. 63-69, 1997.

[48] A. Jeang, "Tolerance Chart Optimization for Quality and Cost," *International Journal of Production*, pp. 2529-2541, 1998.

[49] Y. Tseng and Y. Terng, "alternative Tolernace Alocations for Machining Parts Represented with Multiple Sets of Features," *J. Prod. Res.*, vol. 37, no. 7, pp. 1561-1579, 1999.

[50] Y. Tseng and H. Kung, "Evaluation of alternative tolerance allocations for multiple machining sequences with geometric tolerances," *int. J. Prod.*, vol. 37, no. 17, pp. 3883-3900, 1999.

[51] X. Zhao, T. Kethara and Pasupathy, "Modeling and representation of geometric tolerances information in integrated measurement processes," *Computers in Industry*, vol. 57, no. 3, pp. 319-330, 2006.

[52] B. Anselmettu and H. Louati, "Generation of manufacturing Tolerancing based on ISO Standards," *International Journal of Machine Tools and Manufacture*, vol. 45, no. 10, pp. 1124-1131, 2005.

[53] International Organization for Standarization, "Geometrical product specifications (GPS) — Geometrical tolerancing — Tolerances of form, orientation, location and run-out," ISO, 2012.

[54] American Society of Mechanical Engineers, "ASME Y14.5-2009 Standard," ASME, 2009.

[55] PivotPoint Technology Corp., "SysML Forum," 2015. [Online]. Available: http://sysmlforum.com/sysml-faq/. [Accessed 21 january 2016].

[56] S. Friedenthal, A. Moore and R. Steiner, A practical Guide toSysML, 3rd Edition, Morgan Kaufman, October 2014.

[57] C. Eastman, P. Teicholz, R. Sacks and G. Lee, "Development of a Knowledge-Rich CAD System for American Precast Concrete Industry," in *Computer Aided Design in Architecture (ACADIA)*, Muncie, Indiana, 2003.

[58] S. Azhar, "Building Information Modeling (BIM): Trends, Benefits, Risks, and Challenges for the AEC Industry," *Leadership Manage. Eng.*, vol. 11, no. 3, pp. 241-252, 2011.

[59] T. Johnson, C. Paredis and R. Burkhart, "Integrating Models and Simulations of Continuous Dynamics into SysML," in *Modelica Conference*, Germany, 2008.

[60] F. Valdes and Y. Sun, "Parametric Natural Ventilation Simulation with Real-time," in *SIGRADI*, 2012.

[61] G. Augenbroe, "The role of simulation in performance based building," in *Building Performance Simulation for Design and Operation*, ed. J.L.M. Hensen and R. Lamberts, 2011, pp. 15-36.

[62] AgileModeling.com, "The Object Primer 3rd Edition: Agile Model Driven Development with UML 2," 2014. [Online]. Available: http://www.agilemodeling.com/artifacts/stateMachineDiagram.htm.

[63] G. L. Rocca, "Knowledged-based Engineering: Between AI and CAD. Review of a language based technology to support engineering design.," *Advanced Engineering Inforamatics*, vol. 26, no. 2, 2012.

[64] M. Reichmann, P. Kuhl and P. Graf, "GenaralStore – A CASE-Tool Integration Platform Enabling Model Level Coupling of Heterogeneous Designs for Embedded Electronic Systems," in *11th IEE int. conference and workshop on the engineering of Computer-Based System (ECBS'04)*, 2004.

[65] H. Saiedian and R. Dale, "Requirements engineering: making the connection between the software developer and customer," *Information and Software Technology*, vol. 42, no. 6, pp. 419-428, 2000.

[66] C.-M. Chituc, "Requirements Engineering Reaserach and Long Term Digital Preservation," 2012. [Online]. Available: 2014-02-20].http:ht-tp://link.springer,com/content/pdf/10.1007%2F978-3-642-37804-1. pdf (2012).

[67] E. Sikora, B. Tenbergen and K. Pohl, "Industry Needs and Reserach Directions in Requirements Engineering for Embedded Systems," *Requirements Engineering*, vol. 17, no. 1, pp. 57-78, 2012.

[68] M. dos Santos Soares, J. Vrancken and A. Vebraeck, "User requirements modeling and analysis of software-intensive systems," *Journal of Systems and Software*, vol. 84, no. 2, pp. 328-339, 2011.

[69] J. Holt, S. Perry, M. Brownsword and D. Cancila, "Model-based requirements engineering for system of system," in *System of Systems Engineering (SoSE), 7th International Conference*, Genoa, 2012.

[70] D. Cuddleback, A. Dekhtyar and J. Huffman Hayes, "Automated requirements traceability: The study of human analysts," in *Requirements Engineering Conference IEEE*, 2010.

[71] O. Ingmar, "On principles for model-based systems engineering," *Systems Engineering*, vol. 3, no. 1, pp. 38-49, 2000.

[72] E. Mancin, "How Model Based Systems Engineering streamlines the development of complex systems," in *INCOSE Italian Chapter Conference on Systems Engineering (CIISE2014)*, Rome, 2014.

[73] R. Capilla, M. Ali Babar and O. Pastor, "Quality requirements engineering for systems and software architecting: methods, approaches, and tools," *Quality RE For Sys. & Architecting*, vol. 17, no. 4, pp. 255-258, 2012.

[74] A. Van Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in *Fifth IEEE International Symposium*, 2001.

[75] A. Sutcliffe, "Scenario-Based Requirements Engineering," in *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, 2003.

[76] H. J., "Coupling Simulink and UML Models," in *Symposium FORMS/FORMATS*, 2004.

[77] T. A., "Meta-models and mappings—ending the interoperability war," in *Fall Simulation Interoperability workshop* , 2004.

[78] Y. Vandeperren and W. Dehaene, "From UML/SysML to Matlab/Simulink: Current state and future perspectives," in *Design Automation and Test in Europe DATE*, Munich, 2006.

[79] L. Brisolara, M. Oliveira, F. Nascemento, L. Carro and F. Wagner, "Using UML as a Front-End for an Efficient Simulink-Based Multithread Code Generation Targeting MPSoCs.," in *Design Automation conference*, Calfornia, 2007.

[80] A. Pop, D. Akhvlediani and P. Fritzon, "Towards Unified Systems Modeling with the ModelicaML UML Profile," in *Workshop on Equation-Based Object-Oriented Languanges and Tools*, Berlin, 2007.

[81] C. Nytsch-Geusen, "The Use of UML within the Modeling Process of Modelica-Models," in *International Worshop on Equation-Based Object-Oriented Languages and Tools*, Berlin, 2007.

[82] T. Johnson, C. Paredis and J. Jobe, "Modeling Continuous System Dynamics into SysML," in *Modelica Conference*, Germany, 2008.

[83] A. Brucker and J. Doser, "'Meta-model-Based UML Notations for Domain-Specific Languages'," in *Workshop on Software Language Engineering*, Nashville, 2007.

[84] E. Huang, R. Ramamurthy and McGinnis, "System and Simulation modeling using SysML," in *Winter Simulation Conference*, Washington D.C., 2007.

[85] C. Van der Velden, "An adaptable methodology for automation application developmen," in *International Council of the Aeronautical Sciences*, Anchorage, 2008.

[86] J. Jobe, T. Johnson and C. Paredis, "Multi-Aspect Component Models: A Framework for Model Reuse in SysML," in *International Design Engineering Technical Conference & Computer Information in Engineering Conferences*, Brooklyn, 2008.

[87] Giese, Holgar and Wagner, "From model transformation to incremental bidirectional model synchronization," in *Software and System Modeling*, 2009.

[88] A. Shah, A. Kerzhner, D. Schaefer and C. Paredis, "Multi-view modeling to support embedded systems engineering in sysml," in *Graph Transformation and Model Driven Engineering*, 2010.

[89] W. Schamai, P. Fritzon, C. Paredis and A. Pop, "Towards Unified System Modeling and Simulation with ModelicaML: Modeling of Executable Behaviour Using Graphical Notations," in *Modelica Conference*, Como, Italy, 2009.

[90] C. Paredis, Y. Bernard, R. Burkhart, H. de Koning, S. Friedenthal, N. Rouquette and W. Schamai, "5.5.1 An Overview of the SysML-Modelica Transformation Specification," *INCOSE International Symposium*, vol. 722, no. 1, p. 709, 2010.

[91] M. Marchenko, G. Behrens, R. Wrobel, R. Sheffler and M. Plebow, "New Method of Visualization and Documentation of Parametric Information of 3D CAD Models," in *cadaps*, 2011.

[92] G. Rocca, "Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design.," *Advanced Engineering Informatics,* vol. 26, no. 2, pp. 159-179, 2012.

[93] G. Mosier, "Finding NIMA, An overview of NASA integrated Model-Centric Architecture project," in *Systems Engineering Seminar*, 2012.

[94] S. Herzig, A. Qamar, A. Reichwein and C. Paredis, "A conceptual framework for consistency management in model-based systems engineering," in *International Design Engineering Technical Conferences & Computer Informtion in Engineering Conference*, 2011.

[95] C. Adourian and H. Vangheluwe, "Consistency between geometric and dynamic views of a mechanical system," in *Summer Computer Simulation Conference*, San Diego, 2007.

[96] P. Hehenberger, A. Egyed and K. Zeman, "Consistency Checking of Mechatronic Design Models," in *International Design Engineering Technical Conferences & Computers and Information in Engineering Conference* , Qebec, 2010.

[97] J. Gausemier, W. Schafer, J. Greenyer, S. Kahl and P. Sascha, "Management of cross domain model consistency during the development of advanced mechatronic systems.," in *International Conference of Engineering Design*, 2009.

[98] J. Simmonds, R. Van Der Straeten, V. Jonckers and T. Mens, "Maintaining consistency between UML models using description logic," *EUZENAT and Bernard, Carre, editors, RSTI serie L'Objet,* vol. 10, pp. 231-244, 2004.

[99] T. Mens, R. Van Der Straeten and J. Simmonds, "A framework for managing consistency of evolving UML models," *Software Evolution with UML and XML*, pp. 1-31, 2005.

[100] *The Mission Hospital - st. Joseph Health System,* 2015.

[101] National Institute of Standards and Technology (NIST), NIST Handbook, U.S. Department of Commerce, 2003.

[102] American Concrete Institute, 117-10 Specification for Tolerances for Concrete Construction and Materials (ACI 117-10) and Commentary, 2015.

[103] american Society for Testing and Materials, Standard Test Methd for Determining Floor Flatness and Floor Levelness, West Conshohocken : ASTM, 2001.

[104] American Institute of Steel Construction, Code for Standard Practice for Steel Buildings and Bridges, Chicago: AISC, 2005.

[105] American Institute of Steel Construction, Standard Specification for General Requirements of Rolled structural Steel Bars, Plates, Shapes, and Sheet Piling, West Chosohocken, PA: American Society for Testing and Materials, 2005.

[106] Marble Institute of America, Dimension Stone Design Manual, Cleveland: Marble Institute of America, 2003.

[107] Indiana Limestone Institute of America, Indiana Limestone Handbook, 2002.

[108] American Institute of timber Construction, Standards for Dimensions of Structural Glued Laminated Timbers, Centenial, CO: American Institute of timber Construction, 2001.

[109] American National Standards Institute, Wood Products: Structural Glued Laminated Timber, Washington DC: American National Standards Institute, 2002.

[110] American National Standards Institute, Standard for Particleboard, Washington DC: ANSI, 1999.

[111] Kitchen Cabinets Manufacturers Association, Recomended Performance and Construction Standards for Kitchen and Vanity Cabinets, Potomac Falls: ANSI, 2000.

[112] American Architectural Manufacturing Association, Metal Curtain Wall Manual, Schaumburg: AAMA, 2002.

[113] American National Standard Institute, Dimensional Tolerances for Aluminium Mill Products, Arlington VA: ANSI, 2003.

[114] American National Standard Institute, Specifications for Interior Installations of Cementitious Backer Units, Anderson SC: Tile Council of North American, Inc, 2005.

[115] Americn Society of Testing and Materials, Standard Specification for Installation of Steel framing Members to Receive Screw-Attached Gypsum Panel Products., Chicago Il: Metal Lath/Steel framing Association, 2004.

[116] window and Door manufacturing Association, Industry specification for Architectural wood Flush Doors, Des Plaines Il: WDMA I.S..

[117] Steel Door Institute, Manufacruting Tolerances, Standards Steel Doors and Frames, Cleveland: SDI, 2000.

[118] AISC Board of Directors, "Code of Standard Practice for Steel Buildings and Bridges," American Institute of Steel Construction, Inc, Chicago, Illinois, 2005.

[119] U. Roy and B. Li, "Representation and interpretation of geometric tolerances for polyhedral objects. II.: Size, orientation and position tolerances," *computer-Aided Design,* vol. 31, no. 4, pp. 273-285, 1999.

[120] P. Hoffman, "Analysis of tolerances and process inaccuracies in discrete part manufacturing," *Computer-Aided Design,* vol. 14, no. 2, pp. 83-88, 1982.

[121] R. Hillyard and Braid I, "Analysis of dimensions and tolerances in computer-aided mechanical design," *Computer-Aided Design,* vol. 10, no. 3, pp. 161-166, 1978.

[122] V. Lin, R. Light and D. Gossard, "Variational geometry in computeraided design. Computer Graphics," *Computer Graphics,* vol. 15, no. 3, pp. 171-177, 1981.

[123] J. Turner and M. Wozny, "The M-space theory of tolerances," in *Design Automation Conference,* Chicago, 1990.

[124] S. Gupta and J. Turner, "Variational solid modeling for tolerance analysis," in *ASME International Computers in Engineering,* Santa Clara, 1991.

[125] S. Liu and Z. Dong, "A solid boundary based tolerance representation model," in *Design Automation Conference,* Scottdale, 1992.

[126] D. Whitney and O. Gilbert , "Representation of geometric variations using matrix transforms for statistical tolerance analysis in assemblies," in *International Conference on Robotics and Automation*, Atlanta, 1993.

[127] L. Rivest, C. Fortin and C. Morel, "Tolerancing a solid model with a kinematic formulation," *Computer-Aided Design,* vol. 31, no. 4, pp. 273-285, 1999.

[128] R. Jayaraman and V. Srinivasan, "Geometric Tolerancing. Part II: Virtual Boundary Requirements," *IBM Journal of Research and Development,* vol. 33(2), pp. 105-125, 1989.

[129] K. Chase, "BASIC TOOLS FOR TOLERANCES ANALYSIS OF MECHANICAL ASSEMBLIES," in *Manufacturing Engineering Handbook*, Provo, McGraw-Hill, 2004, pp. 13-26.

[130] I. I. JCGM member organizations (BIPM, "Evaluation of measurement data - Guide to the expression of uncertainty in measurement," 2008.

[131] M. Karsten and Decker, "The Monte Carlo method in Science and Engineering," *Theory and Application Institute for Applied Mathematics, University of Berne, Switzerland,* 1990.

[132] A. Sheffer, "Model Simplification for Meshing using Face Clustering," *Computer-Aided Design,* vol. 33, no. 13, pp. 925-934, 2001.

[133] Y. Lee and K. Lee, "Geometric detail suppression by the Fourier transform," *Computer-Aided Design,* vol. 30, no. 9, pp. 667-693, 1998.

[134] J. Lee and J. H. Lee, "A cellular topology-based approach to generating progresive solid models from feature-centric models," *Computer-Aided Design,* vol. 36, no. 3, pp. 217-229, 2004.

[135] C. Andujar, P. Brunet and D. Ayala, "Topology reducing surface simplification using discrete solid representation," *ACM Transactions on Graphics,* vol. 21, no. 2, pp. 88-105, 2002.

[136] N. Joshi and D. Dutta, "Feature simplification techniques for freeform surface models," *Journal of computing and Information Science in Engineering,* vol. 3, pp. 177-186, 2003.

[137] H. Zhu and C. Meng, "B-Rep model simplification by automatic fillet/round suppressing for efficient automatic feature recognition," *Computer-Aided Design,* vol. 34, no. 2, pp. 109-123, 2002.

[138] m. Rezayat, "Midsurface abstraction from 3D solid models: General theory and applications," *Computer-Aided Design,* vol. 28, no. 11, pp. 905-915, 1998.

[139] R. Donaghy, C. Armstrong and M. Price, "Dimensional reduction of surface models for analysis," *Engineering with Computers,* vol. 16, no. 1, pp. 24-35, 2000.

[140] A. Thankur and A. Banerjee, "A survey of CAD model simplification techniques for physics-based simulation applications," *Computer-Aided Design,* vol. 41, no. 2, pp. 65-80, 2009.

[141] C. Wickman, R. Soderberg and L. Lindkvist, Toward Non-Nominal Virtual Geometric Verification By Combining VR and CAT Technologies., 2001.

[142] J. Maxfield, P. Dew, J. Zhao, N. Juster and M. Fitchie , "A Virtual Environment for Aesthetic Quality Assessment of Flexible Assemblies in the Automotive Design Process," in *SAE 2002 World Congress*, Detroit, 2002.

[143] F. Lo, L. Lindkvist and R. Soderberg, " Visualization of Motion Envelope of Parts and Assemblies Based on Simulation or Measurement Data," in *ASME International Mechanical Engineering Congress and Exposition,* Chicago, 2006.

[144] Y. Kalay, Modeling Objects and Environments, 1989.

[145] N. Juster, "Modelling and representation of dimensions and tolerances: a survey," *Computer-Aided Design,* vol. 24, no. 1, pp. 3-17, 1992.

[146] T. Kandikjana, J. Shah and J. Davidson, "A Mechanism for Validating Dimensioning and Tolerancing Schemes in CAD Systems," *Computer-Aided Design,* pp. 721-737, 2001.

[147] A. Tolk, "Metamodels and Mappings –Ending the Interoperability War," in *Fall Simulation and Interoperability Workshop*, Orlando, 2004.

[148] Q. Renewables, "Solar Racking Solutions," [Online]. Available: http://www.questrenewables.com/quadpod-canopy.html. [Accessed 12th December 2015].

[149] A. Pasko, "Function-based shape modeling: mathematical framework and specialized language," in *Automated Deduction in Geometry*. *Lecture Notes in Artificial Intelligence 2930*, Berlin Heidelberg, Ed. F. Winkler, Springer-Verlag, 2004, pp. 132-160.

VITA

Francisco Valdes, Chilean, obtained his Bachelor in Industrial Design and professional degree in Industrial Design, both from Pontifical Catholic University of Valparaiso, Chile. Before moving to United Stated pursuing his Ph.D. degree with a Fulbright scholarship, Francisco worked 7 years as faculty at the School of Architecture of the Santa Maria University of Technology, where he established his interest in advanced manufacturing, digital design, and Building Information Modeling (BIM). During those years, Francisco also built a professional practice that includes projects in the Antarctic field with the Chilean Air Force and Army, and numerous industry customers. In the United States, Francisco has worked in several sustainable projects at Georgia Tech, has been instructor in the College of Architecture and Industrial Design, and also has worked in other important institutions as the Jet propulsion Laboratory of NASA, where he developed software for the systems architectures group. Currently, Francisco works as a research engineer at Georgia Tech Research Institute (GTRI). Francisco's work focuses in mechanical design and Product Lifecycle Management (PLM) with special emphasis in lean manufacturing and renewable energy systems. Also, he is currently working on the Model Based Systems Engineering (MBSE) domain, where he is developing software to seamlessly integrate mechanical CAD models into Systems Engineering (SE) models.