# Runtime Modelling for User-Centric Smart Cyber-Physical-Human Applications

by

Lorena Castañeda Bueno
B. Systems Engineering, Universidad Icesi, Colombia, 2007
B. Telematics Engineering, Universidad Icesi, Colombia, 2007
M. Informatics and Telecommunications Management,
Universidad Icesi, Colombia, 2012

A Dissertation Submitted in Partial Fulfillment of the
Requirements for the Degree of

DOCTOR OF PHILOSOPHY

in the Department of Computer Science

© Lorena Castañeda Bueno, 2017
University of Victoria

All rights reserved. This dissertation may not be reproduced in whole or in part, by photocopying or other means, without the permission of the author.

Runtime Modelling for User-Centric Smart
Cyber-Physical-Human Applications

**Supervisory Committee:**

Dr. Hausi A. Müller, Supervisor
(Department of Computer Science. University of Victoria)

Dr. Norha M. Villegas, Supervisor
(Department of Information and Communication Technologies. Universidad Icesi)

Dr. Alex Thomo, Departmental Member
(Department of Computer Science. University of Victoria)

Dr. Kin Fun Li, Outside Member
(Department of Electrical and Computer Engineering. University of Victoria)

©Lorena Castañeda Bueno, 2017
University of Victoria

## ABSTRACT

Cyber-Physical-Human Systems (CPHSs) are the integration, mostly focused on the interactions, of cyber, physical and humans elements that work together towards the achievement of the objectives of the system. Users continuously rely on CPHSs to fulfil personal goals, thus becoming active, relevant, and necessary components of the designed system. The gap between humans and technology is getting smaller. Users are increasingly demanding smarter and personalized applications, capable of understanding and acting upon changing situations. However, humans are highly dynamic, their decisions might not always be predictable, and they expose themselves to unforeseeable situations that might impact their interactions with their physical and cyber elements.

The problem addressed in this dissertation is the support of CPHSs' user-centric requirements at runtime. Therefore, this dissertation focuses on the investigation of runtime models and infrastructures for: (1) understanding users, their personal goals and changing situations, (2) causally connecting the cyber, physical and human components involved in the achievement of users' personal goals, and (3) supporting runtime adaptation to respond to relevant changes in the users' situations.

*Situation-awareness* and *runtime adaptation* pose significant challenges for the engineering of user-centric CPHSs. There are three challenges associated with *situation-awareness*: first, the complexity and dynamism of users' changing situations require specifications that explicitly connect users with personal goals and relevant context. Second, the achievement of personal goals entails comprehensive representations of user's tasks and sequences and measurable outcomes. Third, situation-awareness implies the analysis of context towards an understanding of users' changing conditions. Therefore, there is a need for representations and reasoning techniques to infer emerging situations. There are three challenges associated with *runtime adaptation*: first, the dynamic nature of CPHSs and users require runtime models to make explicit the components of CPHSs and their interactions. Second, the definition of architectural and functional requirements of CPHSs to support runtime user-centric awareness and adaptation. Finally, the design and implementation of runtime adaptation techniques to support dynamic changes in the specification of the CPHSs' runtime models.

The four contributions of this dissertation add to the body of knowledge for the development of smart applications centred around the achievement of users' personal goals. First, we propose a definition and architectural design for the implementation of user-centric smart cyber-physical-human applications (UCSAs). Our design proposes a context-aware self-adaptive system supported by a runtime infrastructure to manage CRUD operations. Second, we propose two models at runtime (MARTs): (1) our Galapagos Metamodel, which defines the concepts of a UCSA; and (2) our Galapagos Model, which supports the specification of evolving tasking goals, personal interactions, and the relevant contexts. Third, we propose our operational framework, which defines model equivalences between human-readable and machine-readable, available runtime operations and semantics, to manage runtime operations on MARTs. Finally, we propose our processing infrastructure for models at runtime (Primor), which is a component-based system responsible for providing reading access from software components to the MARTs, executing model-related runtime operations, and managing the propagation of changes among interconnected MARTs and their realities.

To evaluate our contributions, we conducted a literature review of models and performed a qualitative analysis to demonstrate the novelty of our approach by comparing it with related approaches. We demonstrated that our models satisfy MARTs characteristics, therefore making them proper models at runtime. Furthermore, we performed an experimental analysis based on our case study on online grocery shopping for the elderly. We focused our analysis on the runtime operations specified in the framework as supported by the corresponding MART (*accuracy* and *scalability*), and our infrastructure to manage runtime operation and growing MARTs (*performance*).

# Contents

# List of Tables

# List of Figures

ACKNOWLEDGEMENTS

This has been quite a journey.

Foremost, I would like to express my sincere gratitude to my supervisors, Prof. Hausi Müller and Prof. Norha Villegas, for their continuous support, guidance, motivation to carry on, and inspiration during all the time of research and writing of this dissertation. I am grateful I had the opportunity to have you as my advisors, mentors and friends during this journey. The dream team. Thank you, for your insights during our research and teaching me to find my path. Thank you for your patience, funny moments, and showing me that one of the fulfilling sides of doing research is doing it with friends. Thank you for always reminding me the importance to enjoy my PhD. Thank you for all the academic and life lessons, but most of all thank you for your friendship. You are my role models in my goal to become a supervisor some day. I will never forget to *have fun.* I feel *ready to rock and roll.*

I would like to thank Profs. Alex Thomo and Kin Fun Li for serving on my PhD supervisory committee and their valuable feedback on my work. I also thank Prof. Grace Lewis for acting as the external examiner of my thesis, and Prof. Goluskin, for serving as the chair of my final oral examination.

During my journey as a PhD student in the Rigi lab, I met people I'd like to thank. To my first Rigi crowd, Ishita Jain, Andi Bergen, Pratik Jain, Nina Taherimahsousi, Przemek Lach, and Ron Desmarais thank you, for your friendship, all the memories, academic adventures and the meaningful (and meaningless) arguments. I believe someday the Rigi lab will have the perfect layout. To the new crowd, may the journey ahead fill you with good memories too.

Thank you, to my friends and thesis buddies, Germán Poo and Miguel Jiménez, for coaching me the last months with a strict schedule to write this dissertation, proofreading my chapters, and giving me advice, encourage and hope during every written chapter. I learned to push my commits more often, not that you bugged me about it (although you did).

I would also like to thank the faculty and administrative staff of the computer science department, Wendy Beggs, Nancy Chan, Erin Robinson, Jen Knapp, Shanel Higham and Kath Milinazzo. You have contributed in many aspects to the completion of my research. Thank you for helping me throughout this journey and for taking the time to say hi to David during our many visits. Your support has never been unnoticed. Thank you to the faculty of the computer science department, especially

Profs. Ulrike Stege and Sudhakar Ganti, for giving me advice and offering a different perspective on research and life.

I would like to thank the institutions that sponsored my research. This dissertation was possible thanks to the funding by the University of Victoria, the National Sciences and Engineering Research Council (NSERC) of Canada, Universidad Icesi (Colombia), Colfuturo (Colombia), and IBM Corporation. I would like to thank my colleagues and friends from Universidad Icesi in Colombia. In particular, Prof. Gabriel Tamura, the first influence I had of being a researcher, I will always appreciate his advice in academia and life. Profs. Gonzalo Ulloa and Alvaro Pachon for their lessons and encouragement during my long academic journey since I was a young engineering student.

I am lucky to have a group of incredible friends that are family to me and have being there during many moments of this journey. My honorary sisters and brothers in Canada, Colombia and Chile, especially Angela, Ingri, Maryi, Geo, Tania, Miguel, German and Tatiana. Thank you for your support in many ways. You kept my heart and soul connected to the world all this time.

I want to thank my family. My parents who had always supported me my whole life, they are my original fan club, never giving up on me. My little sister who brings joy to my life. My aunts and grandmother who send me blessings every time we speak. Thank you, Jose, for your infrequent but lovely words, taking care of your little brother so we could have a break, and finding the ways to help around. Thank you, David, for being a great napper and learning to sleep through the night. You are the best outcome of this adventure. Thank you to my extended family for driving their support at every opportunity.[1]

Last, but not least, I want to thank my husband. I wish I could have one whole chapter just for thanking him. Thank you, for always believing I could do it, for keeping me strong, calm, and lifting me up during the difficulties of our journey. Thank you for holding the fort, and carrying my weight so many many times.

[1]Le doy gracias a mi familia. Mis papás que siempre me han apoyado durante el transcurso de mi vida, ellos son mi club de fans original, nunca perdiendo la fe en mi. Mi hermanita que siempre trae felicidad a mi vida. Mis tías y mi abuela quienes me bañan con sus bendiciones cada vez que hablamos. Gracias Jose, por tus infrecuentes pero dulces palabras, por cuidar a tu hermanito para que pudieramos tener un respiro, y por encontrar formas de ayudarnos. Gracia David, por ser tan bueno para las siestas y aprender a dormir toda la noche. Tu eres el mejor resultado de esta aventura. Gracias a mi familia extendida quienes siempre me entregaron su apoyo en toda oportunidad.

## DEDICATION

Para mi kokoro

# Chapter 1

# Introduction

## 1.1 Motivation

The internet is rapidly growing as a *socio-technical ecosystem* in which users, software and hardware interact with each other in a complex and dynamic environment [NFG+06]. Internet's growth has focused on adding sensors and actuators that connected to ordinary things enable them with computational capabilities to collect data and process information and affect things. According to the forecast released in 2016 by IHS Markit, by 2015 the internet had an estimated 15.4 billion devices, and is expected to increase to 30.7 billion in 2020 and 75.4 billion in 2025. Such estimates are based on three main factors: automation (enabling devices with computing capabilities through sensors, and actuators), integration (connecting devices), and service-oriented models.[1]

This internet's growth has also brought terms to describe the internet's ecosystems. Some familiar terms include Internet of Things (IoT), Web of Things, Industrial Internet, and the Internet of Everything (IoE). One term that has become mainstream is Cyber-Physical Systems (CPS)[2] and Cyber-Physical-Human Systems (CPHSs) to establish the human as an active component during cyber and physical interactions [SSZ+16].

In this dissertation, we focus on CPHSs. First, CPS is defined by Lee et al. [Lee10][3] as:

[1]https://www.ihs.com/Info/0416/internet-of-things.html

[2]The term was first coined in 2006 by Helen Gill at the National Science Foundation in the US. [LS15]

[3]http://cyberphysicalsystems.org

*"Integrations of computation, networking, and physical processes. Embedded computers and networks monitor and control the physical processes, with feedback loops where physical processes affect computations and vice versa."*

Later Sowe et al. described the role of humans as an active component under the term CPHSs [SSZ+16] defined as:

*"interconnected systems (computers, cyber-physical devices, and people) talking to each other across space and time, and allowing other systems, devices, and data streams to connect and disconnect."*

It is evident that nowadays people have an active role interacting with the computing and machines towards the achievement of goals. We argue that the relevance of CPHSs will increase over the next decade. CPHSs will become larger, more complex and users will be highly involved. The gap between humans and technology is getting smaller. Users constantly rely on their technology to fulfil personal goals. We are in the era where users are more than providers and consumers of technology. Users are active, relevant, and necessary components of the designed system.

However, humans are highly dynamic, their decisions might not always be predictable, and they expose themselves to unforeseeable situations that might impact their interactions with their physical and cyber elements. With the human in the loop as an element of a system, how can CPHSs understand and respond to the dynamic environment introduced by them? How can CPHSs improve users' experiences? How can CPHSs assist users in the achievement of their personal goals? We believe there is a need for a user-centric vision in the design of smart CPHSs.

The first motivation of this dissertation is towards *empowering CPHSs with user-centric capabilities at runtime*, enabling systems to understand and reason about users' relevant situations that might affect its execution, while adapting themselves under changing conditions [End95, ADB+99, CLG+09, MKS09, LGM+13]. A user-centric capability implies that the system recognises the users as first class elements of their interactions actively involved in the achievement of the system objectives. More importantly, users' changing situations affect the behaviour of the system.

Systems designed with a user-centric vision need as much information as possible about users and their personal goals. Luckily, CPHSs are constantly generating context information about the user and the execution environment [CE11, PZCG13, KFM+13, CLPS11]. However, context information might be heterogeneous, distributed, and unreadable by machines. Also, users' context is also dynamic based on users' changing situations.

Moreover, systems need supporting infrastructures with reasoning capabilities to discover and analyse context, infer situations, measure the need for adaptation, and plan activities related to the execution of such adaptations. Related approaches for self-adaptation on CPHSs address their adaptation requirements not focused on the users changing situations, but on solving concerns of software quality such as performance, reliability and flexibility [MSW16]. A close approach of using self-adaptation to react under users' changing situations, would be in the area of system requirements based on personal goals.

The second motivation of this dissertation deals with *models at runtime (MARTs)*[4] and *runtime infrastructures*, which enables CPHSs to manage up-to-date information about the system, environment and users, as well as to perform system's adaptation-related activities during execution time. More importantly, MARTs enable CPHSs to represent changing situations and context information while the system executes, empowering the CPHSs to reason and adapt on runtime information. MARTs have been the focus of research for many years, specially from the perspective of self-adaptive systems and software evolution [BBI13, BB13, BBG+13, MV13]. Since CPHSs exist in a highly dynamic and complex socio-technical environment, these are under constant change. Runtime models are necessary to represent at various levels of abstractions, current components, interactions, and pieces of information of CPHSs at execution time. More importantly, Models at Runtime (MARTs) are fundamental to support self-adaptation, which requires dynamic infrastructures to manage system's adaptations and propagation of changes across the models that represent the elements and interactions of CPHSs.

In conclusion, users have shown to be active elements with their daily applications, the boundaries that separate users from their devices are rapidly fading as the components become smarter and more knowledgeable about the users. It is pertinent that software engineers look at the role of users as first class elements of the systems that are being designed, going further when it comes to personalization, tailoring applications to fit users' dynamic lives. We envision that users will be developing their technological presence through virtual personalities. We envision, CPHSs to be

[4]Also found in literature as *Models@run.time*, M@RT, Execution models, or Models at execution time

improved with situation-aware self-adaptive system to provide users truly personalized functionalities and features.

## 1.2 Problem Statement, Challenges and Research Questions

As explained in the motivation, we identified two main fields of research namely *user-centric CPHSs* and *MARTs*. In light of this, the following two main motivations drive our research:

M1. The need for empowering CPHSs with situation awareness to understand users' context and changing situations.

M2. The need for MARTs and runtime infrastructures to represent and manage CPHSs dynamic requirements based on user-centric concerns and situations.

### 1.2.1 Problem Statement

This dissertation addresses the following research problem:

> CPHSs that assist users in the achievement of personal goals require runtime representations to understand the user's context, personal goals and situations. Moreover, these CPHSs require runtime adaptation capabilities to regulate their requirements satisfaction under unforeseeable changing situations, particularly those associated with the users. In this regard, for CPHSs to become user-centric and situation aware these systems need to be enabled to: (1) understand users, their personal goals and changing situations, (2) causally connect the cyber, physical and human components involved in the achievement of users' personal goals, and (3) support runtime adaptation to respond to relevant changes in the users' situations.

### 1.2.2 Research Challenges

To direct our challenges, we focus on CPHSs whose objectives are the achievement of users' personal goals. These CPHSs are enabled to understand users' changing situations as well as adapting at execution time when required. We posit the following research challenges and group them into two concerns: situation awareness and runtime adaptation.

**Situation-awareness**

CH1. Typical user models represent personal context. Situations are interpretations of sensed information that can be static or dynamic. Some emerging situations might affect the users' capability of achieving personal goals using their applications. Situations are complex, dynamic and can mean different thing to different users. Thus, we require specifications that explicitly connect users with personal goals and relevant context.

CH2. For CPHSs, the achievement of users personal goals using software and hardware technologies implies a set of ordered tasks towards an objective. Therefore, we require a comprehensive representation of user's tasks and sequences that include various types of actions, services, and measurable outcomes.

CH3. Situations are the result of analysing context towards an understanding of users' changing conditions, as these affect their capacity to fulfil personal goals. In light of this, we require the support of representations and reasoning techniques to infer emerging situations.

**Runtime adaptation**

CH4. To satisfy the dynamic nature of CPHSs, we require the definition of architectural models to make explicit the components of CPHSs, which include cyber, physical and human components, as well as their interactions and runtime adaptation capabilities.

CH5. To realize CPHSs oriented to the achievement of users' personal goals, we require the definition of architectural and functional requirements of CPHSs to support personalization and runtime adaptation.

CH6. To support CPHSs exposed to changing conditions of the environment and users' situations, we expect CPHSs to respond at run time to such conditions. Therefore, we require the definition of runtime adaptation techniques to support dynamic changes in the specification of the CPHSs' models.

### 1.2.3 Research Questions

The long-term goal of this research is to investigate software engineering approaches and techniques towards runtime adaptation primarily focused on the design and implementation of user-centric CPHSs. The short term goal of this dissertation is to develop and evaluate a modelling approach and an operational framework for the management of models at runtime for user-centric CPHSs.

Based on our research goals and challenges, we define the following four research questions:

RQ1. What are the [runtime] requirements for the realization of User-Centric Smart Cyber-Physical-Human Applications?

RQ2. What are the [runtime] modelling requirements to support User-Centric Smart Cyber-Physical-Human Applications?

RQ3. What are the appropriate runtime models for the implementation of User-Centric Smart Cyber-Physical-Human Applications?

RQ4. What are the runtime infrastructures required to process and evolve runtime models while maintaining the causal relations among them for User-Centric Smart Cyber-Physical-Human Applications?

## 1.3 Methodological Aspects

### 1.3.1 Research Methodology

In this research we use exploratory sequential mixed methods [Cre13] combining qualitative and quantitative approaches. Figure 1.1 depicts our research methodology in two phases: First a collection and analysis of qualitative data based on a systematic literature review [KC07][5] on modelling approaches and internet technologies that are relevant for CPHSs that assist users' to achieve personal goals. Second, the collection and analysis of quantitative data to support qualitative observations during the previous phase. In this approach, quantitative data is collected through case studies [ESSD08] and controlled experiments [JCP08]. In our methodology, case studies also provide qualitative data. For instance, a prototype implementation of our CPHSs

[5]This dissertation uses the concept of systematic literature review as a synonym for survey.

case study is used to understand the causal connection between our CPHSs models to support runtime adaptation and situation awareness.

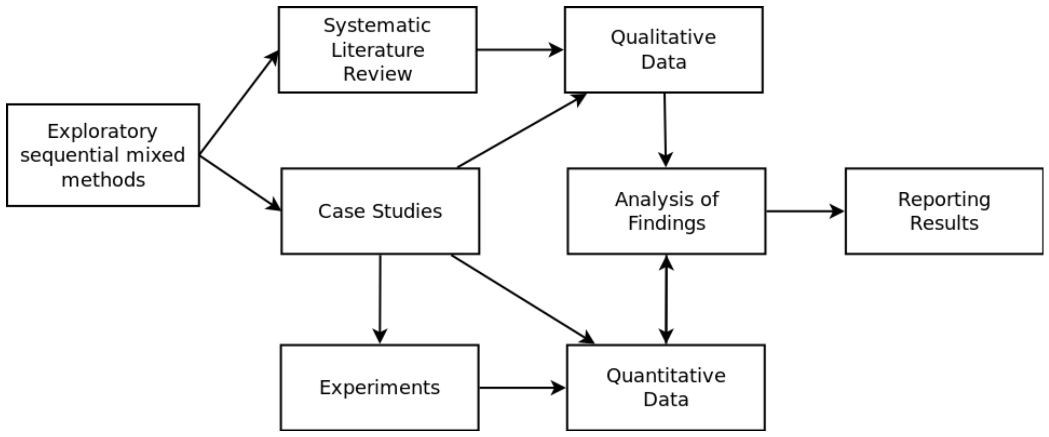


**Figure 1.1:** Research methodology

### Limitations

We recognise that the research area of user-centric CPHSs is broad and complex. This dissertation focuses on two main fields of research presented before: *user-centric CPHSs* and *MARTs*. Thus, the scope of this dissertation is related to those research fields. Our contributions, implementations and experiments, are focused on our research areas of interest.

We identify two potential major limitations: *availability of context sources*, and the *size of context data*. To overcome the limitation related to the availability of context sources and required acquisition sensors, we propose to utilize the SmarterContext monitoring infrastructure proposed by Villegas [Vil13], which provides a self-adaptive mechanism to gather context dynamically. However, in our evaluation we have a limited number of sensors to consume real-world context from third party applications (e.g., social networks and e-commerce applications) and devices (i.e., mobile and desktop). Also, we use simulation techniques for those sensors that are not readily available for our use.

## 1.3.2 Research Approach

The first step in our research was to explore the current state of *web automation and personalization* since the internet provides a close example for a user-centric CPHSs.

Personalization is a natural feature of a user-centric system. We revised academia and industry approaches to understand techniques and proposals seeking to deliver personalized experiences. In particular, we investigated selected approaches to gain insight into the capabilities or intentions to understand changing situations of the user. Later we conducted a survey on models to review related approaches to personal internet-based tasking systems. For each approach we analysed their modelling style and their relationship with our modelling requirements (i.e., personal goals, context and web-tasking).

The findings of the aforementioned exploratory studies contributed to the comprehension of users at the centre of internet interactions, the definition of fundamental elements of user-centric CPHSs (which assist uses in the achievement of personal goals), and the research gaps towards building smart software capable to understand users' changing situations. Additionally, our findings revealed that even when there are approaches to model relevant information for personal goals, there is a lack of runtime support to represent execution time challenges, such as users' situations and goals [CVM13].

The second step was to propose our models at runtime for user-centric CPHSs. To do this, we studied different types of models and focused on those suitable to represent the concepts, interactions, and elements. More importantly, we focused on software models that were capable to be represented at execution time, in a machine readable way. As a result, we defined an ontology and goal models, both of which can be transformed into graphs, which can be read and modified by software applications [CVM14a, CVM14b]

Third, we focused on a particular set of CPHSs that deal with runtime requirements and provide self-adaptive capabilities; we proposed our system architecture to realize user-centric CPHSs [CVM14b]. Our architecture comprises four layers of interaction (three based on the DYNAMICO reference model [VTM+13], and one additional for supporting models at runtime related activities) within five components (one of which is the SMARTERCONTEXT monitoring infrastructure by Villegas [Vil13]). We presented a case study for online grocery shopping that included several services, and various users affecting the personal goal of getting groceries.[6] Later we extended our scenario to be centred on people with cognitive challenges (e.g., elderly), presenting user-centric CPHSs used to assist them into maintaining an independent life style thereby improving their quality of life.

[6]www.rigiresearch.com/research/pwt/susgroceries

Fourth, we focused on the components of our approach responsible for supporting activities related to MARTs. For this, we defined an operational framework comprised of four elements: a mapping for models at runtime (notation to artefact), a catalogue of runtime model operations (model-generic to model-specific), the corresponding runtime semantics to support model operations, and the models' causal connections. For this purpose, we analysed runtime software models and their corresponding representations as graphs, and defined levels of abstraction towards their atomic structure of nodes and arcs. For each of the main runtime operations CRUD (create, read, update, and delete), we defined the propagation of changes among the models using causal connections.

Fifth, we designed and implemented PRIMOR, our software platform responsible for executing and managing runtime model-related operations using our aforementioned models at runtime and operational framework. As a case study, we integrated PRIMOR into our online grocery shopping application and run simulations in which context information would change, thus affecting users' situations.

Sixth, we performed a qualitative assessment in which we compared our approach with related approaches. As a result, our approach differs from others in the capability to represent personal goals based on the measurable outcomes of the tasks of the users, and situations based on two variables of space and time. For our quantitative assessment we evaluated our models, operational framework, and PRIMOR by measuring (1) the accuracy of the models at runtime after the execution and propagation of changes, and (2) the capability of the infrastructure to understand and reason upon changes in the context in terms of understanding users' situations. Our evaluation was an iterative process in which we refined our libraries, and operations until we reached the saturation point where both models and infrastructures behaved as expected.

## 1.4 Contributions

This dissertation summarizes the four contributions of this dissertation:[7]

[7]The supporting files are available at http://www.rigiresearch.com/research/pit

## C1: User-Centric Smart Cyber-Physical-Human Applications

We propose a definition and an architectural design for User-Centric Smart Cyber-Physical-Human Applications (UCSAs). A UCSA is an orchestrated set of cyber, physical, and human components (along with their interconnections) that assist users in the fulfilment of their personal goals. A UCSA manages the smart interaction among the components dynamically, understands and acts upon users' changing situations, and has capabilities to evolve at runtime. There are three characteristics that make a smart CPHSs a UCSA: (1) user awareness, which provides a sphere of information (static and dynamic) containing users' concerns, personal data, and relations, (2) runtime modelling support, which is the capability of the UCSA to store and manage models at runtime and (3) runtime adaptation support, which is the capability to propagate changes across the models at runtime.

Our architectural design is based on the DYNAMICO reference model [VTM+13] and comprises four software components to manage the tasks of the user towards the fulfillment of a personal goal (the tasking knowledge infrastructure, the model processor, the personalization engine, and the internet tasking effector), and a *models at runtime supporting infrastructure*, which is responsible for the access and activities related to the models.

## C2: MARTs for User-Centric Smart CPH Applications

We define two Models at Runtime (MARTs) for User-Centric Smart Cyber-Physical-Human Applications (UCSAs): (1) our GALAPAGOS METAMODEL, which defines the concepts of UCSA by abstracting the three dimensions of CPHSs as well as the smart interactions among them, and (2) our GALAPAGOS MODEL, which supports the specification of evolving tasking goals, personal interactions, and relevant contexts.

To derive our MARTs we defined the modelling requirements for UCSAs based on our analysis on the conceptual definition of UCSA and related approaches on models for internet tasks and user's personal goals. As a characteristic of a MARTs these require to be implemented in a format that can be read and accessed by software applications. Our MARTs are available in the form of graphs.

### C3: Operational Framework for Models At Runtime

We present our operational framework for MARTs in User-Centric Smart Cyber-Physical-Human Applications, which for each MARTs defines the model equivalences (human-readable to machine-readable and vice versa), available runtime operations and semantics, and causal connections with other models.

Our framework comprises four components: (1) a notation-artefact mapping, (2) a catalogue of operations, and (3) the runtime semantics and (4) causal connections. The notation-artefact mapping connects every element that is in the model notation form with its corresponding element in the software artefact form. The mapping is the main element for the translation of the MARTs in the two ways: human readable and machine readable. The catalogue of runtime operations define for every element what the limitations, restrictions, and other considerations are when performing runtime operations. It is worth mentioning that although we define three operations, the catalogue is flexible to any supported runtime operation of a MARTs. The runtime semantics are specified in a programming language and are implemented by smart infrastructures to execute the operations (i.e., software commands) at runtime. Finally, the operational framework describes the causal connections between the MARTs through the *causal links* component, which is shared by all MARTs.

### C4: Processing Infrastructure for Models at Runtime (PRIMOR)

We propose our Processing Infrastructure for Models at Runtime (PRIMOR) to manage operations on MARTs for UCSA. Primor is a component of the UCSA's models at runtime supporting infrastructure. Primor uses the operational framework as the knowledge base for its activities, supporting three main functionalities: (1) to coordinate runtime operations, (2) to orchestrate the propagation of changes among interconnected MARTs based on their causal links, and (3) to manage the MARTs synchronization keeping correspondence between a model notation and its software artefact. Primor is designed to the extensible to other domains.

### 1.4.1 Publications

- **Lorena Castañeda**, Norha M. Villegas, and Hausi A. Müller. Towards personalized web-tasking: Task simplification challenges. In Proceedings 1st Workshop

On Personalized Web-Tasking (PWT 2013) At Ninth IEEE World Congress On Services (SERVICES 2013), pages 147—153. [CVM13]

- Pratik Jain, Andreas Bergen, **Lorena Castañeda**, and Hausi A. Müller. PAL-Task chat: A personalized automated context aware web resources listing tool. In Proceedings 1st Workshop On Personalized Web-Tasking (PWT 2013) At Ninth IEEE World Congress On Services (SERVICES 2013), pages 154—157. IEEE. [JBCM13]

- Andreas Bergen, Nina Taherimakhsousi, Pratik Jain, **Lorena Castañeda**,and Hausi A. Müller. Dynamic context extraction in personal communication applications. In Proceedings 2013 Conference Of The Center For Advanced Studies On Collaborative Research (CASCON 2013), pages 261—273. IBM Corporation. [BTJ+13]

- **Lorena Castañeda**, Norha M. Villegas, and Hausi A. Müller. Personalized web-tasking applications: An online grocery shopping prototype. In Proceedings 1st Workshop On Personalized Web-Tasking (PWT 2014) At Tenth IEEE World Congress On Services (SERVICES 2014), pages 24—29. [CVM14b]

- **Lorena Castañeda**, Norha M. Villegas, and Hausi A. Müller. Self-adaptive applications: On the development of personalized web-tasking systems. In Proceedings ACM/IEEE 9th International Symposium On Software Engineering For Adaptive and Self-Managing Systems (SEAMS 2014), pages 49—54. [CVM14c]

- **Lorena Castañeda**, Norha M. Villegas, and Hausi A. Müller. Exploiting social context in personalized web-tasking applications. In Proceedings 2014 Conference Of The Center For Advanced Studies On Collaborative Research (CASCON 2014), pages 134—147. IBM Corporation. [CVM14a]

- Juan C. Muñoz-Fernández, Alessia Knauss, **Lorena Castañeda**a, Mahdi Derakhshanmanesh, Robert Heinrich, Matthias Becker, Nina Taherimakhsousi: Capturing Ambiguity in Artifacts to Support Requirements Engineering for Self-Adaptive Systems. REFSQ Workshops 2017. [MFKC+17]

## 1.5 Dissertation Outline

The remaining chapters of this dissertation are organized as follows:

**Chapter 2: Research Background**—presents four research background topics of this dissertation: (1) the *smart internet* which is a user-centred internet vision based on three principles of design (a user model, a new paradigm for sessions, and a schema of dynamic collaboration). (2) *Cyber-Physical-Human Systems (*CPHSs*)*, which is the technological domain of this research; (3) the foundational concepts of *situation awareness*, which is the ability of a system to identify, understand and reason about users' situations, and (4) the core conceptual element of this dissertation: *Models at Runtime (*MARTs*)*, which are software artifacts that represent current states of relevant elements and the system capable of evolving during execution time.

**Chapter 3: State of the Art**—presents our study of relevant approaches, focused on models, for smart internet applications. Among the variety of software models, we study modelling approaches associated with the concerns of user-centric CPHSs. We selected three modelling aspects that are relevant for user-centric CPHSs: (1) personal goals, which are relevant to the user-centric vision; (2) tasking, which are relevant to the user interactions; and (3) context, which is relevant to understand users' changing situations. To conduct our study, we followed a *systematic literature review (SLR)* as proposed by Kitchenham *et al.*[KPBB+09] and analysed 24 relevant approaches.

**Chapters 4, 5, 6 and 7**—present our four contributions respectively as outlined in Section 1.4

**Chapter 8: Evaluation**—presents our evaluation results of this dissertation in two parts: First, for our qualitative evaluation we present a comparison with related approaches of models at runtime specification, and models at runtime infrastructures; and second, for our quantitative evaluation we present our proof of concept, which includes our case studies, software-based implementation, test scenarios, simulations, results and findings.

**Chapter 9: Summary, Conclusions and Future work**—summarizes the research and contributions of this dissertation, presents the conclusions and discusses potential future work. Figure 1.2 summarizes this dissertation including the relationships among research challenges, questions, goals, contributions, publications, and evaluation methods.

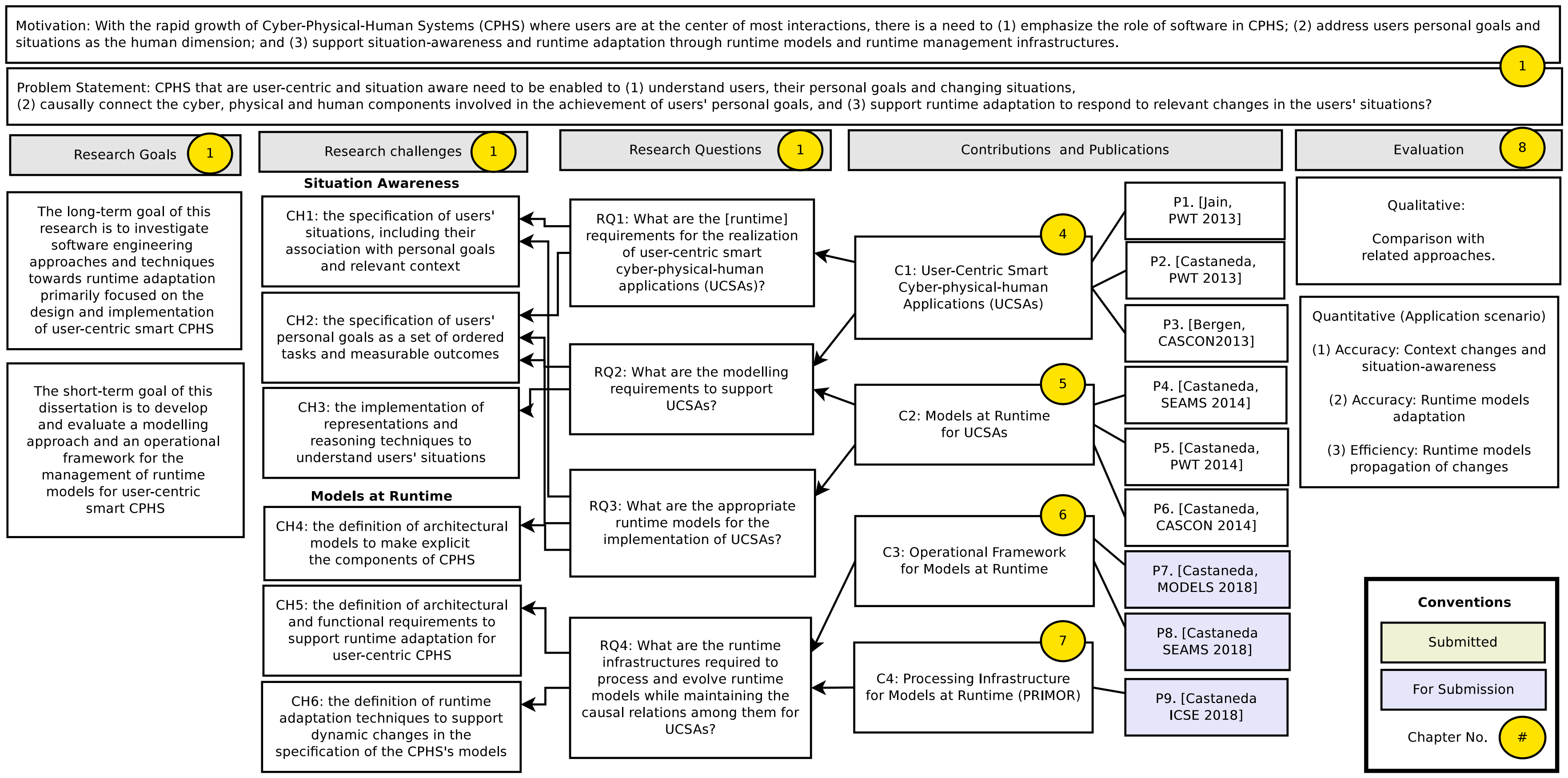

Motivation: With the rapid growth of Cyber-Physical-Human Systems (CPHS) where users are at the center of most interactions, there is a need to (1) emphasize the role of software in CPHS; (2) address users personal goals and situations as the human dimension; and (3) support situation-awareness and runtime adaptation through runtime models and runtime management infrastructures.
1
Problem Statement: CPHS that are user-centric and situation aware need to be enabled to (1) understand users, their personal goals and changing situations, (2) causally connect the cyber, physical and human components involved in the achievement of users' personal goals, and (3) support runtime adaptation to respond to relevant changes in the users' situations?
Research Goals 1
Research challenges 1
Research Questions 1
Contributions and Publications
Evaluation 8
The long-term goal of this research is to investigate software engineering approaches and techniques towards runtime adaptation primarily focused on the design and implementation of user-centric smart CPHS
The short-term goal of this dissertation is to develop and evaluate a modelling approach and an operational framework for the management of runtime models for user-centric smart CPHS
Situation Awareness
CH1: the specification of users' situations, including their association with personal goals and relevant context
CH2: the specification of users' personal goals as a set of ordered tasks and measurable outcomes
CH3: the implementation of representations and reasoning techniques to understand users' situations
Models at Runtime
CH4: the definition of architectural models to make explicit the components of CPHS
CH5: the definition of architectural and functional requirements to support runtime adaptation for user-centric CPHS
CH6: the definition of runtime adaptation techniques to support dynamic changes in the specification of the CPHS's models
RQ1: What are the [runtime] requirements for the realization of user-centric smart cyber-physical-human applications (UCSAs)?
RQ2: What are the modelling requirements to support UCSAs?
RQ3: What are the appropriate runtime models for the implementation of UCSAs?
RQ4: What are the runtime infrastructures required to process and evolve runtime models while maintaining the causal relations among them for UCSAs?
4
C1: User-Centric Smart Cyber-physical-human Applications (UCSAs)
5
C2: Models at Runtime for UCSAs
6
C3: Operational Framework for Models at Runtime
7
C4: Processing Infrastructure for Models at Runtime (PRIMOR)
P1. [Jain, PWT 2013]
P2. [Castaneda, PWT 2013]
P3. [Bergen, CASCON2013]
P4. [Castaneda, SEAMS 2014]
P5. [Castaneda, PWT 2014]
P6. [Castaneda, CASCON 2014]
P7. [Castaneda, MODELS 2018]
P8. [Castaneda SEAMS 2018]
P9. [Castaneda ICSE 2018]
Qualitative:
Comparison with related approaches.
Quantitative (Application scenario)
(1) Accuracy: Context changes and situation-awareness
(2) Accuracy: Runtime models adaptation
(3) Efficiency: Runtime models propagation of changes
Conventions
Submitted
For Submission
Chapter No. #

**Figure 1.2:** Dissertation roadmap.

Chapters 2, 3, and 9,were excluded from this map since belong to the Background, State of the art, and Summary, Conclusions, and Future work respectively.

# Chapter 2

# Research Background

In Section 1.2 we introduced our problem statement and defined that for Cyber-Physical-Human System (CPHS) to become user-centric and situation-aware these systems need to be enabled to: (1) understand users, their personal goals and changing situations, (2) causally connect the cyber, physical and human components involved in the achievement of users' personal goals, and (3) support runtime adaptation to respond to relevant changes in the users' situations. This chapter presents background on four research topics of this dissertation relevant to our problem statement: *Models at Runtime (MARTs)*, *Cyber-Physical-Human System (CPHS)*, *smart internet* and *situation-awareness*. Figure 2.1 depicts the relation between our problem statement and the background topics.

*Models at Runtime (*MARTs*)* constitutes the core conceptual element of this dissertation. *MARTs* are software artefacts that represent current states of relevant elements and the system, capable of evolving during runtime. MARTs enable systems to support dynamic behaviour and runtime adaptation. On this topic, we present the definition of the MART concept, the conceptual reference model for MART , and the reference architecture for MART systems. *Cyber-Physical-Human System (*CPHS*)* is the technological domain of this dissertation, which relates to the *smart internet* topic providing us with a vision of user-centric applications.

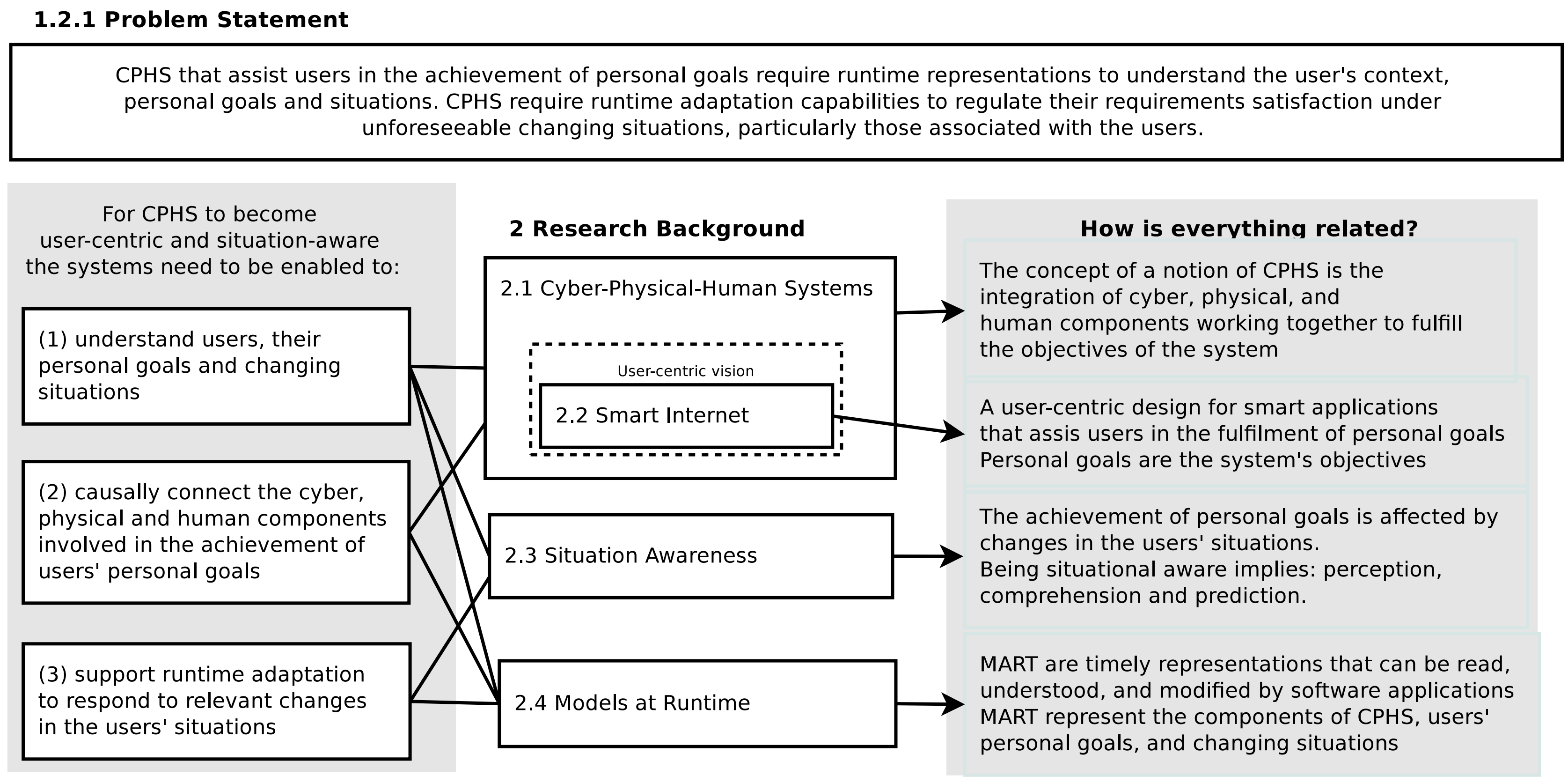

1.2.1 Problem Statement
CPHS that assist users in the achievement of personal goals require runtime representations to understand the user's context, personal goals and situations. CPHS require runtime adaptation capabilities to regulate their requirements satisfaction under unforeseeable changing situations, particularly those associated with the users.
For CPHS to become user-centric and situation-aware the systems need to be enabled to:
(1) understand users, their personal goals and changing situations
(2) causally connect the cyber, physical and human components involved in the achievement of users' personal goals
(3) support runtime adaptation to respond to relevant changes in the users' situations
2 Research Background
2.1 Cyber-Physical-Human Systems
User-centric vision
2.2 Smart Internet
2.3 Situation Awareness
2.4 Models at Runtime
How is everything related?
The concept of a notion of CPHS is the integration of cyber, physical, and human components working together to fulfill the objectives of the system
A user-centric design for smart applications that assis users in the fulfilment of personal goals Personal goals are the system's objectives
The achievement of personal goals is affected by changes in the users' situations. Being situational aware implies: perception, comprehension and prediction.
MART are timely representations that can be read, understood, and modified by software applications MART represent the components of CPHS, users' personal goals, and changing situations

**Figure 2.1:** Problem statement and research background topics of this dissertation

The concept of a notion of CPHS is an integration (mostly seen from their interactions) of cyber, physical and human elements working together to achieve the objectives of the system. The *smart internet* is a user-centred internet vision based on three principles of design (a user model, a paradigm for sessions, and a schema of dynamic collaboration). Since we are interested in CPHS that understand users' changing situation, the *situation-awareness* topic describes the elements require to enable systems with the ability to identify, understand and reason about users' situations.

In our problem statement, we described a user-centric CPHS to be highly dynamic. To support a user-centric CPHSs' runtime requirements, its MARTs have to be effective representations of the elements involved in the execution of the system, such as cyber, physical, and human components, personal goals, interactions, and the environment. We present MARTs as the last topic of this chapter, since we aim to converge on the importance of MARTs after introducing the other topics.

## 2.1 Cyber-Physical-Human Systems (CPHS)

The term *cyber-physical-human systems (CPHS)*[1] is used to describe the integration, mostly focused on the interactions, of cyber, physical and humans elements that work together towards the achievement of the objectives of the system [SSZ+16, LS15]. The description of CPHS is based on the fundamental components of *cyber-physical systems (CPS)* defined by Lee et al. with the addition of the human-in-the-loop component [Lee10, LS15].

CPHS are highly dynamic and complex, as well as being subjected to certain degrees of unpredictable behaviour from the environment and the user. These conditions create various challenges related to the management of CPS which might require runtime capabilities that enable the system to detect, monitor, understand, plan, and act upon those changes, while minimizing (and potentially eliminating) the down time of the system. We define three dimensions of CPHS: cyber, physical and human. Each dimension is connected to the other two through *smart interactions* [NCCY10b] depicted in Figure 2.2.

[1]This dissertation uses the term cyber-physical-human systems (CPHS). However most of the research was conducted with the term cyber-physical systems (CPS) while considering the human as a first class element. Although CPHS is a relatively new term, it embodies our vision of human-driven CPS, and thus we adopted the term in our dissertation.

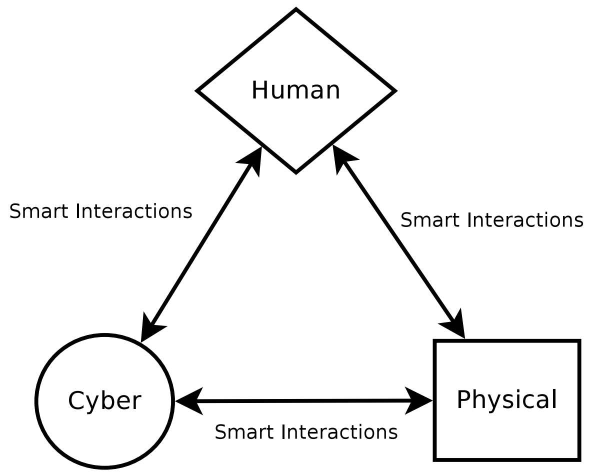


**Figure 2.2:** Three dimensions of Cyber-Physical-Human Systems

The *physical* dimension comprises all resources connected to the system through sensors and actuators. The *cyber* dimension describes all computational, networking and cloud infrastructures that communicate resources' data, processes and software. And the *human* dimension describes the human elements, as well as their situations based on their goals and context. The human dimension is especially relevant for this dissertation when CPHS' objectives are aligned with the achievement of users' personal goals.

### 2.1.1 Characteristics

The characterization of CPHS allows us to understand the application domain of this dissertation, as well as to identify runtime challenges associated with the applications, their environment and the users. Since CPHS is an extension of the definition of CPS, we take Lee's definition which is widely adopted by researchers [Lee10, SWYS11, KM06, MMS15, SSZ+16]. We present eight characteristics as follows:

(1) *Close integration of computation and physical processes.* Close integration refers to the capability of the components to communicate and exchange information. For this purpose, a component of a CPHS must have knowledge and understanding of its neighbour components. This is, awareness of the other components and how to interact with them, as well as comprehension of how the components' structure and services contribute to the overall goal of the system.

(2) *Physical components have embedded software (cyber capabilities) and limited resources.* These capabilities can be provided by adding software that exposes services and related information to cyber elements or wrapping already computational components with additional layers of software to provide higher level capabilities, or attaching sensors and actuators to physical things in order to give them a virtual presence inside the system.

(3) *Large scale and distributed networks.* Networks host cyber and physical components of a CPHS. Humans interact cyber and physical resources through networks such as the cloud. These networks are from a variety of technologies.

(4) *Temporal and spacial restriction variables and multiple scales.* A CPHS as a whole, is bounded by the constraints of spatiality and real time. Even when cyber, physical and human components as individuals, operate under different notions (and considerations) of time and space.

(5) *Dynamic configuration and organization.* The integration of CPHS components suppose a general dynamic behaviour. As a consequence, a CPHS must have monitoring and adaptive capabilities, that will allow the system to understand the changes of their environment, as well as to be able to adapt (autonomously or manually) to fit new configurations.

(6) *High degrees of automation and closed control loops.* CPHSs include autonomous components that enable parts of the system to act upon policies, and make decisions over the normal operation of the system, often at runtime.

(7) *Reliability and security are necessary.* The physical and cyber worlds are not entirely predictable therefore systems compensate with robustness. Traditional CPS research, which is focused on embedded systems, have studied reliability and security from the point of view of the digital components. As a principle, engineers tackled reliability and security with predictions (considering the limits of technological feasibility). However, modern CPHS research, working on the software component, must consider that large scale, complexity, heterogeneous components, and dynamism of the modern CPS, will limit the prediction capabilities, and increase the cost of robustness.

(8) *Human-in-the-loop.* The role of humans in CPHS is to be active elements that can help the system achieve its goals which ultimately are people's goals. The

users' context, personal goals, and situations are relevant during the design and execution of a CPHS.

## 2.2 Smart Internet

The notion of a *smart internet* denotes a user-centric internet in which services and contents are dynamically and automatically composed of multiple sources to fit users' needs [NCCY10c]. According to Ng et al., the current internet has limitations towards the user-centric vision of their approach; those limitations are: (1) lack of integration of web contents and services from the user's perspective; (2) lack of individualization to deliver content and services based on the users' current needs and situations; (3) the absence of server-initiated connections on behalf of users and with awareness of their context situations; (4) lack of the notion of service level collaboration where multiple users can work together on a service instance; and (5) lack of control by users over web entities.

Our research addresses directly the challenges associated with the user-centric design of the smart internet initiative, specifically characteristics (3) and (4). Since dynamism is expected for smart internet functionalities, our research focused on models at runtime, contributes towards the creation of responsive representations and infrastructures required to support the smart internet's dynamic behaviour.

### 2.2.1 The Smart Internet Principles

The *smart internet* vision is based on three principles: (1) an instinctive user-model that places the user at the centre of all web interactions; (2) a paradigm for sessions making them situation- and user-centric; and (3) a schema of dynamic collaboration among users [Ng10, NCCY10a, NCCY10b]. These three principles are defined as follows:

#### Instinctive user model

In the smart internet, interactions are instinctive for the user. User focused design implies three critical design considerations: First, the use of *metaphors* based on objects and operations from real world analogies as a mechanism to map users' domains and concerns in a familiar way. An example is *the like* Facebook's concept used to represent user interests and preferences. Second, the transference of *content's control* of the web to the user, in such a way that it is dynamic, adaptive and tailored for the users and their multiple internet concerns and persona. For this purpose, the content should aggregate internet resources to fit users' purpose while adapting dynamically to the users' situations. Third, a compatibility with the *users' cognition* in such way that interactions are not a mental burden for the users trying to remember, resume, and control the execution of the internet activities while trying to accomplish a goal. The core application domain of this dissertation is smart internet systems fulfilling users' personal goals. Our research on situation-awareness as the identification and representation of users' situations contributes to this user-centric model principle.

### Sessions for users and their concerns

In the smart internet, sessions derive from the perspective of users and their matters of concern, not the server, and are not bounded by users synchronous interactions. The two major implications are the needs for asynchronous interaction patterns (i.e., events and conversations) that can be shared and reused across several sites, with persistent sessions for the users' matters of concern. In this way, even if users switch concerns, services, or persona, their interactions could be resumed and sometimes executed on their behalf. This dissertation contributes on this principle by laying the grounds for composing matters of concerns from diverse sources through the implementation of models at runtime.

### Collaborative and collective web interactions

The smart internet considers users' social needs to resolve shared matters of concern by supporting close collaboration. There are two implications of this principle: the capability for users to share their matters of concern with other users; and the capability of the internet to produce information on units of matters of concern. In this case, collaboration is augmented by harvesting data from user smart interactions, and exploiting it as the collective intelligence of other users. For example, historical behaviour, user ratings, reviews, and feedback. Our research contributes to the basis of collaborative and collective interactions by representing and interconnecting user context, personal goals, and situations through models at runtime.

### 2.2.2 Personalized Web-Tasking (PWT)

Users increasingly rely on internet applications to complete a variety of every day tasks that used to be performed offline. For instance, gathering information on a research topic or a personal interest, executing transactions (e.g., for performing banking operations), and communicating with other people (e.g., through email, chat, or social networks). In the smart internet vision, *smart interactions* are instinctive web operations to assist users in the fulfilment of personal goals [NCCY10b]. Smart interactions comprise an aggregation of resources and content delivered according to users' current concerns or situations. More importantly, content and services might come from multiple sites, but presented to the user as a single unit of interaction.

In 2013, IBM launched the first workshop on *Personalized Web Tasking (PWT)*, to bring together initiatives based on the concept of smart interactions [MNS+13]. PWT proposes the automation of user-centric repetitive web interactions that assist users in the fulfilment of personal goals using internet systems [CVM13, CVM14c]. In PWT, both personal goals and internet systems are affected by unpredictable changes in user preferences, situations, system infrastructures and environments. Self-adaptation enhanced with dynamic context monitoring is required to guarantee the effectiveness of PWT systems that, despite context uncertainty, must guarantee the accomplishment of personal goals and deliver pleasant user experiences.

The initial motivation of this dissertation was the realization of PWT applications with situation-aware capabilities. Our interest concentrated on user's identity, interactions, personal goals, preferences, and context, which determine the decisions and behaviours of the web interactions.

### 2.2.3 Smart Internet Approaches

The smart internet is the internet for the users. However, transforming the internet into a user-centric model is not a trivial problem, and a clean-slate new internet is hardly an easy solution. Research has being leaning towards collecting and analysing information about users in an attempt to know and understand them.

The overall purpose of *personalization* is to tailor the applications to look and feel according to users' preferences and interests. Some examples include, the Android Swype keyboard,[2] an application that learns from the user's text inputs and predicts words to compose new phrases as the user writes on the keyboard. Other applications

[2]http://www.swype.com

analyse the user's behaviour to recommend personalized functionalities such as automatic music players. Similarly, Amazon's recommender system provides users with a sense of personalization when delivering purchase suggestions based on what similar users had purchased before.[3] Google's prediction API[4] exploits knowledge about the user to provide personalized features and can be used to develop Google applications.

There are applications to provide certain automation when performing web tasks. For example, scripting tools to automate tasks after an event, and browser-based applications to record and replay users' web interactions. However, these applications are mostly limited to register users' web actions and inputs rather than to understand users' situations and the context. Despite the efforts of commercial applications to seek automation and personalization to improve users' experiences, they still lack context-aware and self-adaptive capabilities to understand personal goals and behaviour, and adjust themselves accordingly at runtime.

In summary, academia and industry have shown significant interest in developing *smart internet* approaches. These approaches have been bound to the knowledge about users which they have been able to collect and interpret themselves. Users' *location* is being extensively exploited by a variety of applications, especially to deliver personalized functionalities. More importantly, location has being effective to infer users' situations. For instance if a user is located in a grocery store, it is fair to infer the user is achieving her personal goal for grocery shopping. Similarly, location over time has helped applications to learn about users' habits and behaviours, making easier to predict services they might need. This dissertation concentrates on the ability of smart internet applications, capable of exploiting a variety of context sources in order to understand users and their changing situations.

## 2.3 Situation Awareness

Socio-technical ecosystems, such as CPHSs, have to deal with highly dynamic context and situations. The smart internet applications, which are focused on the user, require capabilities to understand users' personal context, and to reason about their changing situations. One of the main challenges is situation-awareness support. It is important to define what it means in terms of its fundamental theory. There are two fundamental approaches: (1) the model of situation-awareness that defines the factors affecting the

[3]Amazon Patent: http://www.google.com/patents/US6317722
[4]https://developers.google.com/prediction

decision-making process that seeks an understanding of a situation [End95]; and (2) the theory of activity and situation-awareness, which provides elements from the psychology domain that we could apply in software systems [BM99].

It is a concern of this dissertation to provision user-centric CPHS with situational-awareness capabilities. The user is at the centre of the smart interactions, her context, concerns, and personal goals, are aligned with the objectives of her applications. The aforementioned model and theory of situation-awareness are the conceptual foundations of this dissertation. For instance, in our domain of user-centric CPHS, activities involve the individual's representation at runtime, while considering herself part of the system. More importantly, smart applications are capable to perceive relevant context, comprehend the user's situation, and predict how it will affect achieving her personal goal.

### 2.3.1 A Model of Situation Awareness

Endsley defined *situation-awareness* as follows [End95]:

> *"Situation awareness is the perception of the elements in the environment within a volume of time and space, the comprehension of their meaning, and the projection of their status in the near future."*

Based on this definition, the proposed model contains factors that associated with humans, such as expectations, goals, experience, training, or memory, might produce different results depending on each individual, thus are considered affecting the situation-awareness capabilities. This dissertation centres its attention towards individual factors calling them *user personal context*.

Figure 2.3 depicts the model of situation-awareness by Endsley exposing the factors affecting the decision-making process. Situation awareness follows three basic levels:

L1. *Perception of the elements in the environment:* The capability of identifying status, attributes and dynamics of the relevant components of the environment.

L2. *Comprehension of the current situation:* The significance of the perceived elements in the light of the goals of the system. Such understanding implies that the decision maker forms a holistic image of the system comprehending elements, events and their interactions.

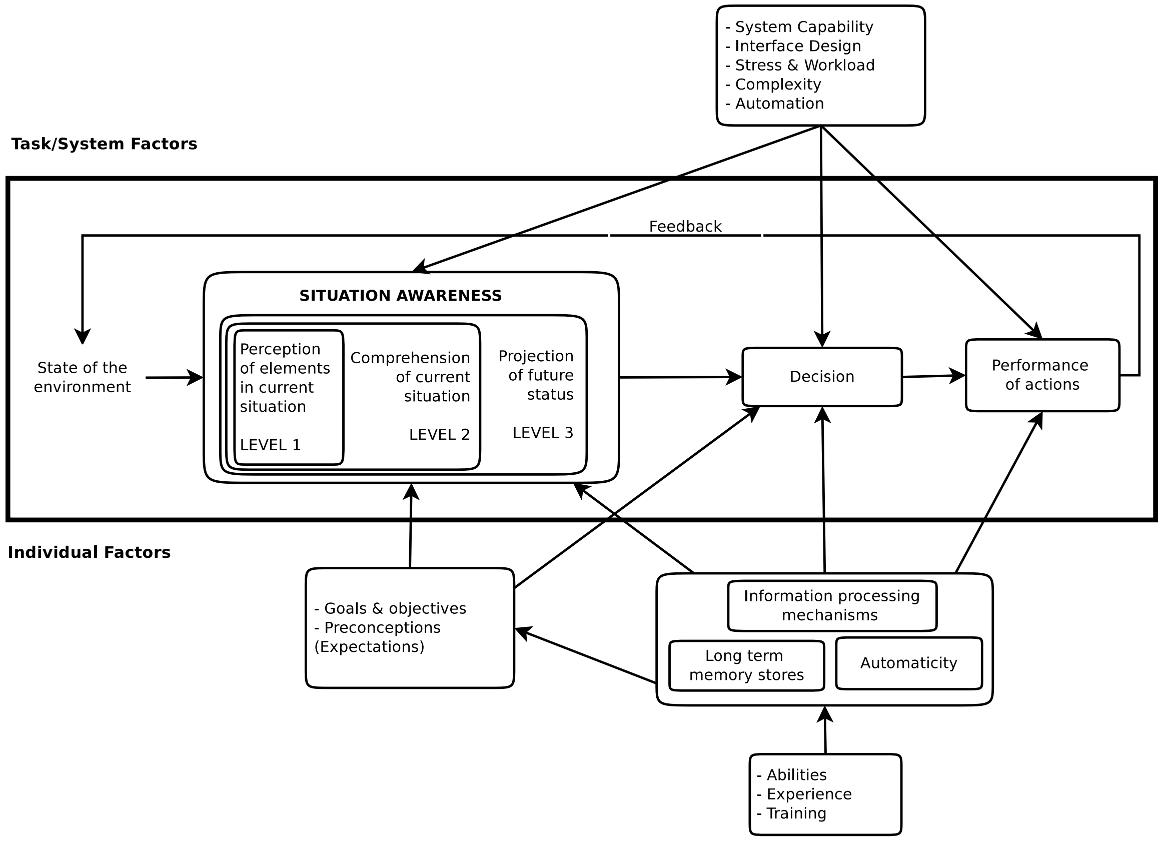


**Figure 2.3:** Model of situation-awareness in dynamic decision making [End95]

L3. *Projection of future status:* The ability to predict future actions and states of the elements of the system, based on the knowledge and comprehension of the system, and the situations.

### Elements, Time and Space

The understanding of the environment relies on the clear identification of the *elements* the system needs to perceive and understand. For this dissertation, we use *context awareness* when referring to this responsibility, and use the SmarterContext framework proposed by Villegas [Vil13] to support context monitoring and modelling. Although situation-awareness refers to a point in time, the knowledge of the system is temporal in nature, acquired over *time*, and taking into account the dynamics of the situations. More importantly, situations might have the past and future temporal aspects of the environment and their predictions. Finally, the *space* determines the scope of the elements that are relevant for the situation, even particular aspects within the elements.

## 2.3.2 A Theory of Activity and Situation Awareness

Bedny and Meister provided another definition of situation-awareness, one that includes the individuals and their goals, as well as a consideration for the dynamic behaviour [BM99]:

> *"Situational awareness is the conscious dynamic reflection on the situation by an individual. It provides dynamic orientation to the situation, the opportunity to reflect not only the past, present and future, but the potential features of the situation. The dynamic reflection contains logical-conceptual, imaginative, conscious and unconscious components which enables individuals to develop mental models of external events."*

Although Bedny and Meister's definition for situation-aware is in the domain of psychology, we can apply elements of their theory into software systems.

In Bedny and Meister's view, situation-awareness needs to be treated in concord with other behavioural concepts, specifically *activity* which comprises relevant terms such as *goal, meaning*, and *sense*. In fact, the basic structural components of activity are goals, as the ideal representation of the future results of the activity. To achieve a goal, there is a logically ordered system of actions. More importantly, an individual may vary her behaviour over a wide range while maintaining the same goal [BM99]. In is worth noting that the relationship between Bedny and Meister's definition of situation-awareness and the concept of personalized web-tasking for the smart internet. Bedny and Meister argue that Endsley's definition treats the user as a box of information, with no consideration for the decision making and execution processes.

In Bedny and Meister's description, situational-awareness implies an important mechanism called *reflective-orientational function*. Reflection is the representation of reality, incorporating the notions of goals, significance, motives, and mental and behaviour actions. Figure 2.4 depicts the process of reflection; the representations are the result of the reflected object featuring the reflected system as well [BM99].

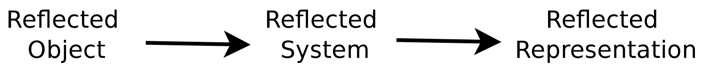


**Figure 2.4:** Reflective functions scheme [BM99]

In software systems *dynamic reflection* can be implemented through Models at Runtime (MART ), which enable systems with self-awareness capabilities.

## 2.4 Model At Runtime (MART)

Traditionally, a *software model* is an abstraction of a system, often associated with design time activities such as documentation and analysis. Unfortunately, *models at design time* might get lost in the documentation of the system, becoming obsolete over time as the software and its relevant elements change. *Models at runtime (MART)* provide up-to-date information about the system and its environment, and can be manipulated and adapted at execution time [BBF09]. More importantly, MARTs are fundamental to support *dynamic reflection*.

Multiple researchers have reported MART as a promising mechanism to monitor and verify different aspects of the behaviour and structure of the software at execution time. For instance, uses of MART to monitor and verify user's changing situations [SBCC13], adaptation goals for self-adaptive systems [AP12, SZF+14], changes in the software requirements [RCBS12], and the management of dynamic runtime environments such as the cloud [ZCZ+13].

There is also research on using MART in self-adapting systems, including simulating runtime environments, monitoring, policy checking, error handling, and supporting systems adaptation requirements [MV13, SZ13]. MART implementations have focused on different challenges: dealing with runtime concerns for complex systems [ABCF12]; and developing techniques and methods to support dynamic systems, modelling languages, and reasoning infrastructures, such as LoREM [GSB+08], GORE [SSLRM11], and SM@RT [SHC+05].

A key aspect of a CPHS is to understand users' changing situations, therefore by providing and exploiting MARTs, the system is capable of representing, monitoring, and exploiting context effectively.

### 2.4.1 Definition

Early definitions of the MART notion are based on the expectations on the model during the execution of the system. First, Blair et al. defined a MART as [BBF09]:

> *"a causally connected self-representation of the associated system that emphasizes the structure, behaviour, or goals of the system from a problem space[5] perspective."*

[5]The problem space refers to the application domain, specifically the information related to the business logic.

Morin et al. then defined a MART in terms of what MART must provide [MBJ+09]:

> *"A MART must provide a high-level basis for reasoning efficiently about relevant aspects of the system and its environment and offer enough details to fully automate the dynamic adaptation process. It is possible to make the design specifications evolve at any time, before initial deployment or while the system is already running."*

Later, Bencomo et al. refined the previous definitions as follow [BBI13]:

> *"MARTs are abstract representations of a system, including its structure and behaviour, which exists in tandem with the given system during the actual execution time of that system. Furthermore, these models should be causally connected to the system being modelled, offering a reflective capability."*

The evolution on the definition of a MART has refined the expectations on the model by exposing four important properties: (1) *representation*—a MART represents the system's complete environment, possibly more than one MART per system; (2) *availability*—a MART is accessible at runtime by the system; (3) *causal connectivity*—a MART and the system must be causally connected; and (4) *evolution*—a MART might evolve at runtime. We now elaborate the characteristics of these properties in detail.

**Representation**

The variety in software models responds to the necessity of representing diverse perspectives of the software at different levels of abstraction. Thus, *structural models* describe the form and construction of the software; *behavioural models* describe control settings and signals of the system; *sequence models* describe the actors of the system as they interact with the system; and *interaction models* describe the various components of the software internal and external, their communication and data flow. For instance, structural models include classes, components, functions or services; behavioural models include function calls or state transitions; and sequence models imply requests and responses. With software models, there is a possibility to represent high- and low-levels of abstraction. This is, while some models describe the interactions of the users with the system (usability models), other describe the information as units of data used in the system (context models).

While the models described above correspond to traditional models at design time, these can be augmented to become models at runtime. For instance, structural models with information that describe time and space constraints might represent the evolution of the software as it changes from one version to the next. More importantly, information about external systems such as, location, services, data exchange, authentication, and events, that interact with the system, along with time and space information, become a representation of up-to-date interactions of the system, which are characteristics that belong to the execution environment.

In this sense, MART representation is not bounded by the information that software engineers conceive of the system at design time, yet they are capable of representing all elements that interact with the system, as these emerge while the system executes.

### Availability

Models at design time are used to assist the communication among people involved in the development process such as engineers, developers, users, managers, and clients. Most of these models are documents in a human-readable language. In contrast, MARTs are used to assist the communication among software systems. Thus, availability implies that MART realizations have to be in a machine-readable form. Although there is research to translate model-to-model and model-to-code languages[6] and tools,[7] these are not sufficient to create general software artefacts that can be read and understood by software the way humans do with models at design time.

Moreover, software that can access MART , can also read, understand and perform operations over them. For instance, reading the structural model gives the system an understanding of its own components, where they are, and how connected they are. Additionally, if the structure of the system changes, the model could be manipulated by adding or removing entities, connections, attributes, or functionalities, to represent the current structure of the system. As a consequence, MARTs become not only

[6]ATL (https://projects.eclipse.org/projects/modeling.mmt.atl), QVTd (https://projects.eclipse.org/projects/modeling.mmt.qvtd), QVTc, QVTr, QVTo (https://projects.eclipse.org/projects/modeling.mmt.qvt-oml)

[7]Epsilon (http://www.eclipse.org/epsilon), Viatra (https://eclipse.org/viatra), and Xtend (http://www.eclipse.org/xtend)

elements of communication, but knowledge for the software systems helping them to understand themselves and their own environment.

### Causal connection

Models at design time are connected with the running software by the understanding of the humans that are reading the models. As documents, models at design time can describe model-software connections using notation or plain words. In some cases, with the help of software tools, models can be *connected*—at a descriptive level—with the code. In contrast, MARTs are causally connected. This is, that by being software artefacts themselves MARTs are a reflection of their code. When the code changes, the system changes and vice versa. Furthermore, MARTs might be causally connected with other MARTs. There are three activities associated with such reflection: (1) *introspection* is the ability of the system to inspect its code; (2) *code generation* is the capability to generate just-in-time new code; and (3) *intercession* is the ability to modify its code [AGJ+14].

### Evolution

Models at design time are likely to become obsolete as the system evolves. Modern software systems are highly dynamic, complex and exposed to unexpected changes. MARTs are software artefacts living in the same ecosystem of the realities they are representing. As the elements of the ecosystem evolve, MARTs are available to the applications through software infrastructures that can read, understand, and modify the models. More importantly, causal connections strongly link models with other models and code, thus MARTs evolve as well.

Beyond the representations and the techniques towards evolving MARTs, the validation and verification of the MART required to guarantee their representations of the reality have been discussed by related approaches [CEG+14, Caz14, GBP+14, TS14, MV13]. We recognise that the challenge of MART evolution is not trivial and requires the representation and instrumentation of MART adaptations and causal connections before moving to the fidelity aspects of evolution. This dissertation seeks to contribute to the body of knowledge of MART adaptations as a step towards trustful evolution.

### 2.4.2 Reference Models

The research of MART is interesting for the development of self-adaptive systems, and for this dissertation. There are key components to enable self-adaptive systems with the runtime capabilities to adapt at execution time, at different levels of the system. In particular, there are two reference models of interest to this dissertation: (1) the conceptual reference model of M@RT (i.e., MART ) which presents a four-level approach to describe how MARTs relate to the system and the environment [BFT+14], and (2) the reference architecture for MART systems, which proposes a generic MART framework that can be implemented in a variety of domains [AGJ+14].

#### A Conceptual Reference Model for M@RT

Bennaceur et al. proposed a conceptual M@RT reference model as MART research framework [BFT+14] as depicted in Figure 2.5. The concepts and terminology of this reference model applies to a variety of adaptive software systems, including open, distributed, and embedded systems (e.g., cyber-physical systems) and cloud-based systems.

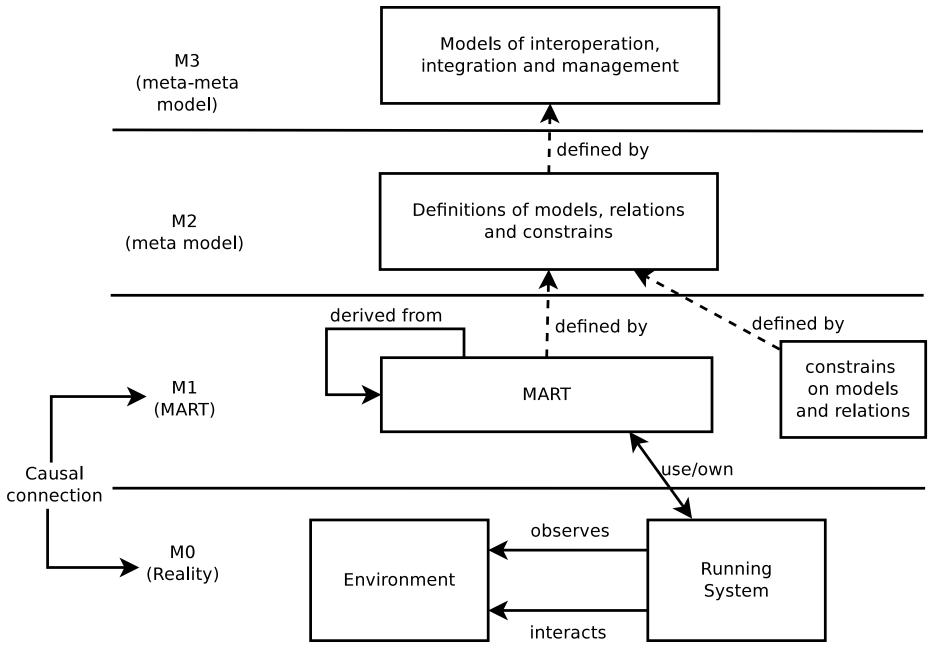


**Figure 2.5:** A Conceptual Reference Model for M@RT [BFT+14]

Bennaceur et al. structured their reference model into four levels:

M0. This level contains the *running system* and the *environment*. Here, the system observes and interacts with the environment. The running system consists of two major parts: (i) the *application*, the one responsible for delivering the functionality; and (ii) the *runtime platform*, the infrastructure in which the application runs. When it comes to representation, it is important to note that the running system might have adaptive capabilities contrary to the environment, to which the system has no control.

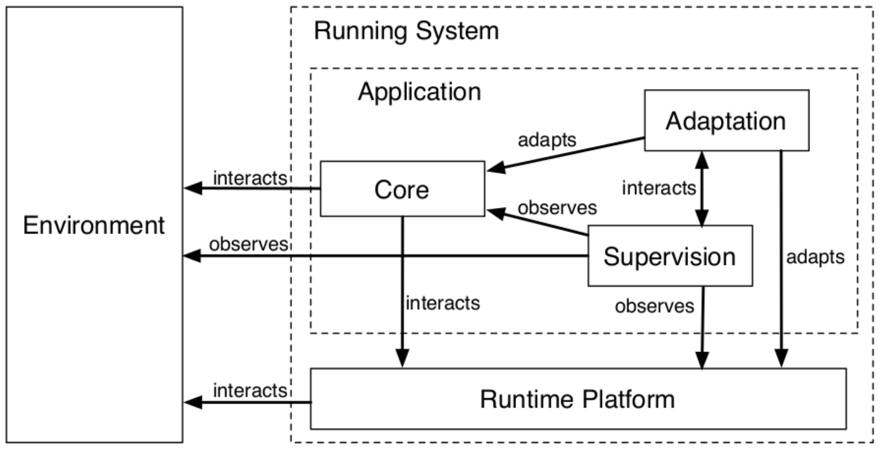


**Figure 2.6:** M0 Level [BFT+14]

Figure 2.6 shows a detailed view of the M0 level. The application part of the running system comprises three sub-components: (i) the *core* (functional requirements); (ii) the *supervision* (model-driven monitoring of the system and the environment); and (iii) the *adaptation* (i.e., reasoning, planning and enforcing adaptations on the system).

M1. This level contains the models at runtime, relations, and constrains and might have a variety of models to represent different parts of the system, as well as levels of abstraction. Models might be derived from other models or being dynamically composed by models defined within M1. As mentioned in Section 2.4.1, MARTs are causally connected with their code. In M1, models are connected with M0 through *events* handled by the supervision component; and *change actions* enacted by the applications component. These causal connections allow models in M1 to reflect the running system of M0.

M2. This level (meta models) contains the languages (i.e., syntax and semantics) that are used to create the models in M1. The use of meta-models facilitates the engineering reuse of model transformation techniques (e.g., transform a model into code).

M3. This level is the meta-meta modelling, which defines the models for interoperation, integration and management of the models in M2 and M1. The use of meta-meta-models eases the integration of various modelling languages and specifications during the design of processing activities.

### A Reference Architecture for Models@run.time Systems

Aßmann et al. proposed the reference architecture for MART systems depicted in Figure 2.7, which provides a generic framework for MART that can be implemented in a variety of domains [AGJ+14].

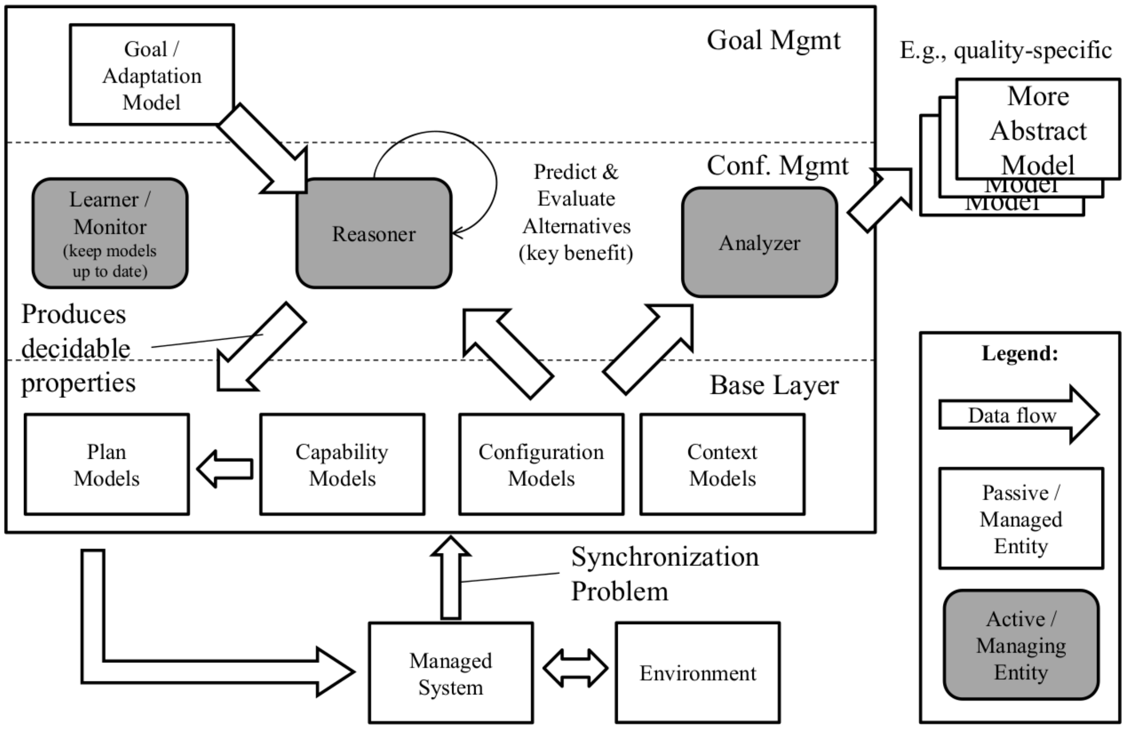


**Figure 2.7:** Reference Architecture for models@run.time Systems [AGJ+14]

It is important to note, that a MART always interfaces with the *managed system*, and through it with the *environment*. A MART system comprises the three layers described as follows:

**Base Layer.** In this layer are the abstractions of four specific aspects for the system: (1) *context models*, which represent relevant information about the observable state of the environment; (2) *configuration models*, which are representations of the configuration of the managed system; (3) *capability models*, which describe the services or features that can affect the managed system, as well as its actuators; and (4) *plan models*, which describe the set of actions to carry out adaptations on the system.

**Configuration Management Layer.** In this layer are the software entities of the system that use the models at the base layer. There are three components: (1) the *reasoner*, which evaluates future configurations of the system, by using the information on the context and configuration models; (2) the *analyser*, which detects if the system is no longer in compliance with the objectives, and realizes the bridge between the MART at different levels of abstractions; and (3) the *learner (optional)*, which is responsible for keeping the model and the system synchronized, as well as for detecting if the reasoner's predictions are beneficial in the long run.

**Goal Management Layer.** In this layer are the goal models of the system, which are used by the reasoner to predict configurations that fulfil the specified objectives. These goal models should be able to change over time as a result of changes in the context or the requirements.

The architecture presented above is a reference model for MART systems which are systems themselves. Thus, MART systems can monitor and control other MART systems, one MART system becomes the managed system.

## 2.5 Chapter Summary

This chapter presented four research background topics that are relevant for this dissertation: *Cyber-Physical-Human System (*CPHS*)* (cf. Section 2.1), *Smart Internet* (cf. Section 2.2), *Situation Awareness* (cf. Section 2.3), and *Model at Runtime (*MART*)* (cf. Section 2.4).

Figure 2.1 depicts the connection between our research background topics and problem statement. *Cyber-Physical-Human System (*CPHS*)* is an integration of cyber, physical and human elements working together to achieve the objectives of the system, and constitutes the technological domain of this dissertation. CPHS comprises three dimensions: The *physical* contains the resources connected to the system; the *cyber* includes software and networks; and the *human* includes the users' context, goals, and situations. The human dimension is specially relevant for this dissertation, since we focus on CPHS that assist users in the achievement of personal goals. We described eight characteristics for CPHS: (1) close integration of computation and physical processes, (2) physical components have embedded software and limited resources, (3) large scale and distributed networking, (4) temporal and spacial restriction variables and multiple scales, (5) dynamic configuration and organization, (6) high degrees of automation and closed control loops, (7) reliability and security are necessary, and (8) human in the loop.

The *smart internet* is a representative case of a user-centric CPHS. It features a user-centric internet vision based on three design principles: (1) an instinctive user-model that places the user at the centre of all web interactions; (2) a new paradigm for sessions making them situation- and user-centric; and (3) a schema of dynamic collaboration among users. This dissertation relates to the user-centric vision of the smart internet which describes the desired structure and behaviour of systems that assist users in the achievement of their personal goals.

We presented fundamental *situational-awareness* approaches: (1) the model of situation-awareness that defines the factors affecting the decision-making process that seeks an understanding of a situation; and (2) the theory of activity and situation-awareness, which provides elements from the psychology domain that we could apply to software systems. For our vision of user-centric CPHS, situational-awareness is the capability of understanding users' personal context, and to reason about their changing situations.

Finally, we presented *Model at Runtime (*MART*)* which are software artefacts that represent current states of relevant elements and the system, capable of evolving during execution time. *MART* constitutes the core conceptual element of this dissertation. MARTs are used to represent the elements of user-centric CPHS including cyber, physical and human components, personal goals, users' context and situations. Furthermore, MARTs are enabled to support adaptations, which allow CPHS to keep updated information about the current state of their elements. We presented two MART reference models that are of interest of this dissertation: (1) the conceptual reference model of M@RT, which presents a four-level approach to describe how MART relate to the system and the environment [BFT+14], and (2) the reference architecture for MART systems, which proposes a generic framework of MART that can be implemented in a variety of domains [AGJ+14].

# Chapter 3

# State of the Art on Models for User-Centric CPHS

In Chapter 2, we presented the conceptual foundations of this dissertation and featured MART as our core conceptual topic. MARTs are used to represent elements of user-centric CPHS, including cyber, physical and human components, personal goals, users' context and situations. Furthermore, MARTs are enabled to support adaptations, which allow CPHS to keep updated information about the current state of their elements. Also, MARTs are connected to the system and the environment; therefore, the realities that MART represent are within those two groups.

This chapter presents our study of modelling approaches for user-centric CPHS. Among the variety of software models, it is necessary to study modelling approaches associated with concerns relevant for user-centric CPHS. We selected three modelling aspects: (1) personal goals, which are related to the user-centric principle; (2) web tasking, which is relevant to the definition of smart interactions; and (3) context, which is important to understand users' changing situations.

This chapter is divided into three parts: First, we describe the methodology of our study and report on our findings. Second, we discuss our findings in light of our selected modelling aspects. Finally, we summarize this chapter.

## 3.1 Methodology

Personal goals, web tasking and context have been addressed separately by different authors in diverse application domains. For our study, we followed a *systematic* *literature review (SLR)* as proposed by Kitchenham *et al.* [KPBB+09]. Following this methodology, we searched academic publications using Google Scholar, IEEE Explorer, ACM Digital Library, SpringerLink, Science Direct, and Elsevier. We used the following keywords for our search strings: *user model, web-task model, task model, goal-oriented models, context model, web-service models, task execution.*

We subtracted relevant literature and sources using a three-layer filter: (1) we removed approaches that did not have the same *conceptualization* for our three concerns; (2) we narrowed the number of sources to keep only those in reference to the *representation* aspects of user, web-task, context, and user's goals; and (3) we kept *focus* selecting sources that exposed clear and sufficient information about our concerns as important elements of their approach.

As a result, we surveyed the 24 relevant approaches summarized in Table 3.1. For each approach, this table captures a description approach and its modelling style. We marked with an X the columns of our three concerns where the approach focuses.

**Table 3.1:** Modelling approaches versus modelling requirements.
Legend: *G: Personal Goals, C: Context,* and *W: Web-tasking*

| **Approach** | **Description** | **Modelling style** | **G** | **C** | **W** |
|---|---|---|---|---|---|
| Bolchini [BM03] | Uses task analysis to create a decomposition of a task into finer subtasks that can be sequenced | Goal models, AWARE goal-oriented notation (i* framework) | X | | X |
| Brezillon [Bré07] | A context-based graph to represent context in human tasks | Contextual graph (Formalism to problem solving) | | | X |
| Carberry [Car88] | Representation of the user's plan | Formal (STRIPS) | X | | |
| Carberry [Car88] | Representation of the user's goal | Tree (TRACK from IREPS system) | X | | X |
| Chopra [CDGM10] | Service oriented application modelling for business-IT gap | Conceptual model | X | | |
| Cui [CLWG04] | Conceptual definition of a web service | Conceptual model | | | |

| **Approach** | **Description** | **Modelling style** | **G** | **C** | **W** |
|---|---|---|---|---|---|
| Dix [Dix08] | Data, Action and Context, to infer and automate user's tasks | Personal ontology (user's context), HTA (tasks) | | X | X |
| Giersich [GFF$^+$07] | Representation of the tasks executed by the user | Dynamic Bayesian Networks (DBNs) | | | X |
| Goschnick [GBS08] | Representation of the tasks in video games computer-player interactions | Agent-Oriented (AO) architectures | X | | |
| John [JVM$^+$02] | Create CPM-GOMS models based on hierarchical task decomposition | Goals, Operators, Methods, and Selection (GOMS), Apex (A cognitive modelling tool) | | | X |
| Kim [KKJ13] | Automatic web services composition using Open APIs | Ontology for Open APIs, using RDF and OWL | | | X |
| Klug [KK05] | Using CTT describes task states and transitions | ConcurTaskTree (CTT) | | | X |
| Liaskos [LMSM10, LLJM11] | Include preferences into the goal models for requirements engineering | Framework (PDDL), Goal models, Hierarchical Task Network (HTN) | X | | X |
| Liaskos [LLJM11] | Graphical representation of goals, task, associations. A numbered sequence of tasks to create a path of interactions | Graph | X | | X |
| Mylopoulos [MCY99] | Define and analyse Non- and Functional reqs. | Goal models. AND , OR operators | X | | X |

| **Approach** | **Description** | **Modelling style** | **G** | **C** | **W** |
|---|---|---|---|---|---|
| Souchon [SLV02] | A Formal notation of a task model to support variations depending con different context of use | Formal notation of a task, ConcurTaskTree (CTT) | | X | X |
| Stoitsev [SS08] | Abstraction for high-level interactions in BTM and task-centric roles | Business Task Management (BTM) | | | X |
| Thom [TLI+11] | Version of Workflow Activity patterns in the BPMN | Business Process Modelling Notation (BPMN) | | | X |
| Villegas [Vil13] | Representation of dynamic context | (SmarterContext) Ontology | | X | |
| Wilson [Wil99] | Model of Information Behaviour to represent the information seeking | Model of Information Behaviour | X | | X |
| Yu [Yu93] | Representation of the user's interest and intentions | Goal models | X | | X |
| Yu [YM94] | Actor-Dependency model | Goal models | X | X | X |
| Yu [YLL+08] | Variability of the system through the exploitation of the OR element in the graph | Graph | | | X |

## 3.2 Findings

The following sections discuss our findings based on our three concerns for user-centric CPHS.

### 3.2.1 Personal Goals

The user model concern was been addressed for almost three decades. In 1988 Carberry applied a task planning system that uses information-seeking dialogues [Car88]. In her approach, a user model provides information for the system to infer the meaning of a user's plan and goals and allow a non-interrupted and robust communication. The most important part of this model is the representation of the user's plan and goals as understood by the system, called by the author as *the context model*.

Carberry argues that the system must know about the purpose of the user and the plans' domain to accomplish its goal. She represents a *plan* using an extension of the STRIPS formalism as a structure of applicability conditions, preconditions, body, and effects. An *agent* is in charge of executing a plan. Applicability conditions and preconditions must exist before the execution of the plan, the body contains subgoals, and the effects are the results of the execution. Figure 3.1 depicts an example of a planning model for a learning course material.

```
Learn-Materia(_agent:&PERSON, _sect:&SECTIONS, _syl:&SYLLABI)
  Plan-Body:
    Learn-From-Person(_agent:&PERSON, _sect:&SECTIONS, _fac:&FACULTY)
      where Teaches(_fac:&FACULTY, _sect:&SECTIONS)
    Learn-FromText(_agent:&PERSON, _txt:&TEXTS)
      where Uses(_sect:&SECTIONS, _txt:&TEXTS)
  Effects:
    Learn-Material(_agent:&PERSON, _sect:&SECTIONS, _syl:&SYLLABI)
```

**Figure 3.1:** Carberry 's plan model example for Learning Course Material using an extension of the STRIPS formalism [Car88]

Carberry's context model is represented by a tree, in which each node corresponds to a goal inferred by the system, and the active path is the sequence from the root to the current node of focus, depending on the user's course of decisions. Each leaf in the tree generates a new set of context, hence a new set of candidate nodes. Figure 3.2 shows an example of the context model tree for the user's goal of getting a major in computer science.

Goal models have been used to represent users' interest and intentions. Traditionally, goal modelling focuses mostly on expressing the steps of the process required to achieve certain objectives. Yu et al. have focused on requirements engineering as one domain for a user to express the intentions of the system behaviour [Yu93]. In their first approach, Yu et al. argued the existence of dependency of agents, and defined three types of operators for those dependencies: goal-dependency, task-dependency,

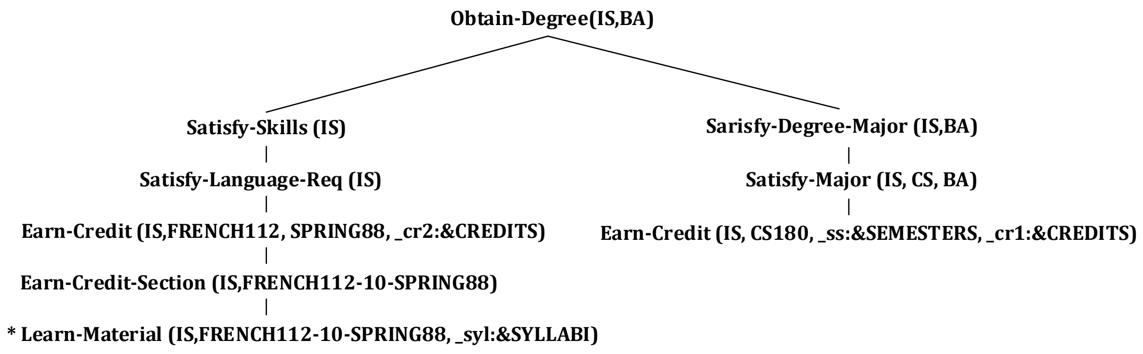

Obtain-Degree(IS,BA)
Satisfy-Skills (IS)
Satisfy-Language-Req (IS)
Earn-Credit (IS,FRENCH112, SPRING88, _cr2:&CREDITS)
Earn-Credit-Section (IS,FRENCH112-10-SPRING88)
* Learn-Material (IS,FRENCH112-10-SPRING88, _syl:&SYLLABI)
Sarisfy-Degree-Major (IS,BA)
Satisfy-Major (IS, CS, BA)
Earn-Credit (IS, CS180, _ss:&SEMESTERS, _cr1:&CREDITS)

**Figure 3.2:** Carberry's Context model tree build using TRACK; a component of the IREPS system to infer user's underlying task-related [Car88]

and resource-dependency. Yu et al. proposed and fourth dependency to express the *satisfaction* concerning *how* the goal was achieved.

Yu presented his initial approach using an extension of the Requirements Modelling Language (RML) expressing a conceptual *plan* as a set of activities required to achieve a goal. His approach includes the existence of relationships among a goal and subgoals. Later, Yu and Mylopoulos proposed the Actor-Dependency (AD) model based on nodes *(actors)* and links *(dependencies)* to represent the actuators and their relationships in a process while achieving a goal. They extended this model to introduce the concept of understanding the intentions behind those steps represented as goals in the dependencies. This modelling defines relevant characteristics of both dependencies (i.e., types and strengths) and actors (i.e., role, position, agent and associations) [YM94].

The AD model describes for main types of dependencies: *goal, soft-goal, task,* and *resource,*; and two types of strengths: *open* and *critical.* An actor can be either in the role of a *depender* or a *depdendee.* Figure 3.3 depicts a graphical representation of the AD model for a simple software project. For example between the *Programmer* and the *Tester* actors, there are some dependencies—for instance, the task-dependency *Test [Module]* implies the testing specific task links the two actors, same as the other two resources: the *test plan* and the *code.* The *Project manager* has a goal dependency with the *Programmer* through the goal *Completed [Implementation]* Similarly there are two soft goals between them: *On Schedule [Implementation]* and *Advancement [Career].*

We identify three dependency types that are relevant to user-centric CPHS: (1) *Goal*, when a task depends on the outcome of another to fulfil its goal; (2) *Task*,

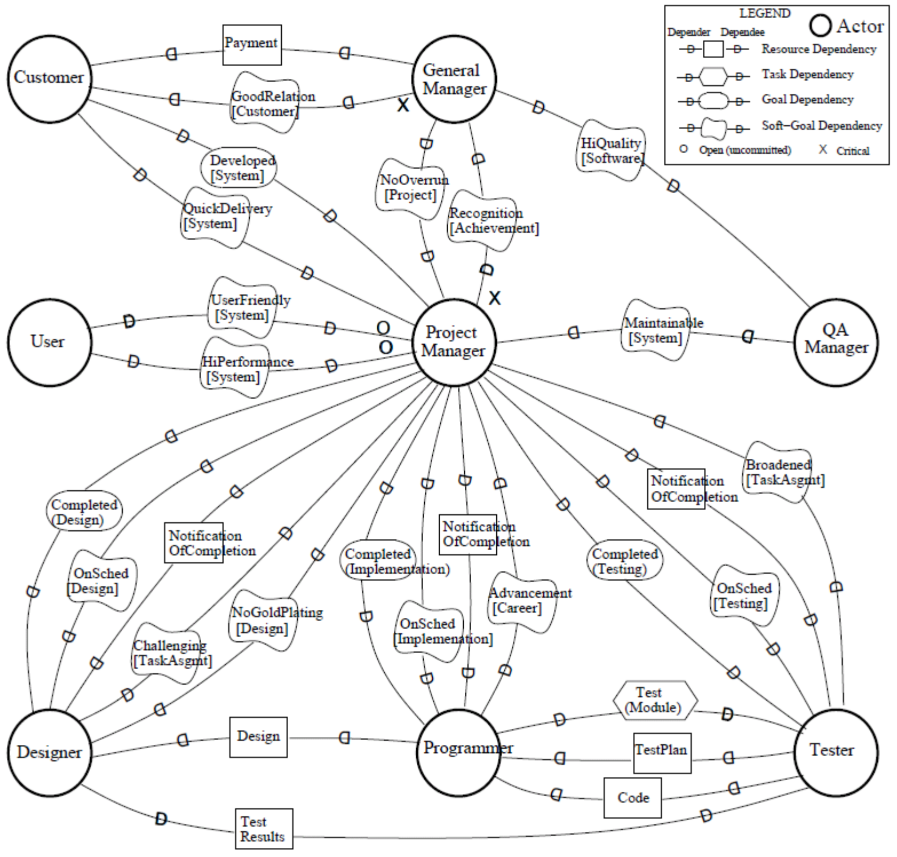


**Figure 3.3:** Yu's Actor-Dependency (AD) model example based on a software project scenario [YM94]

when a task requires another to execute one of its activities; moreover, (3) *Resource*, when a task depends on the availability of resources used by another task to execute its own.

The *strength* of the dependencies described in the AD model by Yu *et al.* [YM94] is a concept that is useful for our modelling to measure the degree of independence or *freedom* among the users' tasks. In our scope of user-centric CPHS, these levels of strength relate as follows: (1) *open*, the outcome of a task is desired by another task but not necessary; (2) *committed*, a task would be affected by any failure in the outcome of another task; and (3) *critical*, the task's final outcome is seriously affected by the execution of another task.

Mylopoulos et al. present an approach in goal models in which goals are used to analyse and define functional and non-functional requirements [MCY99]. In their approach, a *goal* is achieved when all its subgoals are satisfied. Mylopoulos et al.

include logical operators (AND, OR) to connect the operations in the subgoals, and conflict analysis to control potential collisions [MCY99]. Based on this approach, Figure 3.4 depicts a goal model for an online scenario [LLJM11]. The model shows the different alternatives that an online shopping task has while making a purchase.

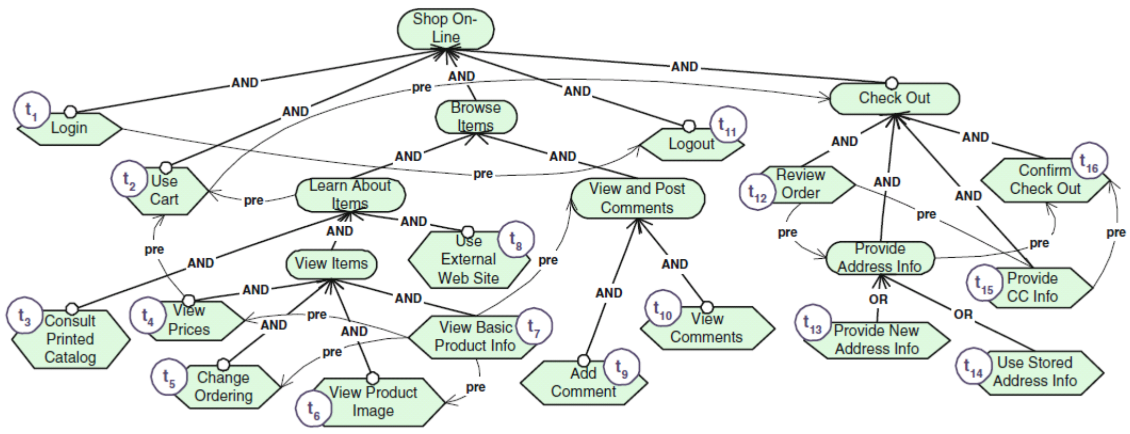


**Figure 3.4:** Liaskos' online shopping goal model example [LLJM11]. The ovals represent the goals, and the hexagons the tasks. Indices $t_i$ express the order of a task in the sequence.

Goal models have been used to add variability to systems by exploiting the OR part of the graph to create different sequences of execution. Yu *et al.* take into account the variability of the problem to integrate it as a variability on the solution [YLL+08]. Figure 3.5 depicts a set of patterns by decomposing the possible features of the goal model graph.

Some other authors have addressed goal-oriented modelling approaches in the scope of requirements engineering [CDGM10, LMSM10, LLJM11, BGM09, CLG+09]. For some of these authors, requirements are an expression of the necessity of the users according to the system's functionality and behaviour. In other words, the specification of the requirements is another way to express the intentions of the user.

## 3.2.2 Web Task

Some authors have model tasks from the point of view of the user activities, and just a few have modelled tasks from the web interactions. In our user-centric CPHS application domain, a *web task* is the integration of web-services along with the

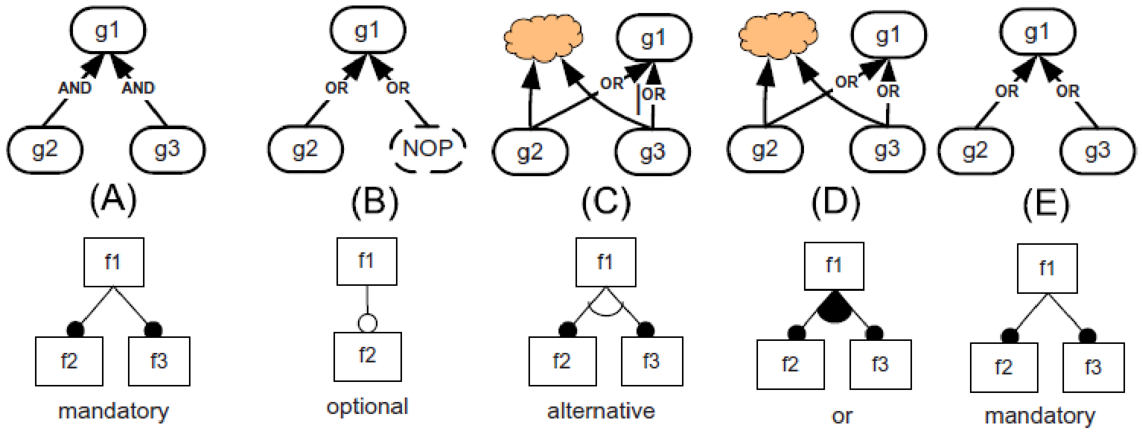


**Figure 3.5:** Yu's goal model set of patterns identified to incorporate variability in the form of features [YLL+08]

interactions and sessions that achieves the users' particular goals [CVM13]. In light of this, we studied approaches in two areas: *task models* and *models of web services.*

### Task Models

*Task models* represent the structure of the task, the activities that comprise the task, and other characteristics. Task models help to collect and analyse a user's task activity making possible to maintain a state of each instance of execution that are valuable in the modelling of personalized web-tasking.

In the approach by Giersich *et al.* [GFF+07], a task is the representation of the activities executed by users. Their task model is a hierarchical decomposition of an activity into individual steps of a task composing a task-tree. They use Dynamic Bayesian Networks (DBNs) to infer actions based on data from former users' activities. Goschnick *et al.* take advantage of agent-oriented architectures into video games given that these often need an autonomous game player (or character) to interact with human players to achieve a goal.

Dix's approach describes techniques to provide automated task support based on data and action, such as *waterfall*, which is observe, infer and then automate; and *threading*, which is sequencing a plan [Dix08]. The most relevant value of these three approaches is the representation of automation of a future task. For this dissertation, it is relevant for the prediction capabilities a smart internet application might have.

In Brezillon's approach, reasoning is divided into two elements: *diagnosis*, which is the analysis of the situation and its context; and *action*, which is the attempt to realize a task. A *task* is considered an activity on the structure of problem solving, and is part of a scheme to achieve a goal in a context-sensitive way. His approach uses graphs to sequence tasks executed by the user to achieve a goal, as well as the possible options and decisions the user has while executing the tasks [Bré07].

Klung and Kangasharju proposed a runtime task model that adapts in response to users' changing activities. They extended the *ConcurTaskTree (CTT)* notation to allow dynamic execution of a task model. In their approach they propose four states of the execution of a task: *inactive, active, enabled,* and *suspended.* Moreover, they present a definition of the semantics for information exchange and rules to verify the completeness and correctness of the model [KK05]. For this dissertation, it is relevant how the authors incorporate the concepts of port to communicate information among the tasks.

Souchon *et al.* provide a formal notation for task models, which includes the dependency of the context when a task is being modelled. In their approach, they focus on user interfaces to manage and execute a task. As in previous approaches, the task is represented as a graph (CCT) in which nodes are the tasks (and subtasks) and the edges are the relations among tasks as well as the sequence and dependency. Finally, the context is denoted as a tuple of *user, platform, and environment.* When a task can be associated with a different context, then the Contextual Task Model (CTM) is denoted as a tuple of *a set of task (subtasks), the root task, a set of transitions,* and a *matrix of all possible contexts* [SLV02]

As mentioned above, goals are identified as the intention of the user over the system. Goal-oriented modelling gives an approach to represent those goals as a set of tasks that are meant to be executed. Bolchini and Mylopoulos, suggest task analysis as the execution of both task identification, which is the classification of the task according to users' concerns, such as fact finding, browsing or exploration; and hierarchical task decomposition activities. As an insight for our research, their approach uses the AWARE goal-oriented notation—an extension of the i* modelling framework—in a website modelling scenario [BM03].

### Web Services

*Web services* allow applications from different platforms and languages to communicate through a standard interface. Despite the standardization of web services, their dynamic composition has been a concern addressed by different approaches. Cui *et* *al.* described key issues while implementing ontology modelling [CLWG04]. In their approach, a web service is composed by a set of input and output parameters, as well as relations between properties including semantic-equal and semi-semantic-equal. In a recent approach, Kim *et al.* implemented automatic web service composition using Open APIs [KKJ13]. They use ontologies such as RDF and OWL, which have structures for subject, property and object, to create automatic mash-up of services.

In 1999, Wilson proposed *the model of information behaviour* depicted in Figure 3.6 to represent the behaviour of the user while retrieving information from different sources [Wil99]. This approach highlights important considerations while modelling the relationship between the user, her personal goal, and the web interactions. This model outlines the various areas covered by the *information seeking behaviour*, which results as a consequence of the users' need to achieve an informational goal by retrieving information from different sources and services. As a result, the outcome can either be success or failure in terms of relevance for the user [Wil99].

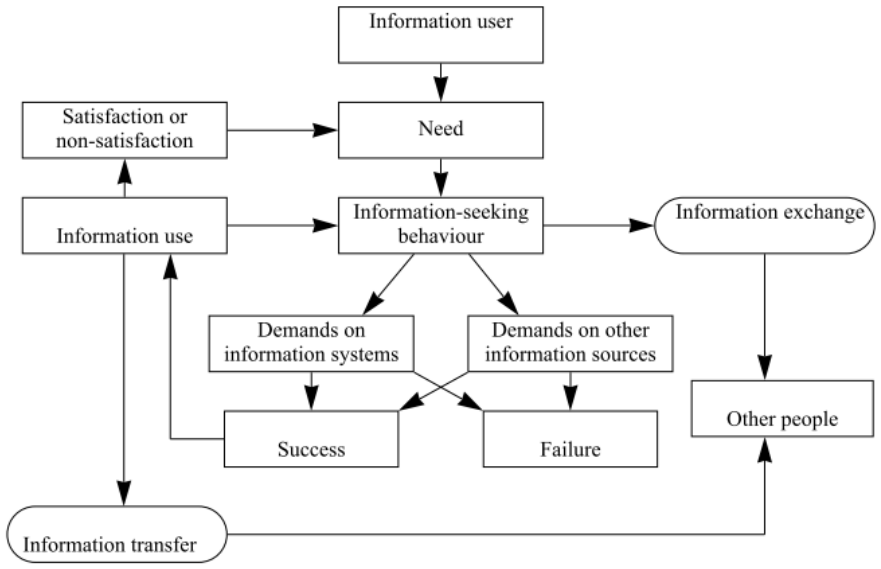

Information user
Need
Satisfaction or non-satisfaction
Information use
Information-seeking behaviour
Information exchange
Demands on information systems
Demands on other information sources
Success
Failure
Other people
Information transfer

**Figure 3.6:** Wilson's model of information behaviour [Wil99]

This representation provides some insights related with the smart interactions domain. For example, the user translates a need into a set of information retrieval activities. In smart interactions, *information retrieval* is a specific case of a personal goal. The *failure* element on the model is a dead end, in the smart internet application interest this may result in a decision to adapt to resolve errors and achieve success.

Traditionally, workflows and Business Process Modelling Notation (BPMN) have been user to represent processes and activities. These modelling approaches usually define a sequence of steps, actors, milestones, including assessment and work products presentation; as well as evaluation activities towards the achievement of repetitive and ordinary task.

Stoitsev and Scheidl characterize the types of user activities as: strategic, tactical, operational, implementation and educational [SS08]. *Operational* activities are those with high predictability given that they are repetitive and ordinary. They conclude that a task representation—considered a structure—must clearly define its subtasks, contextual information, resource, and all elements involved in the execution of the task.

The approach presented by Thom *et al.* take advantages of Workflow Activity Patterns (WAP) representation BPMN to specify the activities in recurrent business processes [TLI+11]. In their approach, they use workflow patterns to compose a process model. The patterns are related with the interactions among the elements. For example, approval patterns, define the type of roles that interact with this activity and the frequency.

John *et al.* implement a modelling method to combine task decomposition and a human resource usage [JVM+02]. The *Goals, Operators, Methods and Selection (GOMS)* method is used for prediction for user tasks that are repetitive. GOMS is used for modelling routine behaviour. In their approach, they focus on the decomposition of these routine tasks as nested goals. Finally, they consider the granularity of the iterative decomposition of the subgoals. In brief, their approach led to decompose a task into primitive interactions of the user (such as moving mouse and simple input of text) providing high degree of automation. Smart internet applications must consider the start and end points in the tasks' life cycle, as well as a level of granularity in their decomposition.

### 3.2.3 Context

In the domain of self-adaptive software systems, *context* is all information that is relevant to the system (i.e., the user, the system itself, and other systems that interact with it), and all the possible states while the system is being affected by the uncertainty of the environment [Vil13, ADB+99].

Previous approaches have been presented to represent and manage the user's personal context. Golemati *et al.* [GKV+07] and Katifori *et al.* [KVD+08] have proposed an ontology to profile the users in order to provide personalization on the application of information retrieval systems. They argue the importance of modelling the user's context and the need of a standard ontology that can be extended to different applications in various domains. Their ontology is used to represent static information. For example, (1) basic user information such as name and date of birth; (2) physical characteristics such as eye colour, weight, and height; (3) education background; (4) interests and activities; and (5) preferences. To represent dynamic information, this ontology includes temporary variables permitting various instances of the same class to exist and represent changes over time.

Villegas *et al.* studied similar approaches to characterize context modelling in the scope of context-aware software systems [VM10]. Later Villegas presented the SmarterContext ontology to represent the relevant context entities that support the smart interactions of the user. Her ontology is extensible to any application domain of situation-aware smart software systems. It was designed to represent dynamic and static context. In her approach, the *personal context sphere* is a model that contains the relevant information of the user according to a particular goal. Moreover, this ontology is the only one available to specify context models [Vil13].

The main module of the SmarterContext is called the *general context (GC)* and through extensibility (vertical or horizontal), it can also be specialized. This module defines context types, and abstract and concrete properties. Moreover, this approach uses the semantic web to manage the dynamic context. For this purpose, two technologies, the Resource Description Language (RDF) and the Ontology Web Language (OWL) are the mechanisms implemented.

Figure 3.7 depicts the context types in the GC module. The superclass is *ContextEntity*, is defined by the OWL Scheme element *Thing*, and all the other entities extend from it. As mentioned above, the GC represents abstract context information. For example, *ArtificialEntity*, which derives from *IndividualContext*, represents the entities that result from human actions, while *HumanEntity* represents the information about a person's characteristics, preferences, behaviour, and interactions.

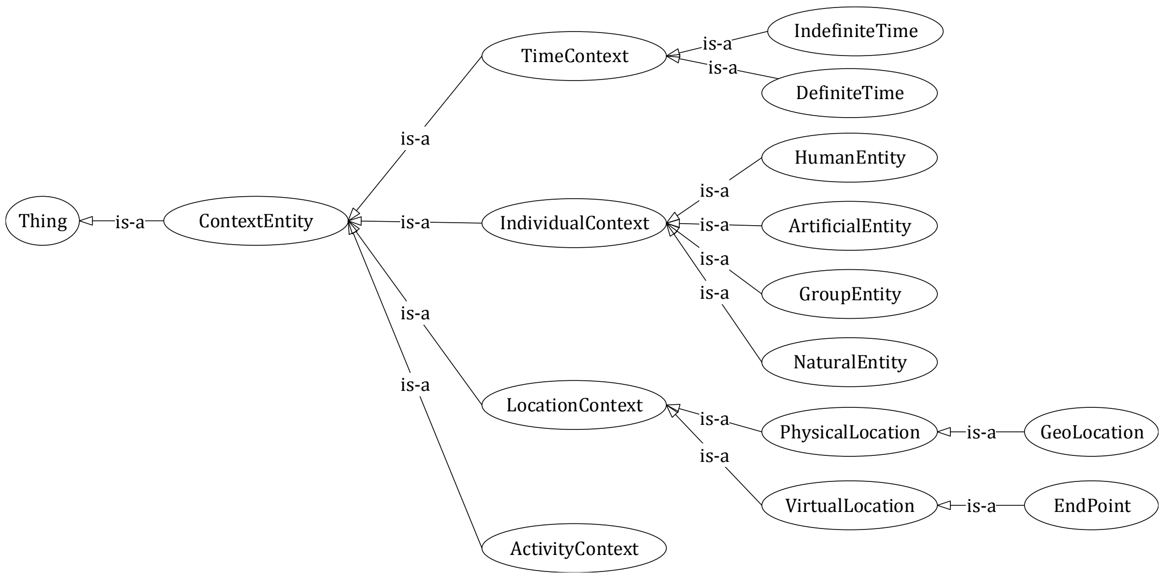


**Figure 3.7:** Villegas' General Context (GC) module context types of the SmarterContext approach [Vil13]

## 3.3 Chapter Summary

This chapter presented our study of modelling approaches for user-centric CPHS. Given the variety of available software models, we studied approaches associated with concerns relevant for user-centric CPHS, namely: (1) personal goals, which are related to the user-centric principle; (2) web tasking, which is relevant to the smart interactions definition; and (3) context, which is important to understand users' changing situations.

We conducted our study following a *systematic literature review (SLR)* proposed by Kitchenham *et al.* [KPBB$^+$09]. As a result, we surveyed 24 relevant approaches as summarized in Table 3.1. Then, we presented our findings grouped into three subsections based on our three concerns. Approaches, such as Yu [Yu93], Mylopoulos [MCY99] and Liaskos [LMSM10, LLJM11], present goal models in connection with users and tasks. More importantly, in this study, we identified the use of graphs as software structures useful for models to be manipulated at execution time, which is a characteristic of a MART.

# Chapter 4

# User-Centric Smart Cyber-Physical-Human Applications

This chapter presents our characterization and architectural design of User-Centric Smart Cyber-Physical-Human Applications (UCSAs), our first contribution of this dissertation.

In Chapter 2, we presented the *smart internet* and *Cyber-Physical-Human System*s *(*CPHSs*)*, which is a user-centric internet where the users are at the centre of all their internet interactions. In this vision, content and services are dynamically composed based on users' needs. A particular focus on the transformation of the internet towards a user-centric vision is in the execution of personal tasks that assist the user in achieving personal goals that can be personalized and automated based on the user's personal context. We discussed that there is a need for smarter internet applications that can exploit and understand the context of users and their changing situations. However, these internet applications co-exist in a socio-technical ecosystem of cyber, physical and human components. We posit that the smart internet is a CPHS. CPHSs add complexity to the realization of smart internet applications by adding diverse sources of context and highly dynamic environments.

There are two research questions associated with this contribution:

RQ1. *What are the [runtime] requirements for the realization of User-Centric Smart Cyber-Physical-Human Application*s*?*

RQ2. *What are the [runtime] modelling requirements to support User-Centric Smart Cyber-Physical-Human Applications?*

To answer these research questions, we concentrate on the smart internet's user-centric vision and its modelling requirements, specifically tasks that assist users in the achievement of personal goals.

We focus on two neglected components of CPHS: (1) humans as first class elements, in which their personal goals, interests, preferences, and interactions are at the core of the cyber and physical elements, thus defining their objectives, behaviours, structures and even their interconnections; and (2) the software component running as the orchestrator of smart interactions among CPHS components and users; as well as the responsibility of maintaining the system under changing conditions.

This chapter is organized as follows. Section 4.1 introduces *personal tasking* which is the conceptual foundation of this chapter and the starting point towards the realization of UCSAs. Section 4.2 presents our case study for online grocery shopping and explains how smarter applications are needed to assist users in the achievement of their personal goals. Section 4.3 presents our definition of a *User-Centric Smart Cyber-Physical-Human Application* along with its three characteristics: *user awareness, runtime modelling support, and runtime adaptation support*. Section 4.4 presents our *architectural design* for the realization of UCSAs as self-adaptive and context-driven systems. Finally, Section 4.5 summarizes the chapter.

## 4.1 Personal Tasking

Users increasingly rely on internet applications (mostly web based) to complete a variety of every day tasks that used to be performed offline. More importantly, users have become used to their devices, applications, and digital ecosystems, using different mechanisms to translate their own concerns and particular personal goals into a sequence of tasks. We define *personal tasking*[1] as follows:

> *Personal Tasking is an ordered sequence of tasks that assist users in the fulfilment of a personal goal. During personal tasking, users' context sources are exploited at runtime to understand changing situations and improve personalized functionalities [CVM13, CVM14b, CVM14c].*

[1]At the time of this research we used the term *Personalized Web-Tasking (PWT)* as we focused on the web as a subset of internet applications.

We now describe the process and conceptual elements of personal tasking as depicted in Figure 4.1. First, in the box with Label 1, the user expresses a personal goal to the system. Second, in the box with Label 2, the goal converts into one or more sequences of tasks (i.e., S1, S2, and S3). Each sequence of tasks corresponds to a set of subtasks (i.e., T$n$ where $n$ is the number of the task in the sequence), dependencies, and an execution plan. A subtask comprises a set of services, inputs and outputs, prerequisites, and restrictions of the subtask. Third, in the box with Label 3, context information from the user, environment, and system is exploited to select one of the candidate sequences, the one that best benefits users' interests and preferences. Finally, in the box with Label 4, the sequence is executed on behalf of the user while maintaining context-awareness and self-adaptive behaviour at runtime under changing conditions.

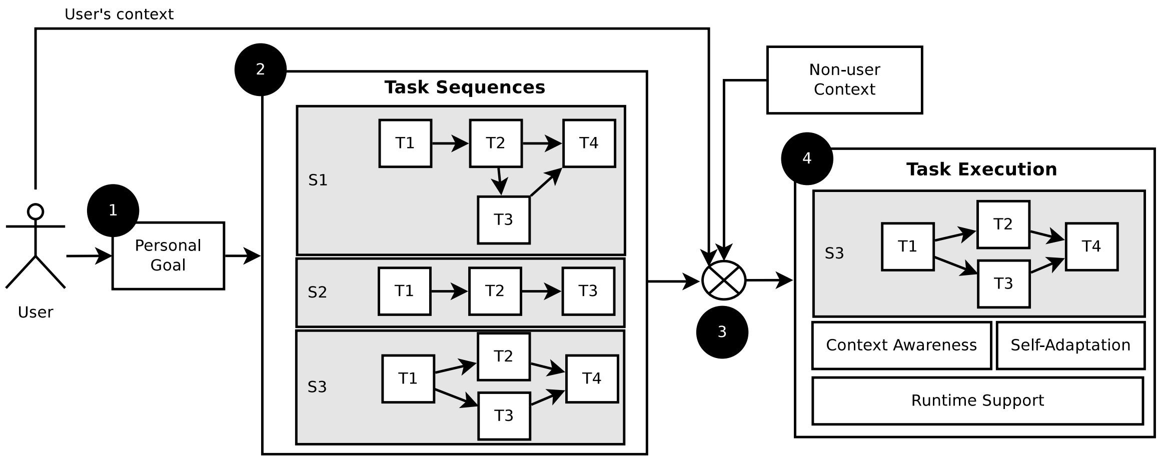


**Figure 4.1:** Conceptual elements of Personal Tasking

In personal tasking, personalization implies tailoring the tasks according to users' preferences, understanding users' situations, and being aware of changes in the context of the system and the environment. Exploiting personal context is not a trivial task for a system, particularly in the case of distributed and heterogeneous context sources. Although many applications support personalized and automatic features, the user remains as the manager, orchestrator, and runtime supporter of her task execution. Manual personal tasking is impractical. For instance, at some point of the execution the user can forget the task sequence, and then spend significant time trying to recover it. Moreover, the user might be unaware of newer and better services that may become available to improve the task sequence, and trying to find them is time-consuming and relies on the user's expertise. The manual management of the composition of a sequence becomes an issue, given that the user must keep track of the task life cycle manually.

Additionally, personal tasking might require the interaction of multiple services and applications and the user would have to control the sessions and permissions of each web service. Finally, the user will have to deal with runtime challenges, such as unexpected exceptions and unavailable services, whereas collaboration with other users will depend on the user's social networking and personal skills. In other words, there is no straightforward way to share her tasking execution with other users who may have the same concerns.

## 4.2 Online Grocery Shopping Case Study

Online grocery shopping is a suitable and interesting personal goal for our case study since it takes place in a socio-technological ecosystem with a variety of internet services, devices, sources of context information, and involves a diversity of users. Moreover, while grocery shopping is a mundane ordinary task it might be affected by context changes in the user's situations, such as, upcoming birthday celebrations, new social connection, dietary restrictions, or sudden health conditions. All of which might jeopardize the user's achievement of her personal goal.

We identify four distinctive [internet] interactions that are executed by users:

(1) The user *gets the shopping list*, which implies logging into her preferred grocery list service using a web browser or a mobile application.

(2) The user *locates the proper stores* using geo-localization services to find nearby grocery stores that she will filter semi-automatically according to her preferences that are already available in her social network.

(3) The user *creates independent shopping lists* by matching both items and stores according to different criteria. For instance, the category *best deals of the season*, which implies comparing prices semi-automatically to select the best offer according to her budget, or best reviews for both products and stores.

(4) The user *proceeds with the purchase* selecting one of two possible ways: if the store provides an online purchasing service, the user can proceed with the payment and schedule the delivery; otherwise she will have to semi-automatically plan the grocery shopping visit while taking into account different conditions

that can affect the pick-up process such as time, traffic, and the store's shopping hours.

Tasks associated with online grocery shopping are affected by the interoperation among different web services (i.e., changes in service compositions and incompatibility of data), and the changes in the user's context and situations (e.g., location, preferences, special events, or her behaviour when browsing the internet).

### CPH Components

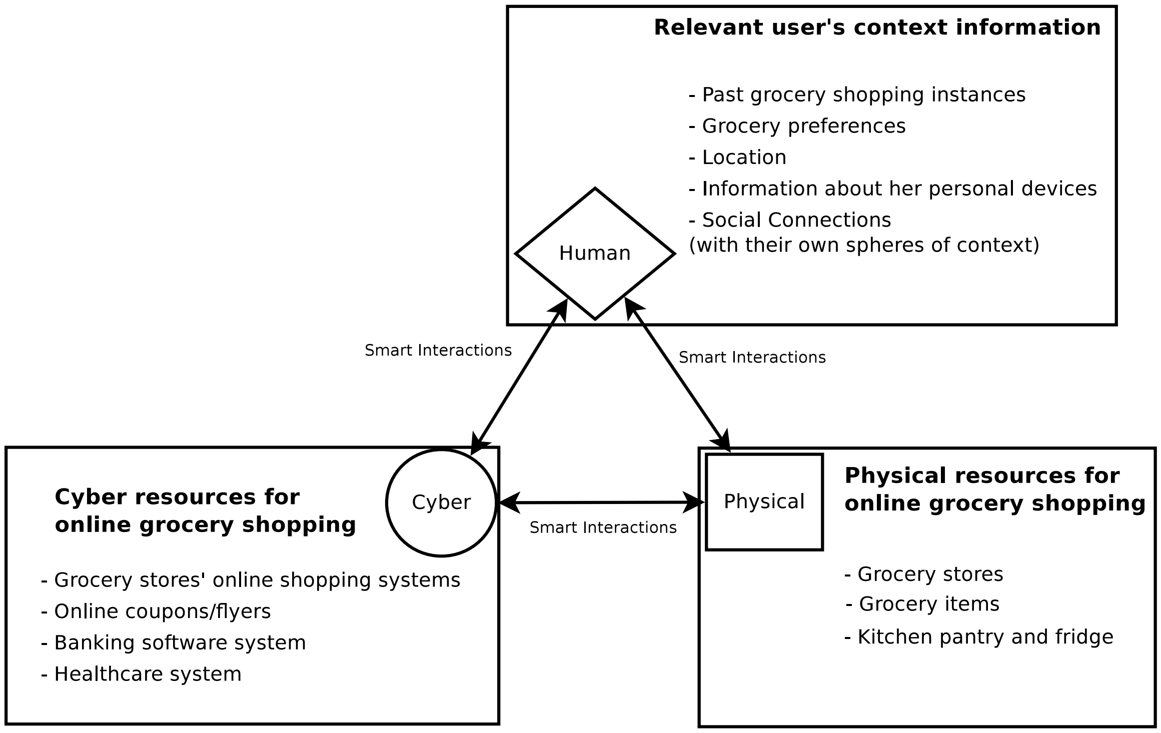

Relevant user's context information
- Past grocery shopping instances
- Grocery preferences
- Location
- Information about her personal devices
- Social Connections (with their own spheres of context)
Human
Smart Interactions
Smart Interactions
Cyber resources for online grocery shopping
Cyber
- Grocery stores' online shopping systems
- Online coupons/flyers
- Banking software system
- Healthcare system
Smart Interactions
Physical
Physical resources for online grocery shopping
- Grocery stores
- Grocery items
- Kitchen pantry and fridge

**Figure 4.2:** CPHS for online grocery shopping

Our online grocery shopping scenario takes place in a diverse socio-technological ecosystem as depicted in Figure 4.2. We distinguish three dimensions of CPHS: The *human dimension*, which comprises the user's personal goal of grocery shopping; and relevant user's context information, which includes knowledge of the user's previous grocery shopping tasks, her preferences (e.g., vegetarian, locally grown produce), her location since she prefers to shop in grocery stores close to her home, and social connections.

In the *physical dimension*, which comprises the resources required to achieve the user's personal goal, including nearby grocery stores, items, and related resources from the user's house.

In the *cyber dimension*, which comprises the computational resources involved in the achievement of the personal goals, including the grocery stores' own online shopping systems, internet flyers and coupons, mobile grocery list applications, and banking and healthcare systems, that although foreign to the personal goal, introduce context information that affects the achievement of the user's goal.

## 4.3 Definition of UCSA

We determined that there is a need for software applications capable of supporting users' personal tasking activities, along with situation awareness and runtime adaptation. As a result, we define User-Centric Smart Cyber-Physical-Human Applications (UCSAs) as follows:

> **Definition 4.1** A *User-Centric Smart Cyber-Physical-Human Application (*UCSA*)* is an orchestrated set of cyber, physical, and human components (along with their interconnections) that assist users in the fulfilment of their personal goals. A UCSA manages the smart interaction among the components dynamically, understands and acts upon users' changing situations, and has system capabilities to evolve at runtime.

It is important to clarify that not all smart CPHS applications are focused on assisting users' achievement of personal goals. For instance, applications whose objectives correspond to industrial processes, business workflows, or file management, do not fall into our category of interest. We only consider applications where users' context is relevant for the fulfilment of the system's objectives. For instance, applications where users are system administrators, sole providers/consumers of non-user-related information, or silent observers are not consider relevant here. To distinguish UCSA from other CPHS applications, we add three new characteristics: *user awareness, runtime modelling support,* and *runtime adaptation support.*

### User Awareness

Technological evolution has transcended towards integration. From the early design of the internet (back when it was ARPANET), we have been connecting things. CPHS,[2] represents an internet growing digital ecosystem of devices, applications and users, connected through the internet [CE11, PZCG13, KFM+13, CLPS11] that are constantly generating context information about the user.

In a seamless way, humans have been connecting themselves to thing, and thus becoming part of the CPHS through their virtual persona. It is fair to say that users can be considered either physical or cyber components. There is more about users than what their applications and devices can contribute to their applications. Users also have personal goals, internet interactions, social relations and activities, interests; and more importantly, transverse personal context across unrelated systems where users are present. Often, personal context is strongly tied to a service or application, but in UCSA, users are enabled to move around systems using their own information to improve their application experiences. The complexity and uncertainty added by the users' own dynamic situations, play a key role when addressing runtime requirements in this dissertation.

We define that user awareness in UCSA needs an explicit identification of the *human dimension*. In other words, user awareness provides a sphere of information (static and dynamic) containing users' concerns, personal data, and relations with other users, that are relevant for—and during—the execution of the CPHS. More importantly, a UCSA should recognize the user at the centre of its interactions, that is, the personal goal of the user should be the principal objective of the system.

### Runtime Modelling Support

In order to support runtime operations, a UCSA should connect (or contain) Models at Runtime (MARTs) that represent up to date information of the system and the environment. Since UCSA require runtime execution, it is necessary to include an infrastructures responsible to monitor, analyse, plan and adapt runtime models. Whether the infrastructure is embedded with the system or not, its functionality

[2]This dissertation uses the term *cyber-physical-human systems*. However, conceptual foundations come from its origins in the literature in relation with other definitions, such as *industrial internet of things, internet 4.0, web of things, internet of things, future internet* and *industrial internet.*

should not interfere with the system's primary objectives of assisting users to achieve personal goals.

To provide the flexibility required by dynamic systems, UCSA's runtime models and infrastructures must support emerging models, operations, and connections.

### Runtime Adaptation Support

A UCSA has an operational runtime constraint, that is, the system can not afford to go offline in order to perform additional activities out of its intended functionalities, such as runtime modelling tasks, and adaptations based on changing situations. Therefore, these systems should have an infrastructure to manage the system's adaptations, capable of storing past, current, and planned versions of the system, as well as planning and executing the version change, while keeping the system working properly.

Since runtime adaptation is not a trivial problem, this dissertation focuses on the capability of the UCSA to propagate changes across the runtime models as a step towards achieving runtime adaptation for CPHS [MV13].

## 4.4 Architectural Design of UCSA

In the vision of creating UCSA, the user's identity, interactions, personal goals, preferences, and context, determine the decisions and the behaviour of the users' interactions. For this purpose, UCSA must understand the user's goal and represent it as a sequence of tasks that can be improved with personalized features and executed automatically on behalf of the user. Context- and situation-awareness are important for UCSA to understand the various situations of the users and the relationships with their tasks when fulfilling personal goals. More importantly, the changing nature of users' situations, personal goals and the context, implies that UCSA systems behave dynamically. That is, at runtime the system is capable of understanding changes and act upon them by adapting itself to meet new requirements.

The realization of UCSA requires the implementation of systems that act upon unexpected context changes by adapting itself at runtime, when applicable. This is the case of self-adaptive systems which are capable of monitoring changes in the context and perform required adaptations themselves—structural and behavioural—

at execution time with the purpose of maintaining or even improving the objectives of the system [CLG+09, MKS09, LGM+13].

Figure 4.3 depicts our architectural design for the implementation of UCSA [CVM14c]. Our architecture is based on the DYNAMICO reference model proposed by Villegas et al. [VTM+13]. DYNAMICO is readily applicable to the implementation of UCSA by allowing the definition of architectural components that can be extended with concepts of UCSA. Moreover, DYNAMICO supports the implementation of self-adaptive systems that are highly affected by changes in goals and context situations at runtime.

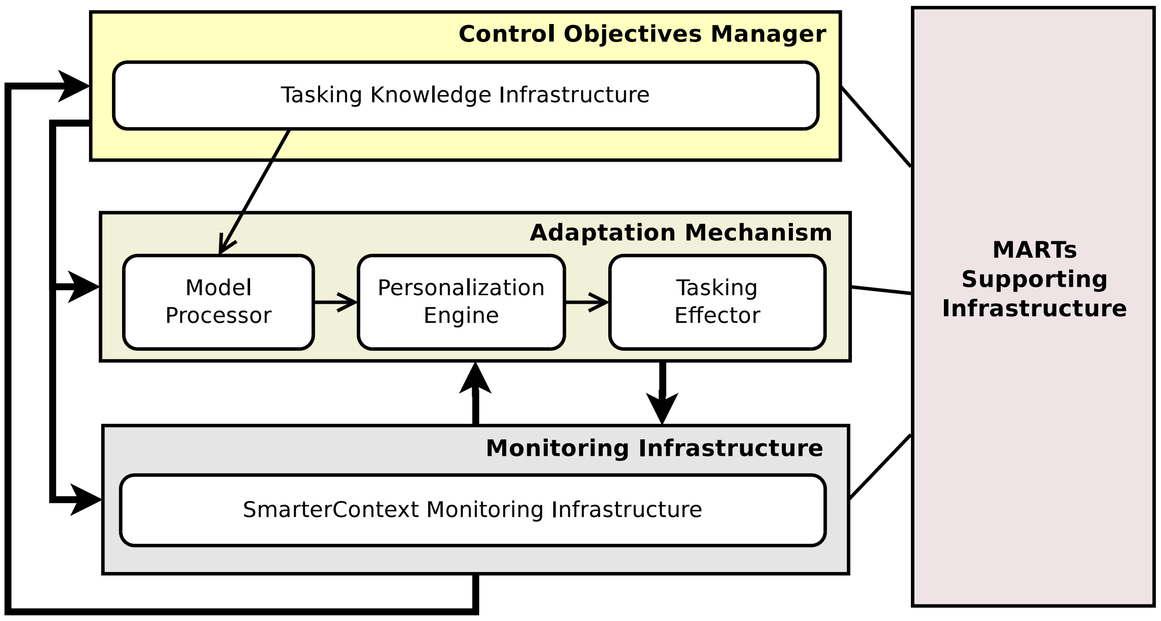

Control Objectives Manager
Tasking Knowledge Infrastructure
Adaptation Mechanism
Model Processor
Personalization Engine
Tasking Effector
Monitoring Infrastructure
SmarterContext Monitoring Infrastructure
MARTs Supporting Infrastructure

**Figure 4.3:** User-Centric Smart Application architectural design

Our UCSA architecture implements three subsystems derived from DYNAMICO, and one for the runtime models support. The subsystems derived from DYNAMICO are: (1) the *control objectives subsystem* maintains the relevance of the system with respect to changes in the user's personal goals; (2) the *adaptation mechanism* allows the system to act upon changes that occur either in the context or in the requirements of the system; and (3) the *monitoring infrastructure* keeps track of context events that are relevant to the execution of task sequences. The *MARTs supporting infrastructure* is responsible for the access and activities related to the models at runtime which are the main focus of this dissertation.

At the *control objectives subsystem* comprises the *Tasking Knowledge Infrastructure*. This component provides the instrumentation to express personal user goals in the form of a MART. The components in this module perform two main activities: (1) recording information from users' internet interactions; and (2) representing this information as an instance of a model at runtime. To record a user's internet interactions the infrastructure needs to identify, interpret and characterize internet actions (e.g., click, selection, or inputs) and data, which implies instrumenting the browser, devices and web sites to extract this information.

The *adaptation mechanism* subsystem contains three components: (1) the model processor, (2) the personalization engine, and (3) the tasking effector.

1. The *model processor* translates the information gathered from the knowledge infrastructure into a corresponding runtime model. This component might determine that for one personal goal there are several instances of the model that are candidates based on the user's historical behaviour, social connections, and environmental information. The *model processor* collects all possible instances and delivers them to the next component for selection.

2. The *personalization engine* exploits users personal context to tailor the corresponding model at runtime, selecting the best match based on users' preferences and interests. Personal context information can be either static (e.g., age or gender) or dynamic (e.g., location, preferences, or social information). As a result, this component produces one instance of the model that will best fulfil the users' personal goals.

3. The *tasking effector* executes the tasking described in the corresponding model at runtime. This component manages the life cycle execution including internet services invocation, requests for user intervention, conflict resolution, and final assessment of the tasking.

The *monitoring infrastructure* subsystem in our architecture is provided by the SmarterContext monitoring infrastructure [Vil13] and its ontologies, which we extended to fit our application domain.

Finally, the *MARTs supporting infrastructure* comprises the MARTs of the system, and software components responsible for granting read and write capabilities to the system over their MARTs.

## 4.5 Chapter Summary

*UCSA* constitutes the first contribution of this dissertation. It allows us to answer research questions RQ1 and RQ2 (cf. Figure 1.2).

In Section 4.1, we introduced *personal tasking* as the sequence of tasks that assist the user in the achievement of a personal goal. In Section 4.2, we introduced our case study of grocery shopping—a suitable example for the realization of UCSA, since it involves a variety of cyber, physical and human elements that add complexity to the execution of the users' tasking.

In Section 4.3, we defined a *User-Centric Smart Cyber-Physical-Human Application (*UCSA*)* as an orchestrated set of cyber, physical, and human components (along with their interconnections) that assist users in the fulfilment of their personal goals (cf. Definition 4.1). A UCSA manages the smart interactions among the components dynamically, understands and acts upon users' changing situations, and has capabilities to evolve at runtime.

We presented three characteristics for UCSA in addition to the ones of CPHS applications: *user awareness* to place the user at the centre of the smart interactions of the system, *runtime modelling support* to provide the system with up-to-date representation of the system and the environment, and *runtime adaptation support* to enable the system to be adapted and extended during execution time.

Finally, we propose an architectural design for UCSA as a self-adaptive context-driven software system based on the DYNAMICO reference model. In our architecture, we defined five components for UCSA: the *tasking knowledge infrastructure* responsible to understand and represent personal goals based on the analysis of users' task interactions. The *model processor* translate the information about users' tasking into runtime models. The *personalization engine* exploits static and dynamic personal context to tailor an appropriate task sequence. The *tasking effector* executes the tasks on behalf of the user while providing runtime adaptation support. Finally, the *MARTs supporting infrastructure* responsible for managing and hosting the corresponding MARTs.

The next chapter presents our MARTs for UCSAs, our approach to represent the personal tasking of the user as models at runtime that will be stored and managed in the *MARTs supporting infrastructure* of the UCSA.

# Chapter 5

# Models at Runtime for User-Centric Smart Cyber-Physical-Human Applications

This chapter presents our two Models at Runtime (MARTs) for User-Centric Smart Cyber-Physical-Human Applications (UCSAs): our Galapagos Metamodel, and our Galapagos Model, which is our second contribution of this dissertation.

In Chapter 4, we presented our definition and architectural design for *UCSAs*. According to Definition 4.1, a UCSA is set of cyber, physical and human components that assist users in the fulfilment of personal goals. More importantly, a UCSA manages the smart interaction among the components dynamically, understands and acts upon users' changing situations, and has capabilities to evolve at runtime. In our architectural design (cf. Section 4.4), we presented the *MARTs supporting infrastructure* component which is responsible for managing and hosting the corresponding MARTs of UCSAs. MARTs are fundamental for UCSAs to represent and understand users' personal goals and changing situations at runtime.

There are two research questions associated with this contribution:

RQ2. *What are the [runtime] modelling requirements to support User-Centric Smart Cyber-Physical-Human Applications?*

RQ3. *What are the appropriate runtime models for the implementation of User-Centric Smart Cyber-Physical-Human Applications?*

To answer these research questions, we define the modelling requirements based on the definition of UCSAs grouped into three concerns: *personal goals, tasks*, and *context*. Moreover, we propose two MARTs: (1) our Galapagos Metamodel that represents the concepts and artefacts of UCSAs, and (2) our Galapagos Model that represents the evolving tasking goals of the user, relevant context, and smart interactions of CPHSs.

This chapter is organized as follows. Section 5.1 presents our modelling requirements organised into three subsections, one for each concern: *Personal goals, Tasks*, and *Context*. Section 5.2 presents our two MARTs—our Galapagos Metamodel and our Galapagos Model. Finally, Section 5.3 summarizes the chapter.

## 5.1 Modelling Requirements

To derive proper models of User-Centric Smart Cyber-Physical-Human Applications (UCSAs), it is important to define what are the requirements for such models. That is, performing an analysis of the realities that need representation. According to the conceptual reference model of M@RT proposed by Bennaceur [BFT+14] (cf. Section 2.4.2), there are two realities: (1) the running system; and (2) the environment. Figure 5.1 depicts our instance of the Level M0 of the conceptual reference model of M@RT for our UCSAs' domain.

The *running system* is our UCSA, which assist users in the achievement of personal goals. The *environment* comprehends relevant context information associated with the three dimensions of CPHSs—physical, cyber or human information. Since the environment is observed by the running system, its modelling requirements must derive from the UCSA's point of view.

We characterize our modelling requirements and grouped them into the following three concerns:

- *Personal goals*, that is, the resulting expectation of the user while performing tasks;
- *Tasking*, including the sequencing and execution of the tasks and subtasks, dependencies, inputs and activities; and
- *Context*, which is information relevant to the achievement of a personal goal that is classified as personal, internal or external.

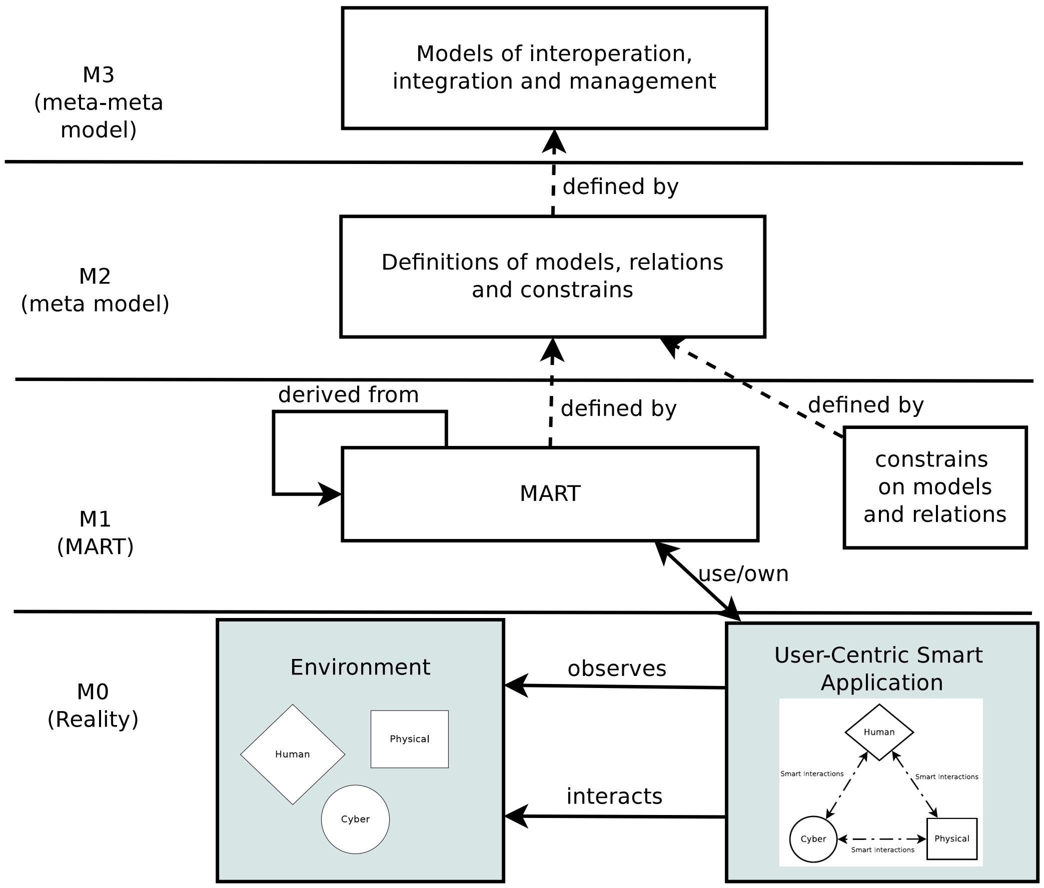


**Figure 5.1:** Level M0 for User-Centric Smart Cyber-Physical-Human Applications based on Bennaceur et al.'s conceptual reference model for M@RT [BFT+14]

.

In this chapter, we use our scenario for online grocery shopping described in Section 4.2 to illustrate our modelling requirements. As we mentioned before, online grocery shopping takes place in a socio-technological ecosystem with a variety of internet services, devices, sources of context information, and involves a diversity of users.

### 5.1.1 Personal Goals

Figure 5.2 depicts a simple view of the personal goal concern of UCSAs. The starting point is a user with a *personal goal* that she wants to fulfil with the assistance of internet technologies. Her personal goal then is transformed into an ordered sequence of *task interactions* involving a variety of internet services, data types, and inputs.

The output of the interactions is a *measurable outcome*, which must reflect the user's expectations.

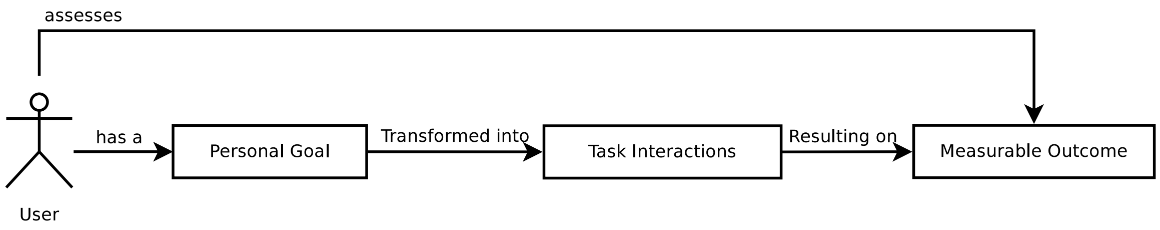


**Figure 5.2:** Concept of a user's personal goal of UCSAs

It is worth noting that measurable outcomes must connect with the user's personal goal. For instance, if the user is looking to shop for groceries, her measurable outcomes might include her bank statement showing her grocery purchase, and her shopping list containing fewer items (cf. Figure 5.3).

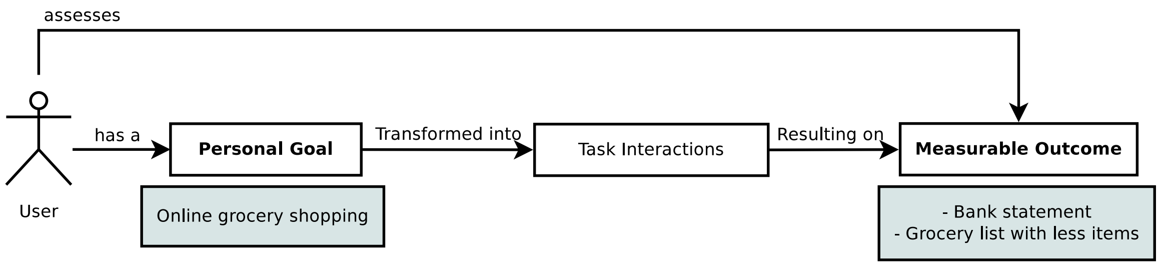


**Figure 5.3:** Personal goal and measurable outcome according to our online grocery shopping scenario

Instead of focusing on the modelling requirements of expressing personal goals, we take a different approach focusing on the modelling requirements of assessing the achievement of personal goals. We take this approach because we want to abide by the user-centric principle of instinctive interactions from the smart internet vision. In most everyday internet activity, users dynamically access different services and applications to achieve a personal goal, rather than introducing their objective into one single software system. Therefore, we pay special attention to personal goals as the results of a set of activities.

Accordingly, the modelling requirements for personal goals are:

- *Observable result*: The outcome has an identity, attributes, and possibly services and states.

  In Figure 5.3, there are two results of interest: an application with fewer items, and grocery purchases charged to the user's credit card. Other observable results might as well be a notification, or calling an event on the user's devices or applications.

- *Satisfaction measurements and thresholds*: The observable result is evaluated based on an acceptance analysis. Criteria have to be related to measurable properties of the result.

  For instance, the user might consider that the grocery shopping list (observable result) is accepted when empty (measurable property), or if 80% of the items were purchased, and thus removed from the list (threshold of acceptance). Moreover, another indicator of satisfaction might be that the total amount of money spent on groceries is within her $200 dollars budget. The amount paid gets measured by the charges on the user's credit card.

- *Satisfaction regulation:* Rules for satisfaction measurements.

  For instance, a rule might establish that being within budget is more important than purchasing 80% of the items in the list. Therefore, even if not all items in the list are purchased but the charges are under $200 dollars, the goal achievement is considered a success.

- *Execution regulation:* Rules for executing or discarding the tasks.

  For instance, buying groceries every 15th of the month, or any time during a period of a month when a quarter of the items are on sale.

### 5.1.2 Tasks

A task is a composition of services, interactions and sessions. Moreover, a task can be an ordered sequence of tasks that are executed independently [CVM13].

Task interactions define a sequence of internet tasks (cf. Figure 5.4), which include a set of tasks, an execution plan, and a set of dependencies. Some examples of dependencies are resources, data, authorizations, and additional information needed during the sequence execution. Each task might contain a set of inputs and outputs, services required, operations, and their place in the execution order.

In light of this, the modelling requirements for tasking are as follows:

- *Task sequence:* A structural representation of the linked tasks within the system.

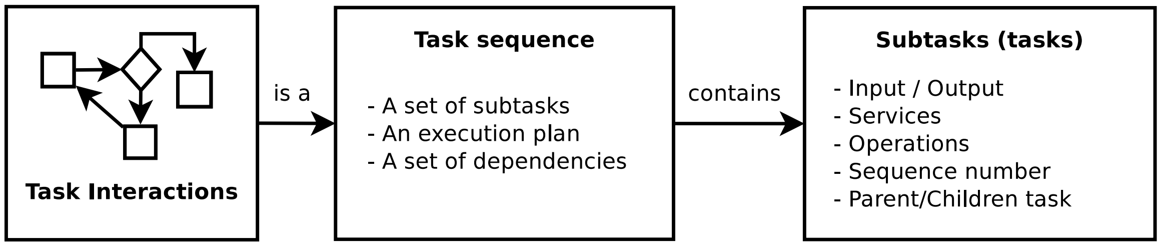


**Figure 5.4:** Task interactions are a set of tasks and subtasks

Links among tasks represent how two tasks communicate with each other and the data flow in their communication. Links provide information to about the compatibility of two tasks.

- *Inputs:* Every task within the sequence must define the required input and its data types.

  An input includes data, type, and source. Input sources could be the user, other tasks, external services, local applications. Moreover, inputs might be shared among internet tasks of the sequence.

- *Outputs:* The representation of the result of the task.

- *Activities:* This represents the steps of the task during its execution.

  Steps are specified in the form of an algorithm. This requirement include conflict resolution and handling exceptions techniques.

- *Conditions:* The specification of particular considerations of the execution of the task.

  Conditions include time, frequency, and other configuration information relevant to the execution.

- *Information resources:* This element represents either the software that is responsible for the logical processing of the information service request, or data repositories.

  Examples of information resources include files, databases, knowledge base, local hardware sensors data, web services, functions, or other applications.

- *Policies:* A representation of the operational objectives of the system that outlines the decisions of the planning and execution.

- *Execution controller:* The life cycle of the task execution.

  Explicitly defines the starting an ending points of the execution, and termination decisions (e.g., time-out).

For instance, Figure 5.5 depicts the *online grocery shopping* main goal, which is achieved by a main task of *shopping grocery items*. This task is decomposed into other tasks, such as *(1) getting the grocery shopping list*; *(2) searching and selecting proper stores*; and *(3) placing and purchasing items*. However, decomposition might continue for several levels. For example, subtask (2), which is a task as well, is decomposed with a sequence of its own label as 2.1, 2.2, and 2.3). Every decomposition iteration also specifies inputs, outputs and other information required in the sequencing of the task. For example, subtask 2.2 determines that the input is the *grocery items*, which comes from the previous task, and its output is a *list of ranked proper stores*. Then it describes a set of operations some of which we can identify as tasks, therefore a new iteration of decomposition.

### 5.1.3 Context

We need to represent context for two main reasons: First, UCSAs need to understand context to support its situation-awareness functionalities. Context awareness allow UCSAs to understand users, personal goals, task interactions and changing situations. Second, the selection and execution of an appropriate task sequence to achieve a personal goal depends on the system's capability of understanding relevant context at execution time.

We group context modelling requirements into three concerns: *personal, internal*, and *external*. Figure 5.6 depicts an overview of our three groups of interest. Personal context includes users and their personal goals. Internal context includes the information related to the system, such as the task interactions and the measurable outcomes. External context is the environment interacting with the tasking execution and the user during the achievement of a personal goal.

- *Personal Context:* relevant information from the user that affects (or is affected by) the behaviour of the system. Personal context includes static or dynamic user-related information, goal-oriented interactions with personal applications, and social relations with other users in relation with the achievement of her personal goal.

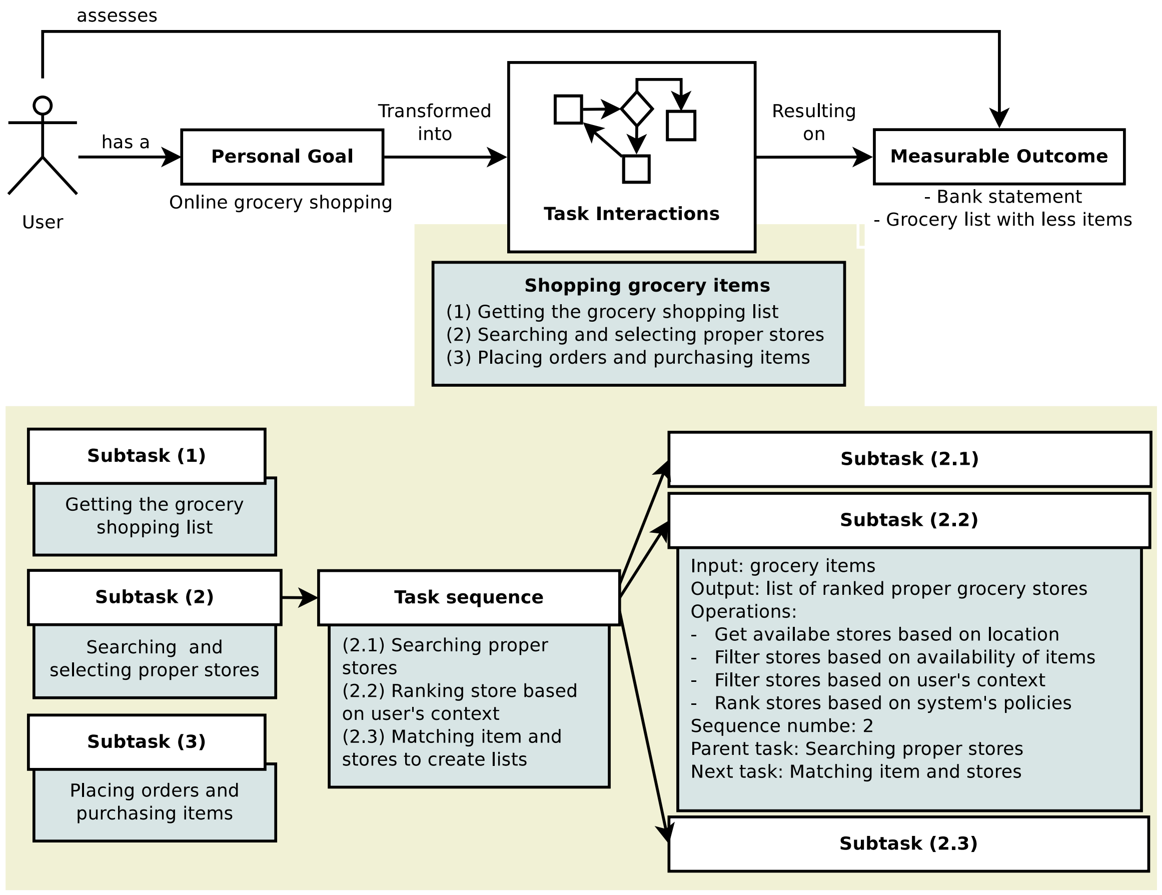

assesses
User
has a
Personal Goal
Online grocery shopping
Transformed into
Task Interactions
Resulting on
Measurable Outcome
- Bank statement
- Grocery list with less items
Shopping grocery items
(1) Getting the grocery shopping list
(2) Searching and selecting proper stores
(3) Placing orders and purchasing items
Subtask (1)
Getting the grocery shopping list
Subtask (2)
Searching and selecting proper stores
Subtask (3)
Placing orders and purchasing items
Task sequence
(2.1) Searching proper stores
(2.2) Ranking store based on user's context
(2.3) Matching item and stores to create lists
Subtask (2.1)
Subtask (2.2)
Input: grocery items
Output: list of ranked proper grocery stores
Operations:
- Get availabe stores based on location
- Filter stores based on availability of items
- Filter stores based on user's context
- Rank stores based on system's policies
Sequence numbe: 2
Parent task: Searching proper stores
Next task: Matching item and stores
Subtask (2.3)

**Figure 5.5:** Tasks for online grocery shopping

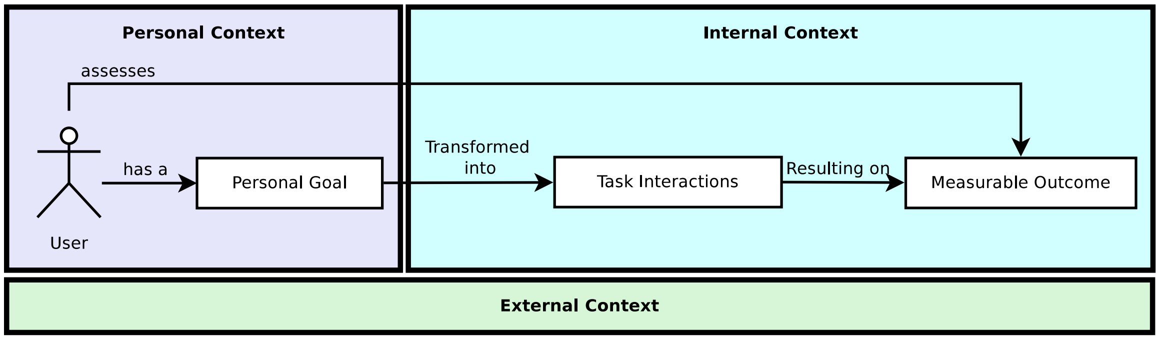


**Figure 5.6:** Context groups of interest

- *Internal context:* refers to the elements within the system's domain.

  For instance, tasks sequences, execution controller, and knowledge. Knowledge might be previous instances of tasking including their adaptations and conflict resolution. Thus, internal context is also the past states of the system.

- *External context:* relates to internet services, applications, interactions or data storage outside the domain of the system.

  The representation of outer internet services and other applications that affect the achievement of the personal goal. This modelling requirement includes the state, service, and communication protocols of the external context with the system.

Figure 5.7 depicts examples of context for our online grocery shopping scenario. Personal context includes user's preferences, location, applications and other settings in relation with her personal goal. Internal context includes connections between the grocery shopping items and the selected stores to purchase those items. The logical relationship item-store that only exists in the domain of the system, which is used to proceed with the purchase. External context includes the online systems of the grocery stores that provide online shopping access, and online banking services to execute payment capabilities in the grocery stores (e.g., PayPal,[1] Dwolla,[2] Authorize.Net[3]).

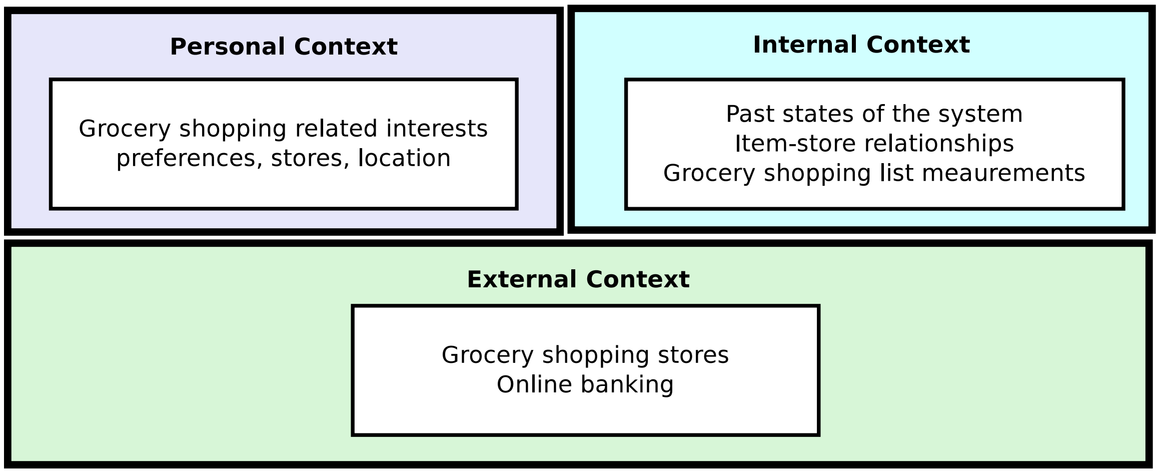

Personal Context
Grocery shopping related interests
preferences, stores, location
Internal Context
Past states of the system
Item-store relationships
Grocery shopping list meaurements
External Context
Grocery shopping stores
Online banking

**Figure 5.7:** Context examples for online grocery shopping

We presented the modelling requirements for the realization of UCSAs based on the analysis of the realities that need to be modelled: the running system and the

[1]http://www.paypal.com
[2]http://www.dwolla.com
[3]http://www.authorize.net

environment. We grouped our requirements into three concerns: personal goals, tasks, and context. Before we defined our MARTs it is necessary to explore related approaches that will meet our modelling requirements. Some elements from our modelling requirements have been addressed separately by different authors in diverse application domains. In Chapter 3, we presented our study of those that are highly relevant for UCSAs concepts, analysed their compatibility, and extracted relevant elements.

From our study, we identified that from the software models' point of view, *task models* and *goal models* are strongly connected, given that tasks are the activities that the user performs to fulfil a goal. Goal models are often used to represent users' intentions as system requirements.

From an implementation point of view, *graphs* are model structures that permit the creation of sequences, easily manageable and modular, which provides flexibility to models. Graphs are employed to represent the task decisions and define the relationships among the tasks, including dependencies and the information that is being exchanged. Moreover, graphs are useful in representing multiple decisions or variability on choices [BM03, LMSM10, LLJM11, LMSM11, MCY99, Yu93, YM94]. Most approaches take into account the existence of the environmental context as well as the user's. In terms of modelling representation, ontologies seem to be the proper way to represent context [Dix08, SLV02, Vil13, YM94].

In summary, some of our modelling requirements have been addressed from various approaches in other domains and applications. However, to the best of our knowledge of the state of CPHSs research, there are no approaches that properly fit our requirements. The next section presents our MARTs for UCSAs.

## 5.2 Our Models at Runtime

Figure 5.8 depicts our layers M1 and M2 instance of the Bennaceur's reference model. We propose two MARTs for UCSAs: (1) Our Galapagos Metamodel that represents the concepts and artefacts of UCSAs, and (2) our Galapagos Model that represents the evolving tasking goals, relevant context, and smart interactions of UCSAs [CVM14c].

It is worth mentioning that since a property of MARTs is *availability* (cf. Section 2.4) a runtime model requires to be implemented in a format that can be read and accessed by software applications. Our MARTs are available in the form of

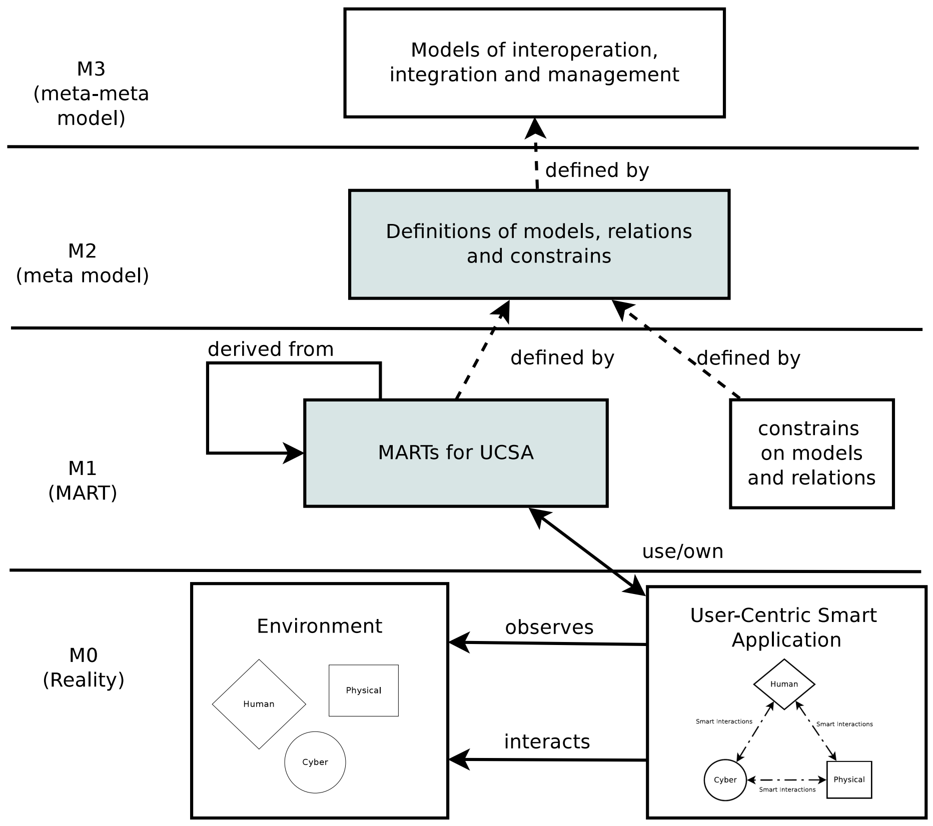


**Figure 5.8:** Level M1 and M2 for User-Centric Smart Cyber-Physical-Human Applications based on Bennaceur et al.'s conceptual reference model for M@RT [BFT+14]
.

OWL2/RDF files.[4][5] Moreover, our metamodel is applicable in different domains, such as tasking supported by stand-alone software systems, thus supporting the growth of the internet and satisfying the *evolution* property of MARTs.

## 5.2.1 Galapagos Metamodel

*The name Galapagos for our MARTs, is inspired in the well-known archipelago of Galápagos in the Pacific Ocean[6]. Our name takes from the idea of Charles Darwin's inspiration for his theory of evolution after observing the living species of the archipelago. Our models, describes the evolving tasking of the user while achieving a personal goal, based on the changes in situations of users and the context.*

[4]http://www.rigiresearch.com/research/pwt/pwtOntology.owl
[5]http://www.rigiresearch.com/research/pwt/gct-onlinegrocery.owl
[6]https://en.wikipedia.org/wiki/Galapagos_Islands

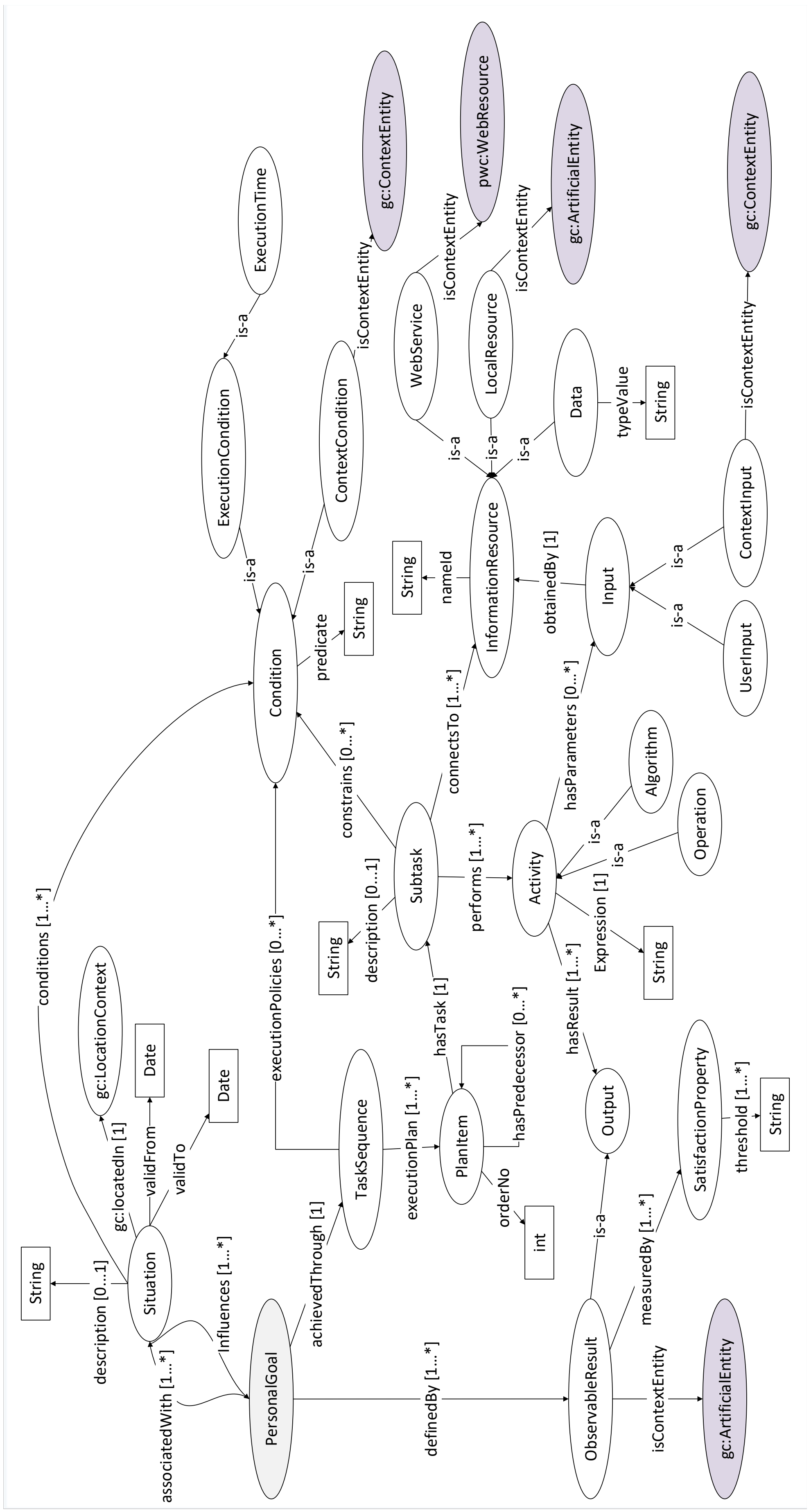


**Figure 5.9:** Galapagos Metamodel

Our first MART, depicted in Figure 5.9, defines the concepts of UCSAs by abstracting the three dimensions of CPHSs as well as the smart interactions among them. It comprises two elements: *entities* and *links*. *Entities* represent concepts, which are complex objects of the problem domain. This is, cyber, physical, human, or smart interaction. *Links* represent the connections between two entities composed by a name, direction and cardinality. *Name* is a descriptor used to describe the type of relation among the entities. *Direction* defines the dependency of one entity over the existence of another. *Cardinality* determines the quantity of instances that are expected at the end of the link.

By definition, MARTs are representations of realities that are owned and used by the running system. The Galapagos Metamodel's reality is the application domain of UCSAs. In order to explain our model, we highlight two key sections from our metamodel: (1) the core concepts and (2) the situation-awareness concepts.

## Core Concepts

Figure 5.10 depicts a simplified version of our model to show selected entities and links that are part of the core concepts of UCSAs. As mentioned above, entities represent complex objects of the problem domain.

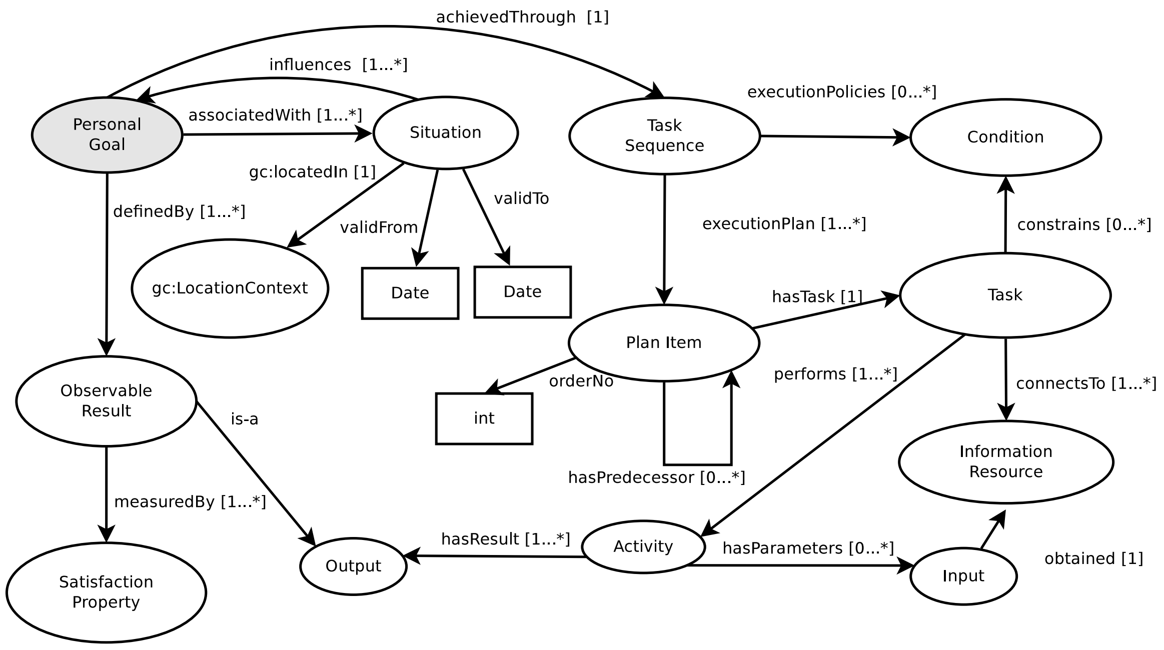


**Figure 5.10:** Simplified view of our Galapagos Metamodel to show core concepts [CVM14c]

An **entity** represents a concept that is one of four types: cyber element, physical element, human element, or smart interaction. *Types* are based on the type of concept within the problem domain. Some *types of concepts* are explicit, such as *PersonalGoal*, which belongs to the user (*online grocery shopping*), and thus is an element in the human dimension. However, the type for other concepts might not be so obvious, such as *Task*, *InformationResource* and *TaskSequence*.

The entity *Task* corresponds to the concept of *internet task*, which in UCSAs is a functionality provided by a software application. The result of a functionality is an output that will be used in the sequence to achieve a user's personal goal. Following our example, an internet task is to *update the grocery shopping list*, which is a functionality provided by a *shopping list application* whose output is a modified list of items. As a result, the *Task* entity is the concept of an *internet subtask*, which is an element in the cyber dimension.

The entity *InformationResource* represents the concept of an *information resource* and can be interpreted as a cyber element (e.g, web service) given that information resources live as virtual objects. However, based on their nature within the problem domain, information resources are data elements that are exchanged and stored, and provide no functionality or logic associated. Resources are objects consumed and read as they are, accessed through services. As a result, the entity *InformationResource* represents an element of the physical dimension.

Finally, an entity for the concept of a *smart interaction* is the one that abstracts the connection of two dimensions with the user-centric principle perspective. For example, the entity *TaskSequence* is the concept of *task sequence*, which is an ordered set of connected internet subtasks (cyber dimension) that assists users in the fulfilment of a personal goal (human dimension). An internet task sequence is selected and executed based on users' personal context.

A **link** is a connection between two entities in our model. For example, the link named *achievedThrough* connects the entities *PersonalGoal* and *TaskSequence* with a cardinality of one. The direction of the link represents a dependency in which a *PersonalGoal* relies on the existence of a *TaskSequence*. The cardinality means only one *TaskSequence* is expected.

Cardinalities express if the connection is mandatory or optional. For example, the cardinality [1...*] of the link *measuredBy*, which connects *PersonalGoal* and *ObservableResult* indicates that it is required a minimum of one *ObservableResult* to assess the fulfilment of a *PersonalGoal*. Cardinalities with zero determines that a connection might not exist. For instance, the cardinality [0...*] in the link *hasParameters* indicates that an *Activity* might not have any parameters.

There are links with no cardinality that are used to connect entities with data types. For instance, the link *is-a* between *ObservableResult* and *Output* determines that the first is an entity type of the second. Similarly, other links are used to determine data types of entity attributes. For example, the link *orderNo* determines that the entity *PlanItem* has an attribute *orderNo* and it is a number.

### Situation-Awareness Components

The entity *Situation* is the key element to support situation-awareness in User-Centric Smart Cyber-Physical-Human Applications. It represents the concept of a *situation* defined as any user event that might described in terms of two properties: *time* and *space*. The **time** property comprises two data properties *validFrom* and *validTo*, which are of type Date. The property **location** is defined though the link *gc:locatedIn* and the entity *gc:LocationContext* of the SmarterContext *GeneralContext* ontology proposed by Villegas [Vil13]. It is worth noting that situations can be *permanent*, which means they don't expire; and *omnipresent*, which means it is valid in all places at all times (e.g., the user is vegan). Figure 5.11 presents a simplified view of our Galapagos Metamodel to show the entities for situation-awareness support.

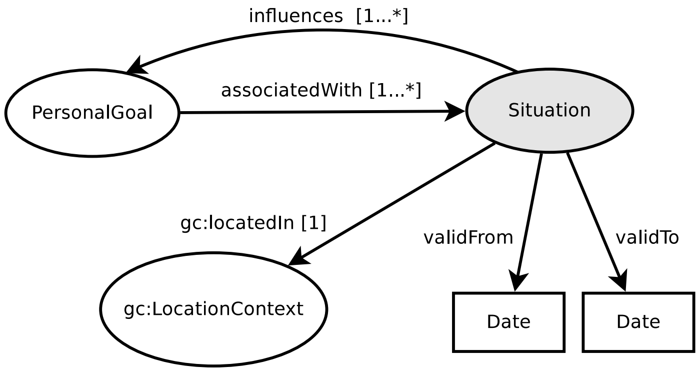


**Figure 5.11:** Simplified view of our Galapagos Metamodel to show situation-awareness components [CVM14a]

In UCSAs, a personal goal is associated with one to many situations, and a situation influences the achievement of one to many personal goals. In our *online grocery shopping* example, there is a policy requiring that the grocery stores and the user are in the same city.

Through the link *associatedWith [1...*]* it is determined that the execution of online grocery shopping relies on the user's situation of being at certain location while achieving her goal. The link *influences [1...*]* represents situations that influence the execution of the personal goal. For example, if the user books a flight to another city , the system will modify her location thus affecting the execution of her online grocery shopping.

### 5.2.2 Galapagos Model

The Galapagos Model uses *G-iStar*, our *i** extension for modelling User-Centric Smart Cyber-Physical-Human Application. *G-iStar* is an adaptation of the iStar framework elements [JHG13, DFH16] to support the specification of evolving tasking goals, personal interactions, and the relevant contexts. In particular, we extended the iStar atomic notions of actor, goals, task and resources, to support the specification of tasking goals, task sequences, and relationships among goals, tasks, actors, and resources. Figure 5.12 depicts *G-iStar*.

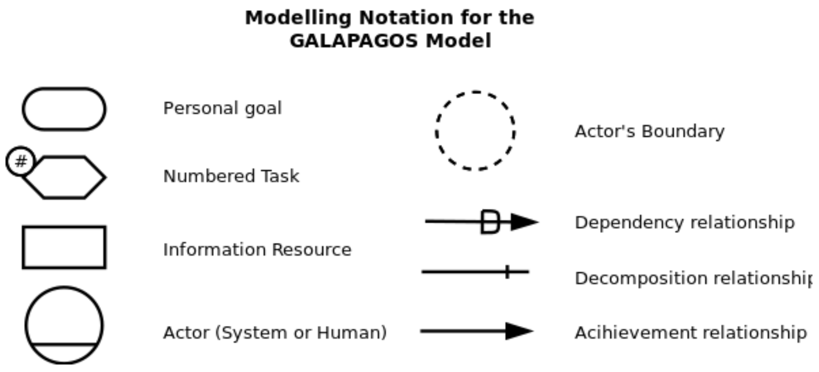


**Figure 5.12:** G-iStar notation

Figure 5.13 depicts a simplified instance of the Galapagos Model for our online grocery shopping scenario. Based on the classic notation, the goal of *online grocery shopping* is achieved through the execution of eight internet tasks (labelled 1 to 8).

Task 1 is to *shop grocery items*, which in turn is decomposed into Tasks 2, 3, and 6: *get grocery items list*, *search proper stores*, and *purchase items per store* Each one of these tasks have different compositions and connections. For example, Task 2 depends on the *list of items* the users wants to purchase, which is an information resource. The access to that resource requires certain activities, such as locating the grocery shopping list application, authenticating with the user's credentials, reading

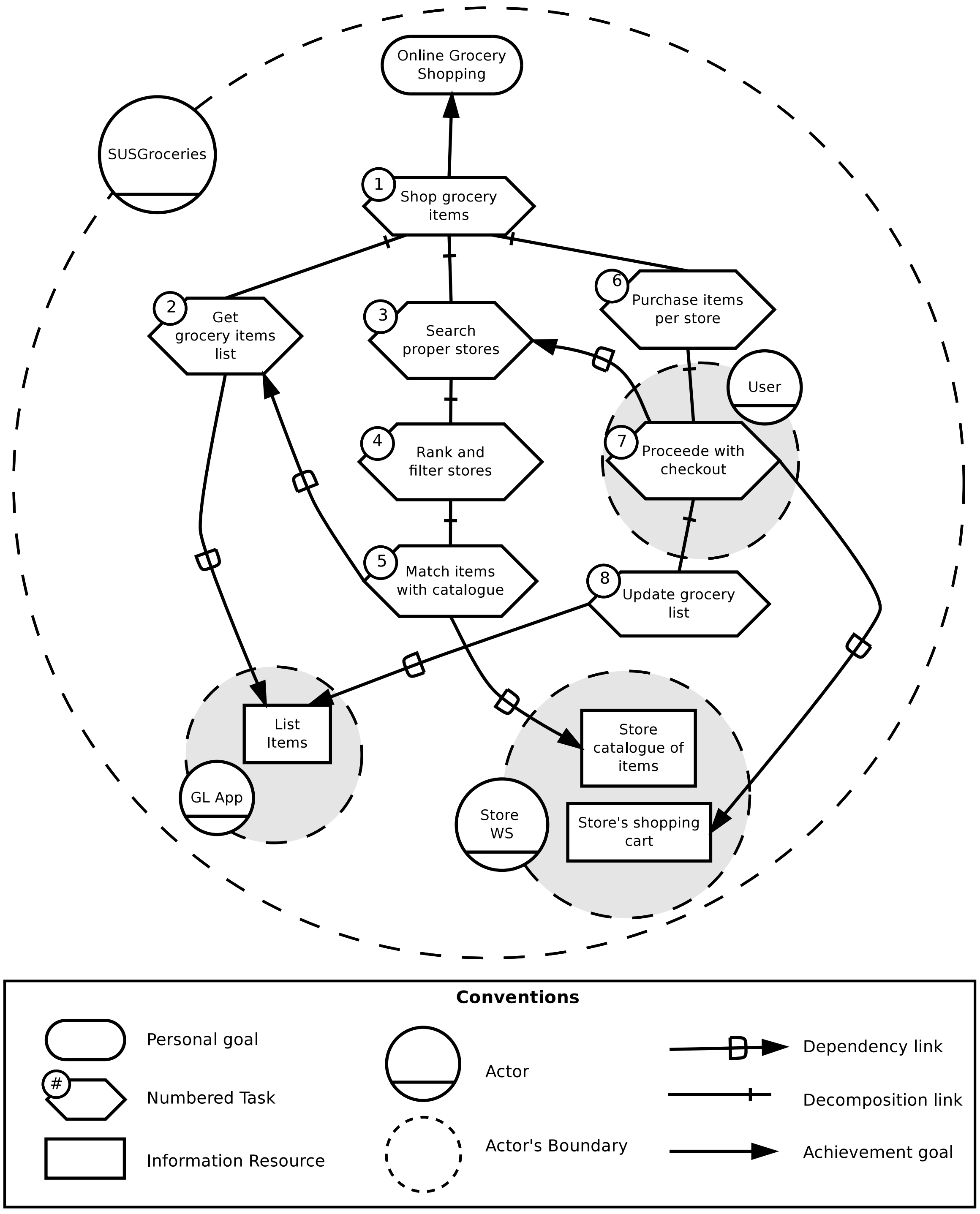


**Figure 5.13:** Simplified instance of the Galapagos Model for online grocery shopping

the grocery list items, and applying personalized formatting. Task 3 is part of a decomposed task (cf. *Subtask 2* in Figure 5.5) along with other on two tasks: Task 4, which exploits personal context information to select and rank proper grocery stores, and Task 5, which uses the grocery list to match appropriate stores with items on the list. It is worth noting that Task 5 depends on the execution of Task 2 and requires access to additional information resource from the stores to get their inventory. Task 6 is decomposed into Tasks 7 and 8. Task 7 depends on the execution of Task 3, and requires an information resource of the store, which is the shopping cart. Task 8 accesses a shared resource, the *grocery list items*, and updates the list based on the outcomes of the purchase.

Actors in our model are used to describe who is responsible for the goal, task, or resources. Our model instance describes four actors: (1) *SUSGroceries*, which is our mobile application example for online grocery shopping; (2) the *user* of the system; (3) the *grocery list application*; and (4) the *grocery store web service*, which are two external systems.

The boundaries of each actor are represented by a dashed circle. It is worth noting that even though the user is at the centre of all this interaction, we presented the example where it is a policy of the system that the user's intervention is required to execute the credit card purchase. Therefore, the user is seen as responsible for the execution of Task 7 in our model instance, *proceed with checkout*.

A personal goal is fulfilled through the execution of an ordered sequence of internet tasks, thus our model can represent the sequence as well as the dependency among tasks, and the decomposition into subtasks. Tasks often require to access information in the cyber or physical dimensions, represented in our model as information resources. Our GALAPAGOS MODEL provides a mechanism to represent the composition and execution of UCSAs.

It is worth pointing out that the model described above is a simplified instance of the MART. The complete version includes all relevant information that User-Centric Smart Cyber-Physical-Human Applications require (cf. Appendix A). For example, the activities of each task, the paths to access the information resources, and the measurable outputs to determine the fulfilment of the goal.

## 5.3 Chapter Summary

This chapter presented our Model at Runtime (MART) for User-Centric Smart Cyber-Physical-Human Applications (UCSAs): (1) Our GALAPAGOS METAMODEL, which represents the concepts of UCSAs; and (2) our GALAPAGOS MODEL, which represents the evolving internet tasking goals, relevant context, and smart interactions of CPHSs. These MARTs constitute the second contribution of this dissertation and allow us to answer research questions RQ2, and RQ3 (cf. Figure 1.2).

First, we determined modelling requirements for UCSAs and grouped them into three concerns: personal goals, tasks, and context. Since software models already exist for some elements of the modelling requirements, we studied related approaches and found that goal models are a suitable starting point to represent the application domain of UCSAs, and graphs are fitting implementations of goal models.

Then, we presented our two MARTs. Our Galapagos Metamodel represents the conceptual component of UCSA. We presented the core elements of the metamodel and their relation with our application domain of UCSAs, and the situation-awareness support by defining the entity *Situation* and providing it with time and space descriptors. More importantly, in our Galapagos Metamodel personal goals and situations are connected in two manners: (1) situations influencing the achievement of personal goals, and (2) personal goals associated with situations.

Our Galapagos Model is specified using *G-iStar*, our extension and adaptation of the iStar framework to support dynamic personal goals and task interactions. Our model supports the specification of evolving tasking goals, personal interactions, and the relevant contexts. We explained our model through the scenario of achieving a personal goal of online grocery shopping.

MARTs are also software artefacts, the next chapter presents our operational framework, which defines the supported runtime operations and semantics of our MARTs and is necessary for the *MARTs supporting infrastructure* to effect runtime adaptation.

# Chapter 6

# Operational Framework for Models At Runtime

This chapter presents our operational framework for Models at Runtime (MARTs) in User-Centric Smart Cyber-Physical-Human Applications (UCSAs), which is our third contribution of this dissertation. Our framework defines model equivalences between human-readable and machine-readable, available runtime operations and semantics, to manage runtime operations on MARTs.

There is one research question associated with this contribution:

RQ4. *What are the runtime infrastructures required to process and evolve runtime models while maintaining the causal relations among them for User-Centric Smart Cyber-Physical-Human Application*s*?*

To answer this question, we define an operational framework that comprises four components: (1) a notation-artefact mapping; (2) a catalogue of operations; (3) the runtime semantics; and (4) causal links. The notation-artefact mapping connects every element that is in the model notation form with its corresponding element in the software artefact form. The mapping is the main element for the translation of the MART in the two ways: human readable and machine readable. The catalogue defines CRUD operations for every element of the model. It also defines restrictions, precondition, post-activities and other considerations when performing runtime operations. Although, we defined three operations, the catalogue is flexible to any supported runtime operation of a MART. Finally, the runtime semantics are specified in a programming language and are implemented by software infrastructures to execute the operations at execution time. A fourth component of our operational framework is the *causal links*. This component is shared across MART in the framework and used by software systems to propagate changes among models that are connected. Figure 6.1 depicts our operational framework between the two views of each MART.

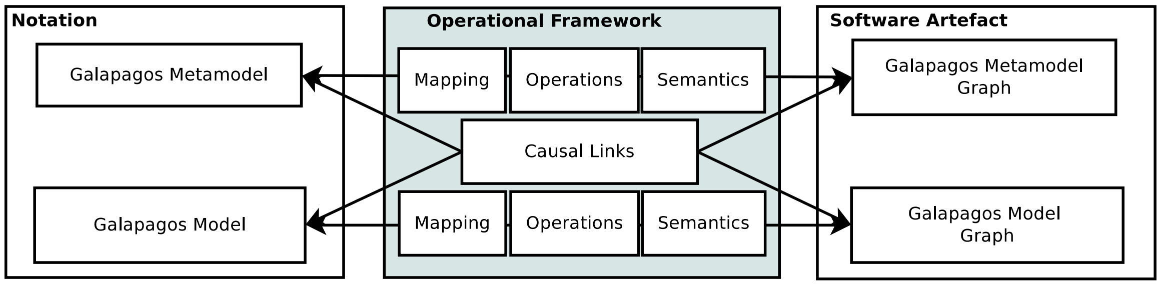


**Figure 6.1:** Overview of our operational framework for our MARTs

This chapter is organized as follows. Section 6.1 presents the mapping, catalogue and runtime semantics for our Galapagos Metamodel. Section 6.2 presents the mapping, catalogue and runtime semantics for our Galapagos Metamodel. Section 6.3 presents the causal link of our two MART. Finally, Section 6.4 summarizes the chapter.

## 6.1 Galapagos Metamodel

Our Galapagos Metamodel describes the conceptual elements of UCSA. As a MART, it allows smart infrastructures to read the model and identify themselves as a User-Centric Smart Cyber-Physical-Human Application. Since this model represents the application domain, we envision low vulnerability to changing situations. More importantly, we define that variations in the model might change their true nature therefore representing a different type of system than a UCSA. As a result, the runtime operations supported in our MARTs are limited and relatively few.

### 6.1.1 Mapping from Notation To Software Artefact

The first component of our operational framework is a mapping between the two views: the MART notation and the MART software artefact (cf. Galapagos Metamodel and graph in Figure 6.1). Our Galapagos Metamodel comprehends two components: entities and links presented in Section 5.2.1. As a software artefact, entities and links become nodes and arcs respectively. However, there are restrictions on what and how nodes are connected. Table 6.1 presents the adjacency matrix between

**Table 6.1:** Adjacency table between the elements of our MART.

| **Entity** | 1 | 3 | 7 | 9 | 10 | 11 | 12 | 14 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Activity | - | - | - | - | L | - | - | - | L | - | - | - | - |
| Algorithm | U | - | - | - | - | - | - | - | - | - | - | - | - |
| Context Condition | - | U | - | - | - | - | - | - | - | - | - | - | - |
| Context Input | - | - | - | - | L | - | - | - | - | - | - | - | - |
| Data | - | - | - | L | - | - | - | - | - | - | - | - | - |
| Execution Condition | - | L | - | - | - | - | - | - | - | - | - | - | - |
| Execution Time | - | - | L | - | - | - | - | - | - | - | - | - | - |
| Input | - | - | - | L | - | - | - | - | - | - | - | - | - |
| Internet Subtask | L | L | - | L | - | - | - | - | - | - | - | - | - |
| Internet Task Sequence | - | L | - | - | - | - | - | - | - | - | L | - | - |
| Local Resource | - | - | - | U | - | - | - | - | - | - | - | - | - |
| Observable Result | - | - | - | - | - | - | - | - | L | - | - | L | - |
| Operations | U | - | - | - | - | - | - | - | - | - | - | - | - |
| Personal Goal | - | - | - | - | - | - | L | L | - | - | - | - | L |
| Plan Item | - | - | - | - | - | L | - | - | - | - | L | - | - |
| Situation | - | L | - | - | - | - | - | - | - | L | - | - | - |
| User Input | - | - | - | - | U | - | - | - | - | - | - | - | - |
| Web Service | - | - | - | U | - | - | - | - | - | - | - | - | - |

LEGEND: Column numbers correspond to the Entities listed in Table 6.2.
*Entities that are not connected have been omitted from this table.*

the nodes. Rows determine the node of origin and columns the target. Arcs contain a *locked* attribute, which has values true or false. In our table is marked as L or U, for locked and unlocked respectively. Dash values indicate that the two nodes cannot be linked. When the *locked* attribute is true, it means that the entity is mandatory for the ontology and is not allowed to change at execution time.

For example, the row-column pair (4,3) corresponds to the relation between a *Context Condition* (row 4) and a *Condition* (column 3). The *locked* attribute is false (value U in the table), which determines that the entity *ContextCondition* can be modified or removed from the model. Possible modifications include replacing it with another context entity from the SmarterContext ontology or another entity in compliance with the conceptual definition of UCSA.

In contrast, the pair (17,14) that corresponds to the relation *definedBy* between *Personal Goal* and *Observable Result*, has a value of true for the attribute *locked*. It is clear that the relationship between this two elements should not be exposed to runtime modifications because it belongs to the core concepts of UCSA.

| | |
|---|---|
| 1 Activity | 12 Internet Task Sequence |
| 2 Algorithm | 13 Local Resource |
| 3 Condition | 14 Observable Result |
| 4 Context Condition | 15 Operations |
| 5 Context Input | 16 Output |
| 6 Data | 17 Personal Goal |
| 7 Execution Condition | 18 Plan Item |
| 8 Execution Time | 19 Satisfaction Property |
| 9 Information Resource | 20 Situation |
| 10 Input | 21 User Input |
| 11 Internet Subtask | 22 Web Service |

**Table 6.2:** Legend for column numbers in Table 6.1

### 6.1.2 Catalogue of Operations

Our ontology describes the conceptual definition of UCSA. We anticipate that there will no be new entities or links into the ontology at execution time. Likewise entity deletion is not considered in our catalogue. We take this assumption on the basis that our ontology defines the nature of a software application and major modifications produced by adding or deleting entities and links will generate a new definition for a new type of application. As a result, our catalogue defines operations for unrestricted objects, that is, updating, adding or deleting attributes of entities and links. Table 6.3 presents a summarized view of the catalogue of operations for our Galapagos Metamodel. Unspecified *Entities* imply that there are not supported operations. There are three types of operational behaviours described in Table 6.3: (1) *Yes*, which defines that the pair Element-Operation is allowed with no restrictions. For example the pair *All-Add* defines that for all the entities and links of the ontology, it is allowed to *add* descriptive attributes. (2) *No*, defines that the pair Element-Operation is not allowed with no exception. For example, the pair *All-Update* defines that it is not allowed to *update* any attributes that is *locked*. (3) *With validation (W-Val)*, defines that the operations require validation (policy based or human intervention) before executing the operation. For example, the pair *Activity-Add* defines that an *Activity* can be added to the ontology with a validation of its relationship with the other entities it will be connected.

**Table 6.3:** Catalogue of operations for our GALAPAGOS METAMODEL.

| **Element-Operation** | **Add** | **Delete** | **Update** |
|---|---|---|---|
| **All** | `Yes` Descriptive attributes | `W-Val`: Descriptive attributes. `No`: Locked attributes | `Yes`: Descriptive attributes. `No`: Locked attributes. `W-Val`: cardinality if unlocked |
| **Activity, Condition, Execution Condition, Information Resource, Input** | `W-Val`: Type of relationship with the origin/target Element | `W-Val`: Type of relationship with the origin/target Element | `W-Val`: Type of relationship with the origin/target Element |

### 6.1.3 Runtime Semantics

The last element of the framework is a definition of runtime semantics for the operations described in the catalogue. The runtime semantics of a MART comprise two elements: (1) *Sentences*, which specify patterns used by software systems to validate the correctness of requests for operations, and invoke the corresponding code functions; and (2) *Code*, which is the implementation of the operations described in the catalogue.

The sentences expressed with our semantics have the following structures: (1) action-object-object, and (2) action-object. An action is one of the CRUD operations. The object at the end refers to the target subject of the change. The object in the middle of (1) is the one carrying the action. If there is one object, it means it is both carrier and target.

We use Java to specify our semantics, using objects to translate our concepts, and function calls to represent the operations. We chose Java because it allows the implementation of conditions and loops giving flexibility to our semantics. Moreover, Java is a well-known language in the domain of software models, there are various modelling implementations and tools supported by Java, such as the Eclipse Modelling Framework (EMF),[1] and IntelliJ IDEA.[2] Java allows our framework to be extensible and implemented by other applications fairly easily.

As described in the catalogue, runtime operations are supported as a pair *Element-Operation*. To explain our semantics, we present three examples of *Element-Operation*

[1]http://www.eclipse.org/modeling/emf
[2]https://www.jetbrains.com/idea

pairs. We selected three elements: two entities *PersonalGoal* and *Situation*; and the link *associatedWith*. Figure 6.2 presents a simplified view of the classes, we use for the specification of our runtime semantics.[3]

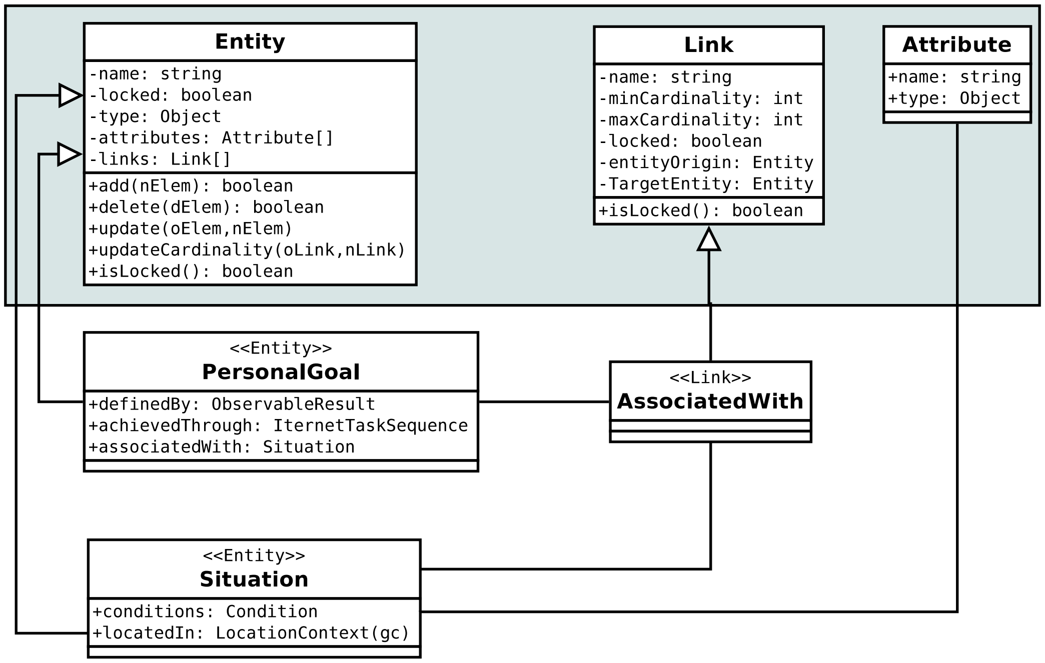


**Figure 6.2:** Simplified view of the Java classes used in the implementation of our runtime semantics for our Galapagos Metamodel

We created three main classes: `Entity`, `Link` and `Attribute`. These classes describe the main elements in the ontology, and every entity in our model is a specialized version of the three. For instance, the classes `PersonalGoal` and `Situation` are specific versions of `Entity`.

## Element-Add

Figure 6.3 depicts the sentence of adding a new attribute to the entity `Situation`. Source Code 6.1 depicts a simplified view of the Java code for this sentence. According to our catalogue of operations, it is allowed to add descriptive attributes for all entities. For this action, our semantic follows the sentence structure action-object-object. The corresponding method is `+add(nElem):boolean`, which has for parameter the new element and returns a boolean true or false if the operation is

[3]The complete implementation contains classes for all entities and links of the Galapagos Metamodel. Available at http://rigiresearch.com/research/pit

successful or not respectively. Since the target is an object of `Situation`, the method gets called from its instance.

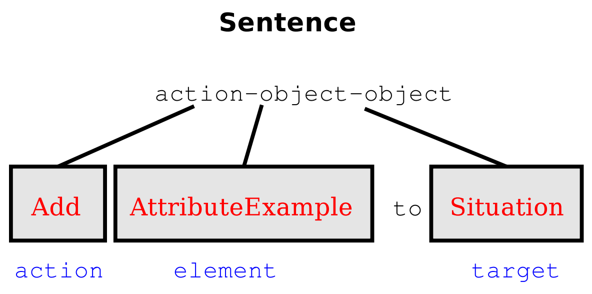


**Figure 6.3:** Sentence example for adding a new attribute to an `Entity`

**Source Code 6.1:** Java code example for adding a new attribute to an `Entity`

```java
public class Entity{
  public boolean add(nElem)   throws EntityException,
      RuntimeException{
    if(isValid('add',nElem)){
      if(nElem instanceOf Attribute){
        this.attributes.push(nElem);
        return true;
      }
    /* ... */
    }else
      return false;
  }

  private boolean isValid(action,element) throws
      ValidationException, RuntimeException{
    if(this.locked) return false;
    switch(action){
      case 'add' :
        //All entities can add descriptive attributes
        if(element instanceOf Attribute){
          return true;

    }
    /* ... */
  }
}
```

The method `+add()` calls `-isValid()` in the same class. The validations within the method follows the restrictions described in our catalogue of operations. First, it evaluates if the entity allows changes. If it is locked the method returns false, and the addition does not take place. On the contrary, if the entity is unlocked, it proceeds to validate the operation based on the action and element's type. In this example, returns true for the action *add* with an object type `Attribute`. As a result, the method proceeds to append the new element in the array of attributes of the object `Situation`.

### Element-Delete

Figure 6.4 depicts the sentence of attempting to delete the link `associatedWith` from the entity `PersonalGoal`. Source Code 6.2 depicts a simplified view of the Java code for this sentence. In this case, the method `+del(nElem):boolean` calls `-isValid(action,element):boolean` in which case returns false, given that the link `AssociatedWith` is locked for changes according our table of relations Table 6.1.

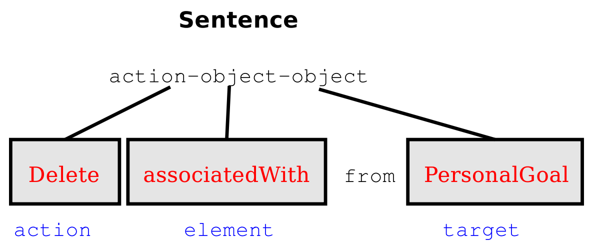


**Figure 6.4:** Java code example for removing a `Link` from an `Entity`

**Source Code 6.2:** Java code example ffor removing a `Link` from an `Entity`

```java
public class Entity{
  public boolean delete(nElem)
    throws EntityException, RuntimeException{
    if(isValid('delete',nElem)){
      /* ... */
    }else
      return false;
  }

  private boolean isValid(action,element)
    throws  ValidationException, RuntimeException{
    if(this.locked) return false;
    switch(action){
      case 'delete' :
      //Attributes might be deleted (validate)
      if(element instanceOf Attribute){
        return validate(action,elem,this);
      }
      /* ... */
    }
  }
}
```

### Element-Update

Figure 6.5 depicts the sentence of updating the entity `Situation`. Source Code 6.3 depicts a simplified view of the Java code for this sentence. Based on our restrictions, entities can only update their attributes. Therefore, the method `+update(nElem):boolean` only updates the array of attributes, ignoring other elements of `nElem` that might be different from the original.

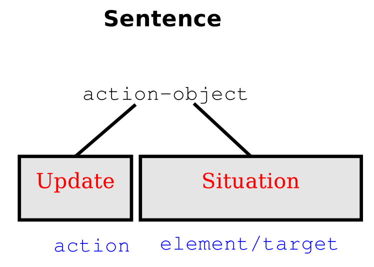


**Figure 6.5:** Java code example for updating an `Entity`

**Source Code 6.3:** Java code example for updating an `Entity`

```
public class Entity{
  public boolean update(nElem)
    throws EntityException, RuntimeException{
    if(isValid('update',nElem)){
      for(i in this.attributes){
        if(!this.attributes[i].isLocked()
        && this.attributes[i].equal(nElem.attribute) ){
          this.attributes[i] = nElem.attributes[i];
          return true;
        }
      }
    /* ... */
    }else
    return false;
  }

  private boolean isValid(action,element)
    throws  ValidationException, RuntimeException{
    if(this.locked) return false;
      switch(action){
      case 'update' :
      //All entities can update unlocked attributes
      if(element instanceOf Entity){
        return true;
      }
    /* ... */
    }
  }
}
```

## 6.2 Galapagos Model

### 6.2.1 Mapping From Notation To Software Artefact

To transform from our goal model specification into a graph implementation (nodes and arcs) and vice versa, our operational framework implements a mapping of elements grouped into two concerns: components and relationships.

- **Components:** Objects, such as *goal, task, resource*, and *actor* are represented as nodes. Connectors, such as *achieve, decompose, depend*, and *responsible* are represented as arcs. Figure 6.6 presents the graph view of our MARTs instance for the online grocery shopping example depicted in Figure 5.13.

- **Relationships:** Table 6.4 presents the relationships between all four mode types. Every relationship is described based on the possible connection type. A row represents the node of origin and a column the target node.

  A *Goal* has only an origin relation with other goals in the form of *dependency* or *decomposition*. That is, a *goal* depends on the fulfilment of another separated *goal*, or the *goal* has *subgoals*. In the same way, a *task* has an origin relation with other *tasks*. A *task* has an *achievement* relation to a *goal*. At a minimum

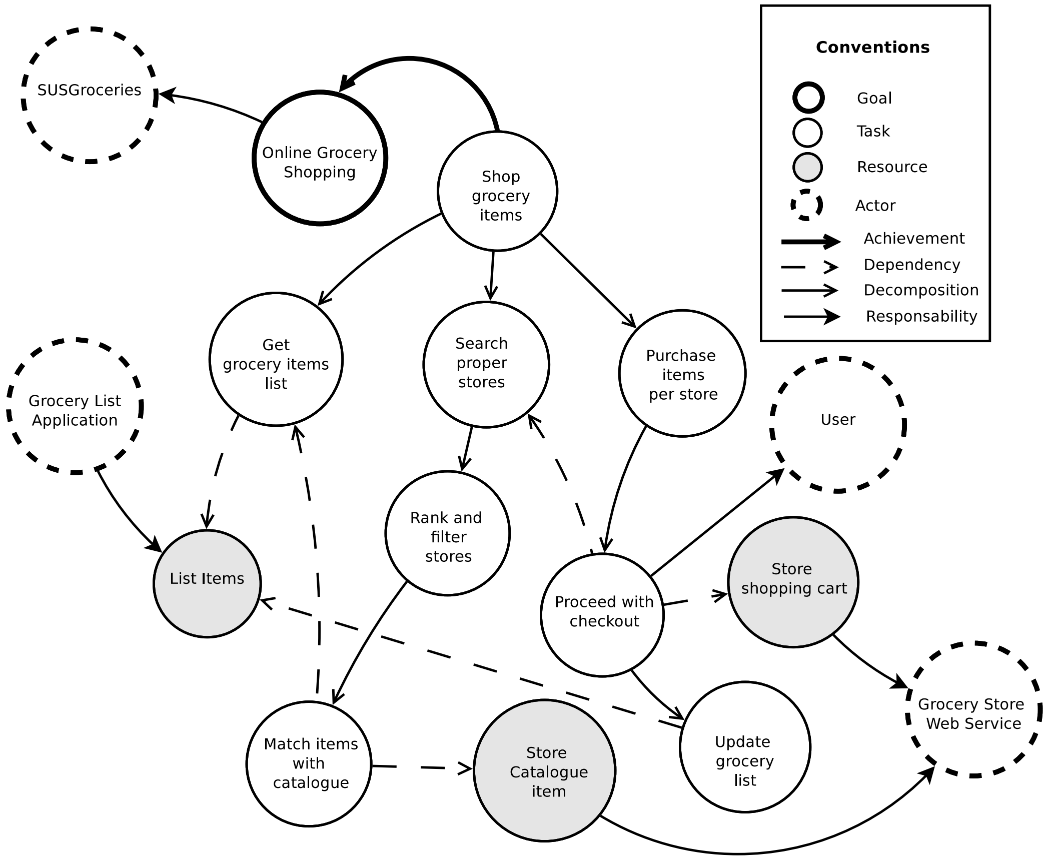


**Figure 6.6:** Our GALAPAGOS MODEL instance graph for online grocery shopping

one task is required to fulfil one *goal*. A *resource* has only *dependency* relations with *tasks* or other resources. Because the relationship with an *actor* defines *responsibility*, an *actor* is a target responsible for a a *goal*, *task* or *resource*. No relationships are originating from an *actor*.

**Table 6.4:** Relationships between the elements of our GALAPAGOS MODEL. *LEGEND: achievement (a), dependency (dp), decomposition (dc), and responsibility (r)*

| | **Goal** | **Task** | **Resource** | **Actor** |
|---|---|---|---|---|
| **Goal** | dc, dp | - | - | - |
| **Task** | a | dc, dp | dp | - |
| **Resource** | - | - | dp | - |
| **Actor** | r | r | r | - |

### 6.2.2 Catalogue of Operations

Table 6.5 summarizes the conditions or the runtime operations on the elements of the Galapagos Model. There are four types of operation behaviours described in this table: (1) *Yes*, which defines that the pair Element-Operation is allowed with no restrictions. For example, for the pair *Connector-Update* it is allowed to for any connection to *update* its target element. (2) *No*, defines that the pair Element-Operation is not allowed without exception. For example, the pair *All-Delete* describes that it is not allowed to *delete* locked attributes. (3) *With validation (W-Val)*, defines that the operations require validation (policy based or human intervention) before executing the operation. For example, the pair *Goal-Update*, defines that to *update* a goal is necessary to validate the task sequence, measure outcomes and execution conditions. (4) *Precondition (PRE)*, defines activities that need to be executed before the operation. For example, the pair *Task-Add* defines that to *add* a Task is necessary that a Goal or Task is defined as parent. (5) *Post activity (POST)*, defines activities that need to be executed after the main operation. For example, the pair (Goal-Delete) defines that after removing a goal there should be an activity to remove children elements. Moreover, our catalogue uses the logical operations AND and OR.

### 6.2.3 Runtime Semantics

The sentences expressed with our semantics have the following structures: (1) action-object-object, and (2) action-object. Action and object have the same meanings as the semantics for our Galapagos Metamodel. Likewise, we use Java to specify our semantics using objects to translate elements, and function calls to represent operations.

As described in the catalogue, runtime operations are supported in a pair *Element-Operation*. To explain our semantics, we present three examples of *Element-Operation* pairs. We selected three elements namely *Goal, Task*, and *Information Resource*; and the connector *Decomposition*.

Figure 6.7 presents a simplified view of the classes, we use for the specification of our runtime semantics for our example. We define three main classes: `Element`, `Connector`, and `Attribute`. These are used to describe the main elements in the Galapagos Model, and every component in our model is a specialized version of these three. For instance, the classes `Goal`, `Task`, `InformationResource` are specialized versions of `Element`.

**Table 6.5:** Catalogue of operations for our Galapagos Model

| **Element-Operation** | **Add** | **Delete** | **Update** |
|---|---|---|---|
| **All** | `PRE`: all required attributes | `No`: Locked attributes | `No`: Locked attributes |
| **Goal** | `PRE-Optional`: Parent Goal (for subgoals) | `No`: Other goals have dependency relations with the goal. `POST`: waterfall removal of the branches based on the connectors: achievement and decomposition | `W-Val`: task sequences, measure outcomes, execution conditions |
| **Task** | `PRE`: Parent (goal OR task) | `No`: if there are dependency relations to the task. `W-Val`: if the task has children | `W-Val`: task sequence |
| **Information Resource** | `PRE`: Dependency relation (task OR actor OR resource) | `W-Val`: if there are dependency relations to the resource | |
| **Actor** | `PRE`: Element of responsibility (goal OR task OR resource) | `No`: if the actor is responsible for a goal. `W-Val`: If there are responsibility relations from the actor | `W-Val`: if the actor is responsible for a goal |
| **Connectors** | `PRE`: between two compatible elements | `PRE`: validate branch removal AND prepare branch for removal. `POST`: Remove orphan branch | `No`: Origin element. `Yes`: Target element. `PRE`: Validate compatibility and new branch composition |

### Element-Add

Figure 6.8 depicts the sentence of adding a new internet task attribute to a goal. Source Code 6.4 depicts a simplified view of the Java code for this sentence. It is worth mentioning that our example is a simplified version of our Java code. The complete implementation includes private and public methods required to support the entire operations as defined in our catalogue.

The method `+add(nElem):boolean` verifies that the element is valid for the operation calling the private method `-isValid(action,element):boolean`. Based on our catalogue this method makes sure all required attributes are not null. When the operation is valid, the add method verifies the type of element being added. In this case, we are adding an `Task` and based on our catalogue a new task requires a parent, either a goal or another task. If it is a goal (as in this example) it verifies if there is a sequence associated with the achievement of a goal. If this is the first task then it

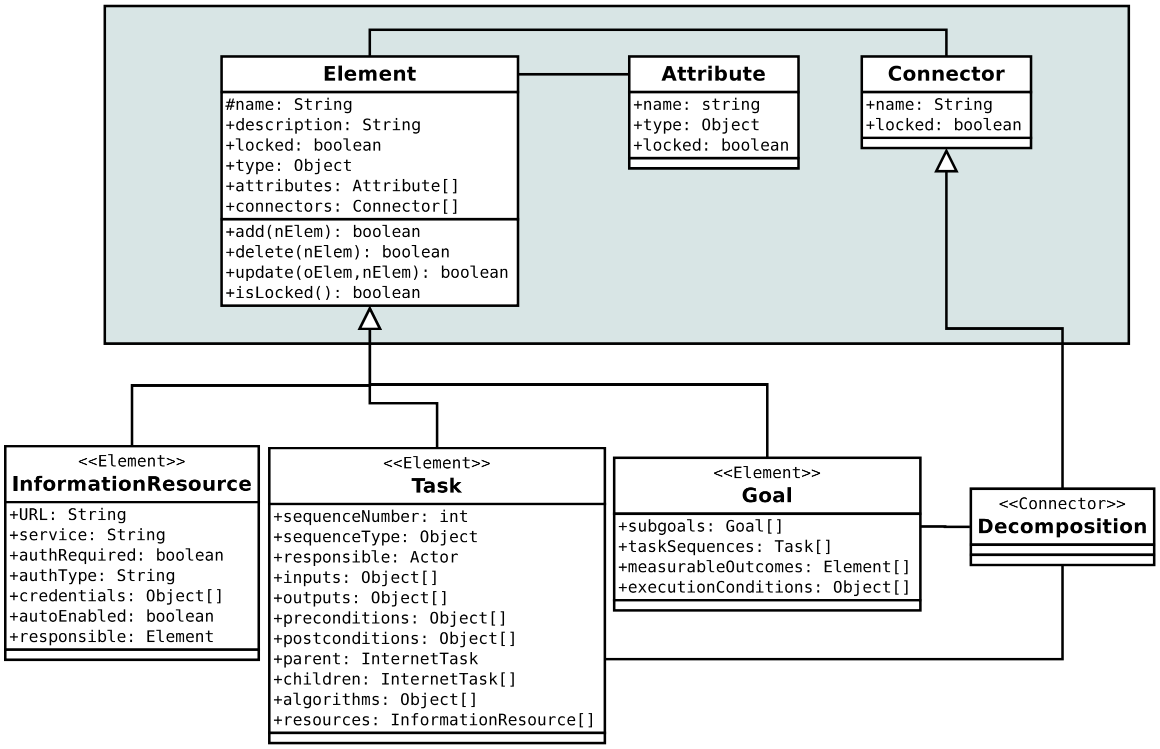


**Figure 6.7:** Simplified view of the Java classes used in the implementation of our runtime semantics for our GALAPAGOS MODEL

becomes the first element in the sequence. Otherwise starts to search its parent by calling the private method `-getTask(taskName):Task`.

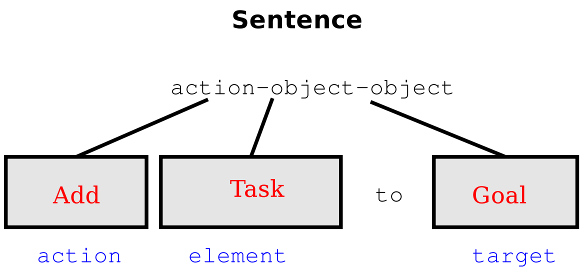


**Figure 6.8:** Java code example for adding a new internet task to a goal

**Source Code 6.4:** Java code example for adding a new internet task to a goal

```
public class Element{
  public boolean add(nElem)
    throws ElementException, RuntimeException{
    if(isValid('add',nElem)){
      if(nElem instanceOf Task
        && this instanceOf Goal){
```

```
        if(this.taskSequeces[0]==null){
          this.taskSequeces[0] = nElem;
          return true;
        }else{
          Task parent =
             this.taskSequeces[0].getTask(nElem.parent.name);
          nElem.setParent(parent);
          nElem.parent.updateChildren(nElem);
          nElem.updateChildren();
          return true;
        }
      }
    }
    /* ... */
    return false;
  }

  private boolean getTask(taskName)
    throws ElementException, RuntimeException{
    Task initTask = null;
    if(this instanceOf Goal){
      return taskSequences[0].getTask(taskName);
    }
    if(this instanceOf Task){
      if(this.name.equals(taskName)) return this;
      else
      if(this.children.length==0) return null;
      else
      for (i in this.children){
        initTask = this.children[i].getTask(taskName);
        if(initTask != null) return initTask;
      }
    }
    return initiTask;
  }
}
```

### Element-Delete

Figure 6.9 depicts the sentence for removing an information resource from the model. Source Code 6.5 depicts a simplified view of the Java code for this sentence.

The sentence follows the structure action-object, which implies the action is to be performed over the object as an element and a target as well.

The method `+delete(nEleme):boolean` validates if the element is valid to be removed. Based on the type of element the method validates restrictions as described in our catalogue of operations. As an `InformationResource` instance, it validates if the resource has associated dependencies. If true, it verifies the policies to delete them. For example, in case two the policy indicates a waterfall delete, this is, to remove all connected dependencies (and elements if unlocked). Waterfall starts with the main goal.

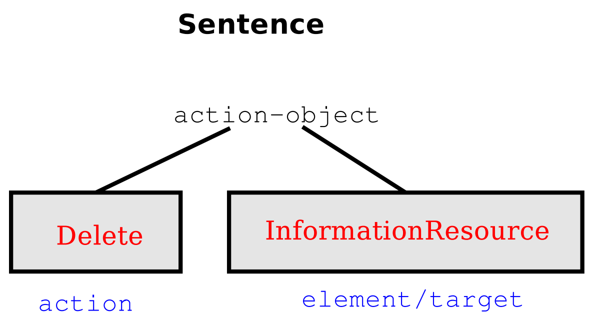


**Figure 6.9:** Java code example for removing an information resource

**Source Code 6.5:** Java code example for removing an information resource

```java
public class Element{
  public boolean delete(nElem)
    throws ElementException, RuntimeException{
    if(isValid('delete',nElem)){
    if(this instanceOf InformationResource)}
        if(nElem.hasDependencies()){
          switch(nElem.verifyDeletePolicy()){
            case 1: //can not delete if dependencies
              return false;
            case 2: // waterfall delete
              Goal mainGoal = findGoal(this);
              return mainGoal.waterfallDelete(nElem);
            default: return false;
          }
        }else{
          Goal mainGoal = findGoal(this);
          return mainGoal.waterfallDelete(nElem);
        }
        }
      }
      /* ...*/
      return false;
    }
}
```

## Element-Update

Figure 6.10 depicts the sentence for updating the decomposition of a task. Source Code 6.6 depicts a simplified view of the Java code for this sentence.

In this case, an internet task will have a new decomposition of subtasks. The element `Decomposition` contains two objects of the same type (i.e., `Task`) for origin and target. The origin will contain one internet task, the one to be decomposed. The target contains a set of subtasks. The method `+update(nElem):boolean` verifies the operation and performs the validations based on the type of element in accordance to our catalogue of operations. In a decomposition, it is important to validate that the new sequence will continue to provide the outputs required by the main task that is being decomposed. The method `-verifyAlterSequence(Task[]):int` verifies that the output requirement of the task corresponds to the output of the various subtasks in the decomposition. If the new decomposition is compatible, the children become the set of `Task` in the `Decomposition` target property.

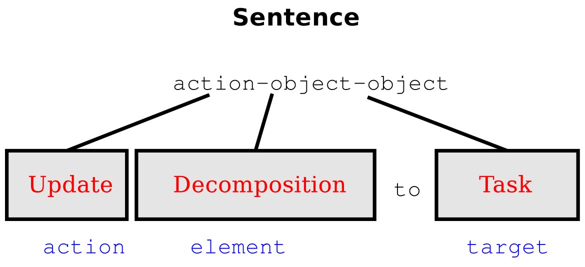


**Figure 6.10:** Java code example for updating the decomposition of a task

**Source Code 6.6:** Java code example for updating the decomposition of a task

```
public class Element{
  public boolean delete(nElem)
  throws ElementException, RuntimeException{
    if(isValid('update',nElem)){
      if(nElem instanceOf Connector && nElem.origin instanceOf
          Task)}
```

```
        Task origin = findTask(nElem.origin.name);
        switch(origin.verifyAlterSequence(nElem.target.children)){
          case 1: // not compatible
            return false;
          case 2: // compatible
            origin.children = nElem.target.children;
            return true;
          default: return false;
        }
      }
    }
    /* ...*/
    return false;
  }
}
```

## 6.3 Causal Links

The last component of our operational framework is the *causal links*. A causal link connects two MARTs that will be affected in the case where one of them changes. Causal links are used by the MART management infrastructure to propagate changes.

There are two types of causal links: (1) vertical link, which connects MARTs at the same level of abstraction; and (2) horizontal link, which connects MARTs across different levels of abstraction.

In our approach, our MARTs are connected horizontally, since our Galapagos Metamodel defines the concepts of UCSA, and the Galapagos Model defines the users' tasking and changing situations. More importantly, we only specify causal links in one direction. Only changes in the Galapagos Metamodel effect changes in the Galapagos Model, but not vice versa. This is because the Galapagos Metamodel defines the meaning of the elements in the Galapagos Model.

Figure 6.11 depicts a simplified view of some of the elements that connect to each other in our two MARTs. For example in both MARTs there is an element to represent *personal goals*, which implies that the entity *Personal Goal* and the element *Personal Goal* are connected. Changes in the Galapagos Metamodel ensure that the definition of a personal goal will impact the definition of the personal goal in the Galapagos Model. Similarly with the concept of *task*. Causal links can be from entity-to-element, or from entity-to-attribute. For instance, the entity *Plan Item* is connected to the *number* attribute of the *task*. Other connections, not depicted in the figure, include those between *links* and *connectors*. Since these connections derive form the manipulation of entities, we consider them part of a primary change of an entity.

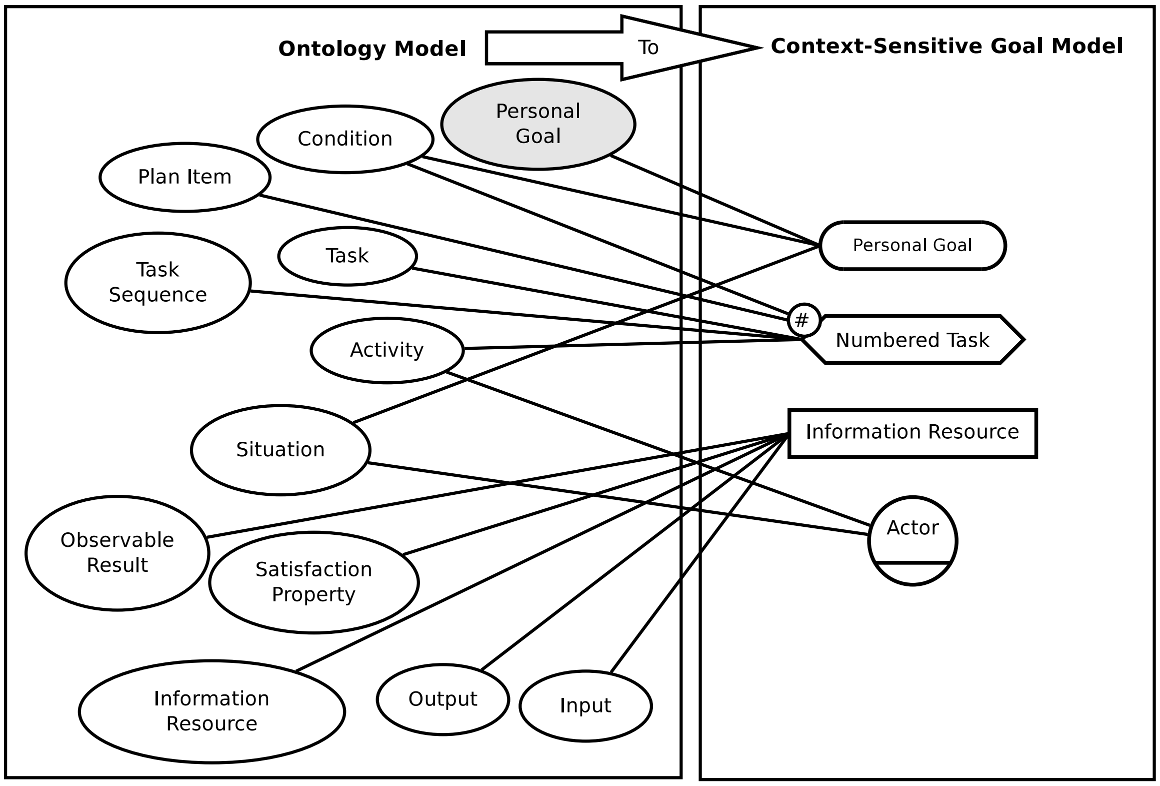


**Figure 6.11:** Simplified view of the causal connections among elements of our two MARTs

Figure 6.12 depicts a simplified view of our Java implementation for causal links. We have three packages, `com.rigiresearch.lcastane.mart.ontology` and `com.rigiresearch.lcastane.mart.goal` contain the definition of the software artefacts of our MARTs and runtime semantics described before in Sections 6.1.3 and 6.2.3, respectively. The third package `com.rigiresearch.lcastane.mart.causallink` contains one main class `CausalLink` that defines the *origin* and *target* MART (ontology and GALAPAGOS MODELS respectively) and the type of link (vertical or horizontal).

There is one public operation, `+registerLink(origin,target,links[])`, which is used to register the changes one element effects over another. The runtime semantics described in the MARTs are used in this method to add, modify or delete elements in the model as a result of a change.

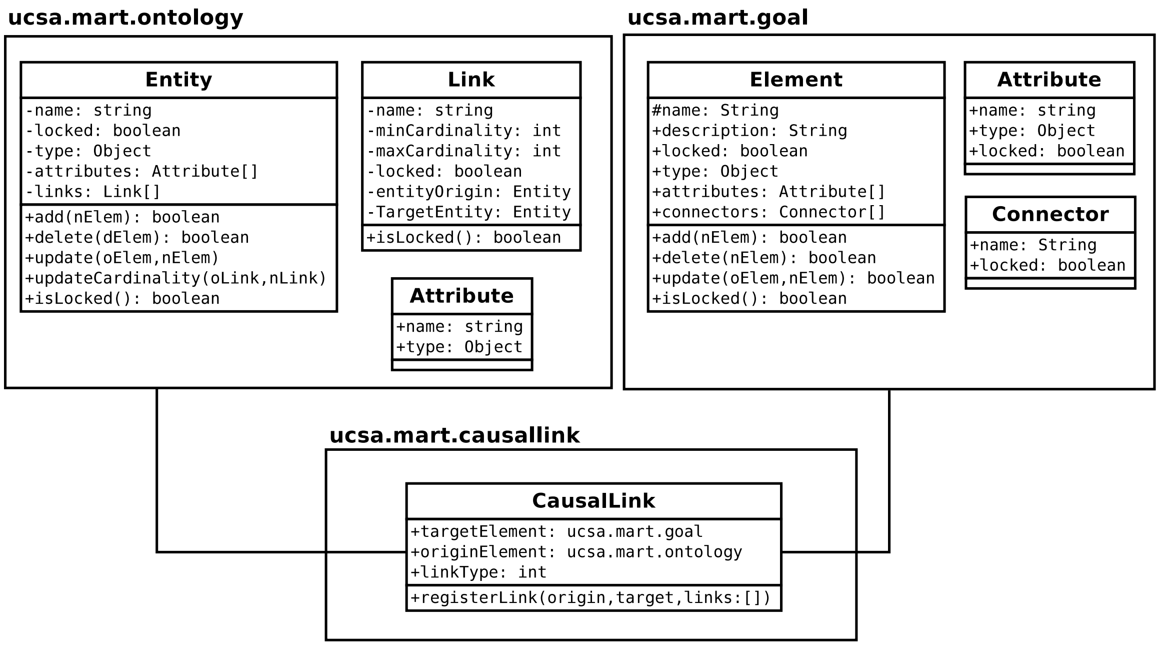


**Figure 6.12:** Java implementation for causal links

## 6.4 Chapter Summary

This chapter presented our operational framework for MARTs used to manage runtime operations and translation between a MARTs notation and its corresponding software artefact form. Our framework constitutes the third contribution of this dissertation, and allows us to answer research question RQ4 (cf. Figure 1.2).

Our operational framework comprises four components: (1) a notation-artefact mapping between the MART human-readable and machine-readable views; (2) a catalogue of operations that describes the considerations and restrictions of every element in the model for runtime CRUD operations; (3) the runtime semantics to execute the operations in the MARTs. In both cases, we use Java to define our runtime semantics. ; and (4)the *causal links* component, which is shared among the MART and describes causal connections among MARTs.

Now that we have defined MART and their operational framework, the next chapter presents our PRIMOR, which is a component of the *MART supporting in-*

*frastructure* that is responsible for orchestrating and managing runtime requests for operations.

# Chapter 7

# Processing Infrastructure for Models at Runtime (PRIMOR)

PRIMOR in Spanish translates to the skill, care and delicacy when doing or saying a thing.

This chapter presents our Processing Infrastructure for Models at Runtime (PRIMOR), which is our fourth contribution of the dissertation. There is one research question associated with this contribution:

RQ4. *What are the runtime infrastructures required to process and evolve runtime models while maintaining the causal relations among them for User-Centric Smart Cyber-Physical-Human Applications?*

To answer this question, we define PRIMOR, which is our infrastructure responsible for managing MARTs' CRUD operations. PRIMOR's main functionalities are: (1) providing reading access from software components to the MARTs; (2) executing model-related runtime operations; and (3) propagating changes among interconnected MARTs and their realities. PRIMOR is a component-based system, implemented in Java, and designed to be extensible to other domains.

In Chapter 4, we presented the components of the UCSA architectural design. The *MARTs supporting infrastructure* component is responsible for hosting and managing MARTs through two components: (1) PRIMOR and (2) the MART repository (cf.

Figure 7.1). The repository contains the models and their operational framework presented in Chapters 5 and 6, respectively.

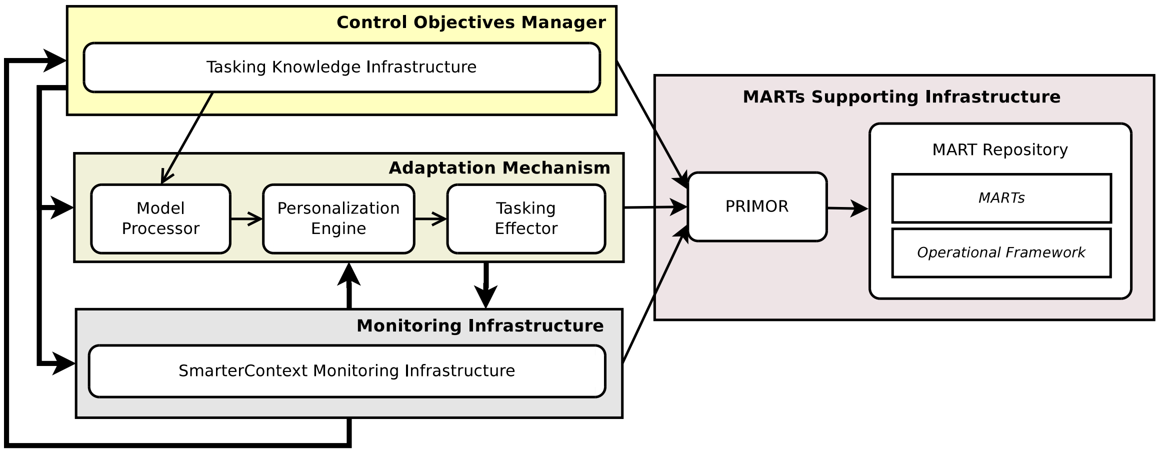

Control Objectives Manager
Tasking Knowledge Infrastructure
Adaptation Mechanism
Model Processor
Personalization Engine
Tasking Effector
Monitoring Infrastructure
SmarterContext Monitoring Infrastructure
MARTs Supporting Infrastructure
PRIMOR
MART Repository
MARTs
Operational Framework

**Figure 7.1:** Architectural design for the implementation of User-Centric Smart Cyber-Physical-Human Applications (UCSAs)

This chapter is organized as follows. Section 7.1 describes the functionalities of PRIMOR. Section 7.2 describes the architecture and Java implementation of PRIMOR. Finally, Section 7.3 summarizes the chapter.

## 7.1 Functionalities

PRIMOR receives two types of requests: *Access Requests* and *Adaptation Requests*, which are handled by two PRIMOR components: the *Access Manager* and the *Operation Manager*, respectively. Figure 7.2 presents an overview of these components and their interactions.

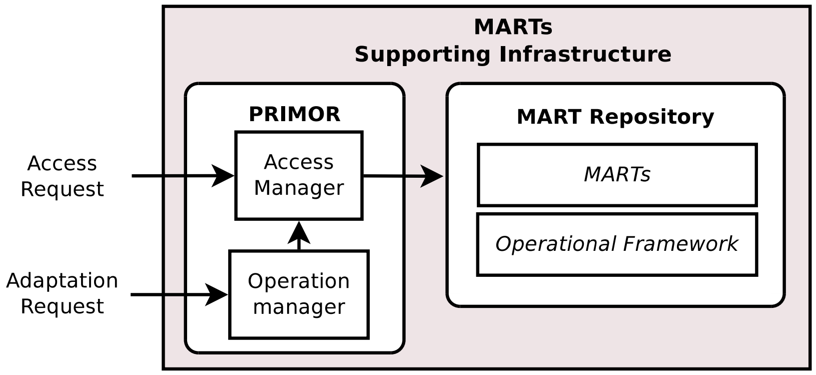


**Figure 7.2:** Overview of PRIMOR based on its responsibilities

### 7.1.1 Access Manager

An *Access Request* seeks reading capabilities from authorized components to specific MARTs. For example, Source Code 7.1 depicts an *Access Request* arriving from an external component (e.g., the *Tasking effector* component in Figure 7.1) requesting an instance of the Galapagos Model. The *Access Request* contains one argument, which is an object type `UCSAGoal`, which is a MART identified with `id=1`. The *Access Manager* component replies to an *Access Request* with a reference to the software artefact of the corresponding MART.

**Source Code 7.1:** Access request example received by Primor

```
request.type.ACCESS(UCSAGoal.id(1))
```

### 7.1.2 Operation Manager

An *Adaptation Request* seeks writing capabilities to execute a CRUD operation on the MARTs. For example, Source Code 7.2 depicts an *Adaptation Request* from the *Personalization Engine*, which contains two arguments: a MART and a *Sentence*. This request is received and handled by the *Operation Manager*.

**Source Code 7.2:** Adaptation request example received by Primor based on the example of Figure 6.8

```
request.type.ADAPTATION(
  UCSAGoalGoal.id(1),    // a reference to the targeted MART
  new Sentence(    // a sentence with action and objects
    Sentence.action.ADD,
    new Task(...),
    new Goal().id(0)
  )
)
```

Handling the *Adaptation Request* consists of four activities: (1) operation identification, (2) causal link analysis, (3) operations planning, execution and assurance, and (4) MART's deployment.

#### Operation Identification

First, the *Operation Manager* analyses the Sentence in the *Adaptation Request* based on the operational framework of the MART. In Source Code 7.2, the MART is an object type `UCSAGoal` identified with `id=1`, which is an instance of the Galapagos Model. From our operational framework, we identify that there are two supported sentence structures for this MART: *action-object-object* and *action-object* (cf. Section 6.2). The *Sentence* in Source Code 7.2 follows the *action-object-object* structure. The action is an *adding* operation, where the first object (element) is a `Task` and the second object (target) is a `Goal` identified with `id=0`.

### Causal Links Analysis

Second, the *Operation Manager* prepares the propagation of changes by analysing the causal links of the MART and creating new sentences for the additional operations. There are two types of change propagation: *internal* and *external*.

- *Internal propagation* occurs when an operation on the MART triggers a set of additional changes in other parts of the same model, either to fit the requested change or as a consequence of the operation. The operational framework specifies internal propagation under preconditions (PRE) and postconditions (POST), which are in the runtime semantics.

- *External propagation* occurs in two modes: (1) *model-to-model*, which implies that changes on a MART affect other MARTs; and (2) *model-to-system*, which implies that changes on a MART affect the structure or behaviour of a system. In both cases, the operational framework specifies the respective connections through causal links.

### Operations Planning, Execution and Assurance

Third, the *Operation Manager* composes an execution plan, coordinates its execution, and controls runtime exceptions by taking corrective or informative actions. These actions are based on policies and might include to roll-back changes on previous MARTs, try alternative operations, continue with the propagation of changes, or notify the external systems about the modification. More importantly, the *Operation Manager* is responsible for executing assurance activities while the MARTs (and their realities) are being modified to guarantee that the overall system remains truthful.

### MARTs' Deployment

Finally, the *Operation Manager* deploys the new versions of the MARTs in the repository, and guarantees both views (notation and software artefact) remain truthful to each other by synchronizing them. That is, software systems and humans have access to the same information of the running system at the same time. To execute deployment and synchronization, the *Operation Manager* uses the mapping of elements specified in the operational framework.

## 7.2 Architectural Design and Implementation

Figure 7.3 depicts the complete architectural design of UCSAs.

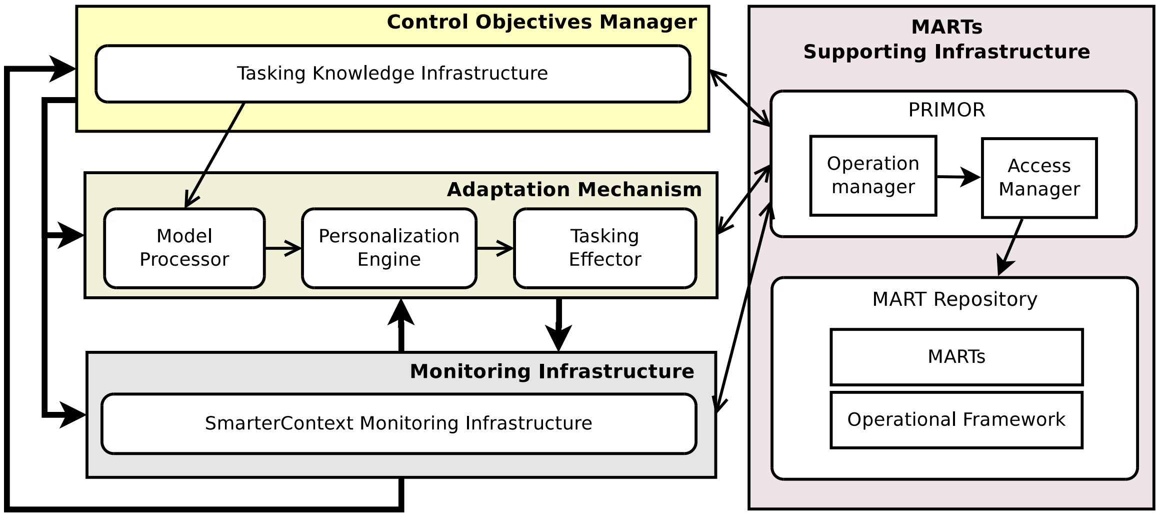

Control Objectives Manager
Tasking Knowledge Infrastructure
Adaptation Mechanism
Model Processor
Personalization Engine
Tasking Effector
Monitoring Infrastructure
SmarterContext Monitoring Infrastructure
MARTs Supporting Infrastructure
PRIMOR
Operation manager
Access Manager
MART Repository
MARTs
Operational Framework

**Figure 7.3:** UCSA Architectural design

In our implementation, we chose Java since it allows our infrastructure to be extensible and implemented by other applications fairly easily. Figure 7.4 provides a closer look into the classes of Primor (`com.rigiresearch.lcastane.primor`) and our implementation of the Galapagos Model as a graph (`com.rigiresearch.lcastane.mart.goalsmodel`) and our operational framework (`com.rigiresearch.lcastane.mart.framework`).

The interface `Manager` exposes the services of Primor to external systems through public methods, and is implemented with the class `ManagerIml`. The method `+registerModel(MART)` allows systems and humans to add a MART into the system's repository. The parameter is a `MART`, which can be either a `Notation` or an

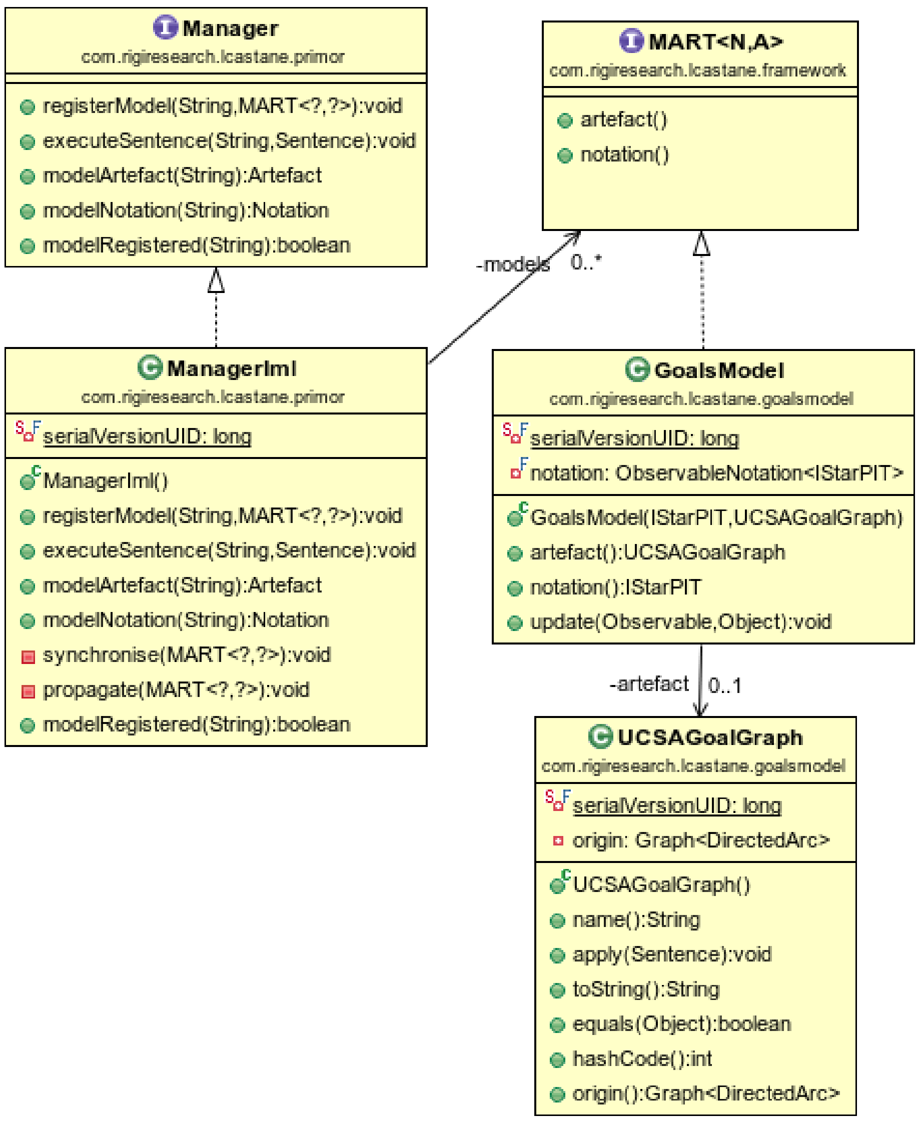

Manager
com.rigiresearch.lcastane.primor
registerModel(String,MART<?,?>):void
executeSentence(String,Sentence):void
modelArtefact(String):Artefact
modelNotation(String):Notation
modelRegistered(String):boolean
MART<N,A>
com.rigiresearch.lcastane.framework
artefact()
notation()
-models
0..*
ManagerIml
com.rigiresearch.lcastane.primor
serialVersionUID: long
ManagerIml()
registerModel(String,MART<?,?>):void
executeSentence(String,Sentence):void
modelArtefact(String):Artefact
modelNotation(String):Notation
synchronise(MART<?,?>):void
propagate(MART<?,?>):void
modelRegistered(String):boolean
GoalsModel
com.rigiresearch.lcastane.goalsmodel
serialVersionUID: long
notation: ObservableNotation<IStarPIT>
GoalsModel(IStarPIT,UCSAGoalGraph)
artefact():UCSAGoalGraph
notation():IStarPIT
update(Observable,Object):void
-artefact
0..1
UCSAGoalGraph
com.rigiresearch.lcastane.goalsmodel
serialVersionUID: long
origin: Graph<DirectedArc>
UCSAGoalGraph()
name():String
apply(Sentence):void
toString():String
equals(Object):boolean
hashCode():int
origin():Graph<DirectedArc>

**Figure 7.4:** Overview of the JAVA classes for the implementation of PRIMOR and our GALAPAGOS MODEL

`Artefact`. For example, a software engineer who specifies a model using a modelling tool, registers the model instance as a *notation*. In the same way, components responsible for analysing systems and inferring models, such as the *Internet Tasking Knowledge Infrastructure*, register the model instance as an *artefact*. Registering a MART includes the operational framework that supports runtime operations in the model.

The method `+executeSentence()` allows systems to request the execution of operations on a MART. As mentioned before, MARTs use sentences implemented with runtime semantics. Primor receives a `Sentence` and applies it to the corresponding MART invoking the `-applyOperation()` method from the `Artefact` class. It is worth mentioning that Primor does not support operations over notation views of the model, since we do not consider notations to be machine-readable.

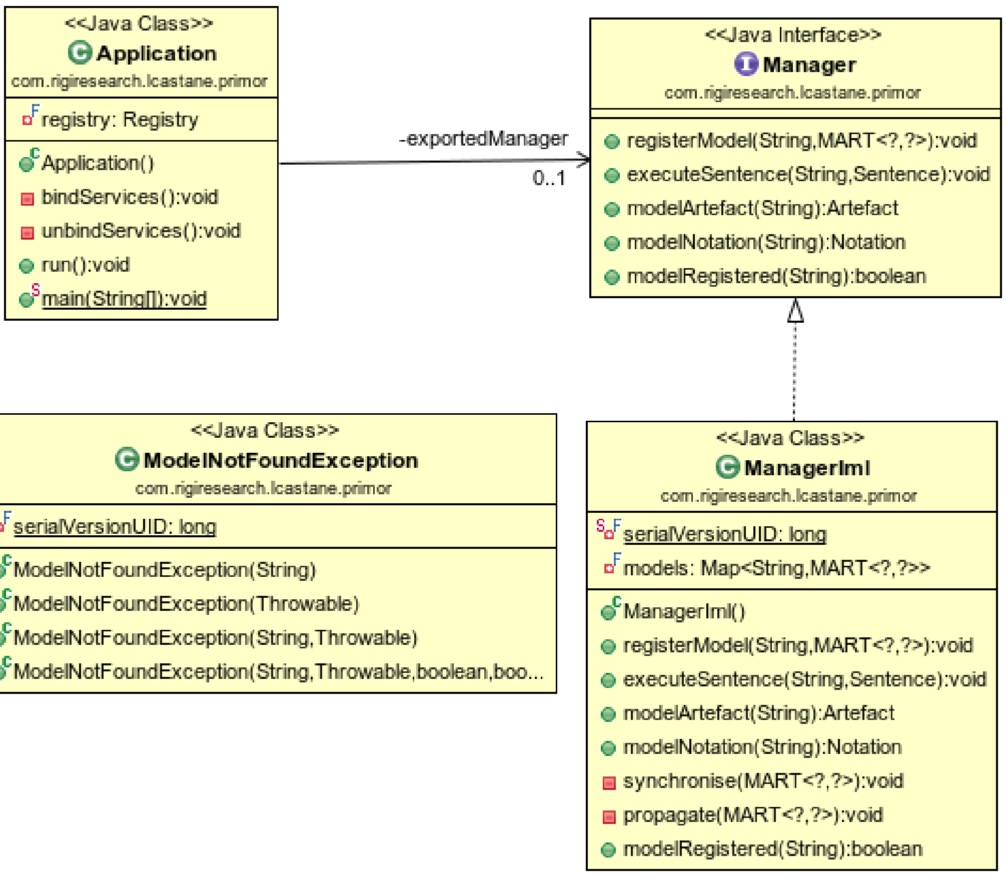
The method `+modelArtefact()` returns the artefact view of a MART. Systems might request the artefact to perform their functional activities. For instance, the *Personalization Engine* and the *Internet Tasking Effector* components of a UCSA need an instance of the Galapagos Model to analyse personalized features and to execute the internet tasking, respectively. In both cases, the components need the graph view, which is the software artefact a system can understand. Similarly, the `+modelNotation()` method returns the notation view of a model. For instance, a software engineer who wants to see the current state of the system might request the notation view and import the model into a modelling tool.

The method `-synchronise(MART)` is a private method, and its responsibility is to perform model synchronisation on the views of a MART. Finally, the method `+propagate()` receives for parameter a `MART`, an `Operation` and a set of `Object` that represent the operands. This private method is used by Primor to manage the propagation of changes based on the causal links of a MART.

In our approach, the Galapagos Model is a *graph* in its artefact form represented by the `UCSAGoalGraph` class. The `UCSAGoalGraph` is composed by nodes, arcs and operations represented by the classes `Node, Arc` and `Operation`, respectively. This class contains the operational behaviour of our Galapagos Model according to our catalogue of operations described in Section 6.2.

Additional classes, not depicted in Figure 7.4, represent elements of the model providing specialised nodes. For example, the `Goal<Node>` class contains attributes that represent measurable outcomes and execution conditions, the `Task<Node>` contains attributes to represent their number in the sequence of tasks, and other control attributes, such as termination and validation flags. The class `Resource<Node>` contains attributes for the location path of the resource, authorization mechanisms, and credentials when required, and the class `Actor<Node>` defines attributes, such as type of actor, virtual identification and location, and a list of permissions of the actor over tasks, resources and goals.

Figure 7.5 depicts another view of the Java classes of our implementation of Primor. The class `ManagerIml` is an instance of the interface `Manager`, which defines the services of Primor presented in the architectural design. In our implementation we created Primor as a service to be deployed in the cloud and that can be called by the UCSA to manage CRUD operations on MARTs. The class `Application` is the executable class of Primor that deploys the service.



<<Java Class>>
Application
com.rigiresearch.lcastane.primor
registry: Registry
Application()
bindServices():void
unbindServices():void
run():void
main(String[]):void
-exportedManager
0..1
<<Java Interface>>
Manager
com.rigiresearch.lcastane.primor
registerModel(String,MART<?,?>):void
executeSentence(String,Sentence):void
modelArtefact(String):Artefact
modelNotation(String):Notation
modelRegistered(String):boolean
<<Java Class>>
ModelNotFoundException
com.rigiresearch.lcastane.primor
serialVersionUID: long
ModelNotFoundException(String)
ModelNotFoundException(Throwable)
ModelNotFoundException(String,Throwable)
ModelNotFoundException(String,Throwable,boolean,boo...
<<Java Class>>
ManagerIml
com.rigiresearch.lcastane.primor
serialVersionUID: long
models: Map<String,MART<?,?>>
ManagerIml()
registerModel(String,MART<?,?>):void
executeSentence(String,Sentence):void
modelArtefact(String):Artefact
modelNotation(String):Notation
synchronise(MART<?,?>):void
propagate(MART<?,?>):void
modelRegistered(String):boolean

**Figure 7.5:** Java classes of Primor

Source Code 7.3 presents a section of the implementation of the class `ManagerIml`. Line 6 shows the attribute `models`, which represents the repository of MART as a `Map` in which the *key* is a `String` and the *value* is a `MART`. Line 15 shows the method `executeSentence()`, which receives the MART identifier and the object `Sentence`. There are two exceptions that can be launched from this method: `ValidationException`, which is propagated from the MART; and `ModelNotFoundException`, which is launched by the method when there is no MART with the identifier sent in the parameter. Lines 19 to 21 apply the `Sentence` if the MART exists, by calling the artefact (i.e., graph) of the MART from the repository (i.e., models `Map`).

**Source Code 7.3:** Implementation of the method `executeSentence`

```java
public final class ManagerIml implements Manager, Serializable {
  /**
  * The model repository.
  */
  private final Map<String, MART<?, ?>> models;

  /*
  * (non-Javadoc)
  * @see com.rigiresearch.lcastane.primor.Manager
  *  #executeSentence(java.lang.String,
  *  com.rigiresearch.lcastane.framework.Sentence)
  */
  @Override
  public void executeSentence(final String identifier,
  final Sentence sentence)
    throws ValidationException, ModelNotFoundException {
      if (this.models.containsKey(identifier)) {
        this.models.get(identifier)
        .artefact()
        .apply(sentence);
      } else {
      throw new ModelNotFoundException(
      String.format(
      "Model %s not found",
      identifier
      )
      );
    }
  }
}
```

Figure 7.6 depicts some classes of the implementation of our operational framework, our MART and PRIMOR.

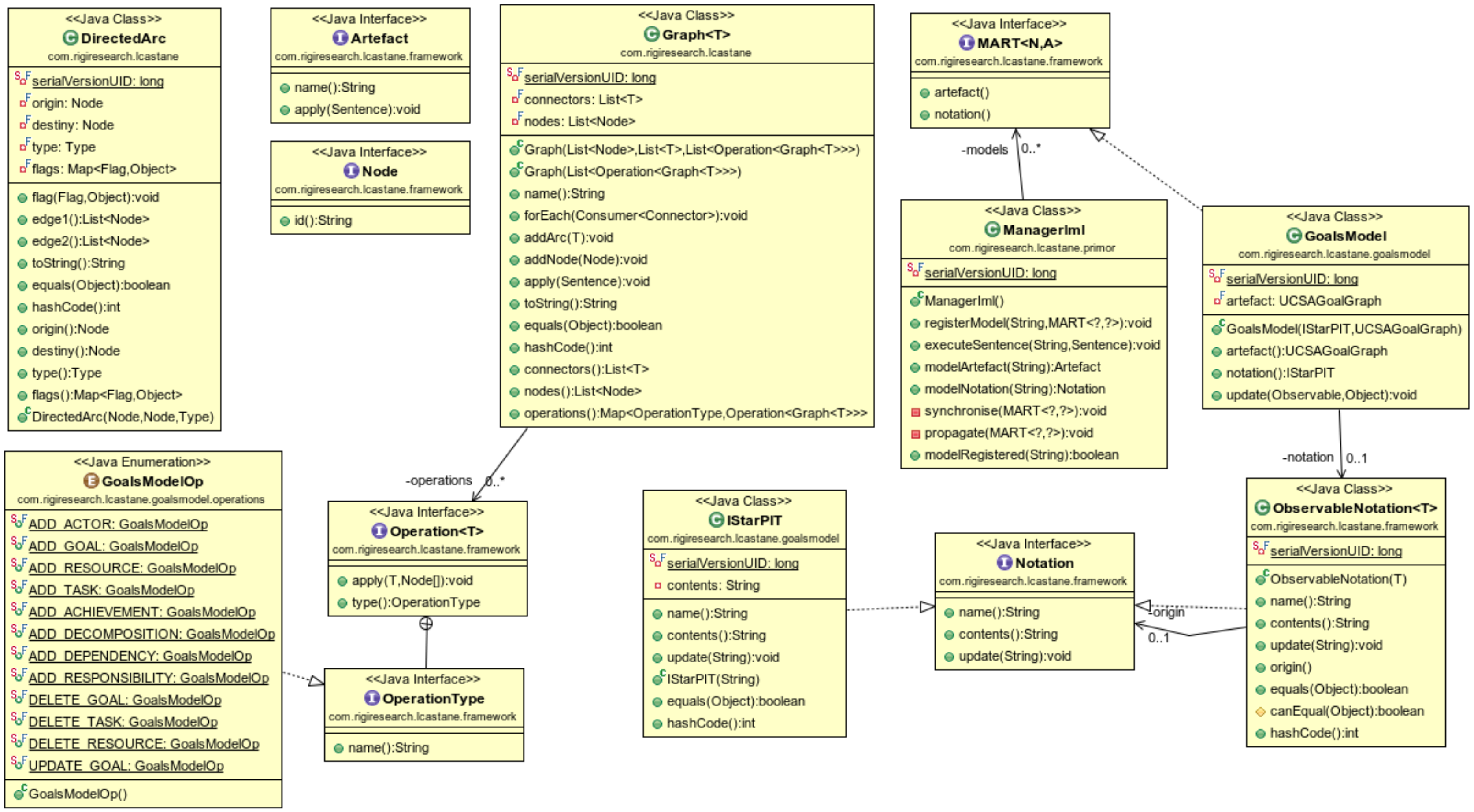


**Figure 7.6:** Java classes of our Operational Framework

## 7.3 Chapter Summary

This chapter presented the Processing Infrastructure for Models at Runtime (PRIMOR), which is our infrastructure to manage runtime operations on Model at Runtime (MART). PRIMOR constitutes the fourth and last contribution of this dissertation and allows us to answer research question RQ4 (cf. Figure 1.2).

PRIMOR manages runtime operations for our two MARTs (cf. Chapter 5): our GALAPAGOS METAMODEL and our GALAPAGOS MODEL. PRIMOR's main functionalities are: (1) providing reading access from software components to the MARTs, (2) executing model-related runtime operations, and (3) propagating changes among interconnected MARTs and their realities.

PRIMOR receives two types of requests: an *Access Request* and an *Adaptation Request*. An *Access Request* asks for reading capabilities on MARTs, which is handled by the *Access Manager* component. An *Adaptation Request* asks the execution of changes on the MARTs, which is handled by the *Operation Manager* component. Handling the *Adaptation Request* consists in four activities: (1) operation identification, (2) causal link analysis, (3) operations plan, execution and assurance, and (4) MART's deployment. PRIMOR is a component-based system, implemented in Java, and designed to the extensible to other domains.

The next chapter of this dissertation presents the evaluation of our contributions in two parts: First, a qualitative evaluation of our modelling approaches compared with related approaches; and second, a quantitative evaluation based on controlled experiments.

# Chapter 8

# Evaluation

This chapter presents the evaluation of the contributions of this dissertation. Section 8.1 presents the qualitative analysis for our second contribution, our GALAPAGOS METAMODEL and our GALAPAGOS MODEL. Through this qualitative analysis, we demonstrate the novelty of our approach by comparing it with related approaches.

Section 8.2 presents the experimental analysis of our four contributions: our definition and an architectural design for User-Centric Smart Cyber-Physical-Human Applications (UCSAs); our two MARTs as software artefacts; our operational framework; and our Processing Infrastructure for Models at Runtime (PRIMOR). For our experimental analysis, we evaluated our design and implementation in terms of three quality attributes: *accuracy, scalability*, and *performance* [Gor11]. The experimental results of this analysis demonstrate the applicability of our MARTs and PRIMOR. Finally, Section 8.3 summarizes the chapter.

## 8.1 Qualitative Evaluation

### 8.1.1 Related Approaches Comparison

Chapters 5 and 3, described the modelling requirements for the implementation of user-centric web-tasking cyber-physical-human systems, and reviewed approaches in the scope of modelling some of the concerns associated with our requirements respectively. This section presents a comparison of our approach with related works based on a qualitative assessment of our modelling requirements. For each approach, we characterize the way authors represent our requirements and discuss how these align with our approach.

### Personal Goals and Task Interactions

Table 8.1 summarizes approaches related to two of our core concepts: *personal goals* and *task interactions*. Users achieve personal goals through the execution of task interactions, that are ordered in a logical sequence. Our study of related approaches provides evidence that goal models are often used to represent user's intentions and graphs are useful to represent multiple decisions or variability on choices. Moreover, interactions are represented in the form of a sequence of decisions using ordered activities.

Our approach differentiates from others in the way we represent the achievement of a goal. As mentioned in Chapter 5, we focus on the results of the tasks and consider outputs as the *measurable outcomes* of the goal.

In our MARTs, the concept of a personal goal contains more than the sequence of the tasks, by including the specification on how to assess the execution and achievement of the goal. The assessment is made through the specification of *observable results* and *satisfaction measurements and thresholds*, which defines the achievement of a goal based on the output of the tasks. Additionally, our specification includes regulation information, such as *satisfaction* and *execution rules*, which allow infrastructures to manage and control the execution of a task sequence.

**Table 8.1:** Mapping between related approaches and two of our core concepts: *personal goals* and *task interactions*

| **Approach** | **Core Concepts** | |
|---|---|---|
| | **Personal goals** | **Task interactions** |
| Bolchini [BM03] | Uses the user's goal and the stakeholders preferences to analyse the task | |
| Carberry [Car88] | Structure of conditions, body and effects. | |
| Carberry [Car88] | A node is a goal inferred by the system | The active path corresponds to the user's decisions |
| Chopra [CDGM10] | Uses goals (TROPOS methodology) to specify a user | |
| Goschnick [GBS08] | Hierarchical task model for goals and subtasks | |

| Approach | Core Concepts | |
|---|---|---|
| | **Personal goals** | **Task interactions** |
| Liaskos [LMSM10, LLJM11] | AND/OR decompositions to represent alternatives | |
| Liaskos [LLJM11] | Goal (ovals), Hexagons (task), arcs (relationship) | A numbered sequence of tasks to create a path of interactions |
| Mylopoulos [MCY99] | Goals and subgoals (tasks) | |
| Wilson [Wil99] | The goal is an information seeking achievement | |
| Yu [Yu93] | Goal dependencies AND satisfaction | |
| Yu [YM94] | Goal Dependency strength and type | |

### Tasks

Concerning the modelling requirements for tasks, we grouped related approaches into two groups: *task definition*, and *task execution*. First, Table 8.2 summarizes approaches related to the definition of a task. In this table, approaches are compared to some of our modelling requirements for personal tasks: *inputs, outputs, activities*, and *resources*. Tasks and goals are strongly connected since tasks are the activities that the user performs to fulfil a goal. Most of the studied approaches focus on representing the *activities* of a task. Second, Table 8.3 summarizes approaches related to the execution of a task. In this table, approaches are compared to some of our modelling requirements: *task sequence* and *execution controller*. Some approaches implement hierarchical models (e.g., graphs and workflows) to provide an ordered structure of task sequences.

Our approach differentiates from others in the way we specify explicit information about the execution of a task. Our MARTs represent *preconditions*, which are implemented as validations prior the execution of the task. Moreover, our models support the specification of *pre-* and *post-activities* regarding the sequencing of a particular task with others and using logical operators to create alternative or parallel executions. Finally, our MARTs specify *policies* to regulate the task execution and to guide runtime infrastructures with decision making processes.

**Table 8.2:** Mapping between related approaches and some of our task definition modelling requirements: *inputs, outputs, activities* and *resources.*

| **Approach** | **Task definition modelling requirements** | | |
|---|---|---|---|
| | **Inputs/Outputs** | **Activities** | **Resources** |
| Bolchini [BM03] | | Task analysis, identification and decomposition | |
| Brezillon [Bré07] | | Task model to describe the actions in a degree of generality | |
| Cui [CLWG04] | Inputs and outputs as parameters | | A web service is a composition of input and output parameters |
| Giersich [GFF$^+$07] | | Task tree to represent the activities | |
| Klug [KK05] | The information exchange is specified in the task model (exchanged through ports) | | |
| Souchon [SLV02] | | Nodes of the graph are tasks (subtasks) activities. Edges are the relationship | |
| Stoitsev [SS08] | | Types of tasks (activities): strategical, tactical and operational TO interaction features | |

| Approach | Task modelling requirements | | |
|---|---|---|---|
| | Inputs/Outputs | Activities | Resources |
| Yu [Yu93] | | Task dependencies | Resource Dependencies |
| Yu [YM94] | | Task Dependency (strength and type) | Resource Dependency (strength and type) |

**Table 8.3:** Mapping between related approaches and some our task execution modelling requirements: *sequence* and *execution controller*

| Approach | Task execution modelling requirements | |
|---|---|---|
| | Task sequence | Execution controller |
| Bolchini [BM03] | Hierarchical task decomposition activities | |
| Brezillon [Bré07] | The graph connects the tasks | |
| Dix [Dix08] | Hierarchical Task Analysis (HTA), to create sequences of tasks in the form o a Plan | |
| Giersich [GFF+07] | Hierarchical decomposition of an activity into individual steps of a task | |
| John [JVM+02] | Model human behaviour in complex dynamic tasks | |
| Klug [KK05] | | Includes mechanisms to support correctness and completeness while executing a sequence of tasks |
| Liaskos [LMSM10, LLJM11] | The order of the execution tasks are relevant | |
| Mylopoulos [MCY99] | AND, OR operators to connect the subgoals | Conflict analysis to control potential collisions |

| Approach | Task execution requirements | |
|---|---|---|
| | Task sequence | Execution controller |
| Souchon [SLV02] | A direct graph that connects a set of task (subtasks) with a root and a set of transitions | |
| Thom [TLI+11] | BPMN to design the execution of activities | |
| Wilson [Wil99] | | The model describes the various elements that take place while seeking information |
| Yu [YLL+08] | Alternate sequences of the solution | |

## Context

Concerning the modelling requirements for context, we focused on related approaches that specifically represent *internal, external*, or *personal context.* More importantly, we focused on representations of context that consider some of its dynamic nature since our applications exist in a highly-dynamic socio-technological environment. Table 8.4 summarizes approaches related to context modelling.

Villegas' personal context sphere provides a repository to store, share, and infer dynamic information about users, their preferences and interactions [Vil13]. For our approach, we use Villegas' smarter context ontology to represent context by extending her ontology for the grocery shopping domain.

Our approach differentiates from others in the way our MARTs represent *situations* explicitly as pieces of information that connect the user with the achievement of a personal goal. Situations are the result of analysing personal context information, as it affects a goal the user intends to achieve.

**Table 8.4:** Mapping between Context modelling requirements and modelling approaches

| Approach | Context modelling requirements | | |
|---|---|---|---|
| | Internal | External | Personal context |

| | | | |
|---|---|---|---|
| Dix [Dix08] | | "Data-detectors" in the form of textual analysis to detect key terms | Personal Ontology to model static context information |
| Souchon [SLV02] | Context considers: user, platform and environment | | |
| Villegas [Vil13] | Context Entities represent contextual information in the form of RDF graphs, which support dynamic behaviour. | | The Personal Context Sphere (PCS) is a repository of the user's information and preferences. The PCS is controlled by the user. |
| Yu [YM94] | | | The user is an actor, which comprises a role, position, agent, and associations |

In this section, we assessed our models according to related approaches in order to highlight how our approach differs. More importantly, given the dynamic nature of our application domain, we add the value of our models through their runtime capabilities. In Chapter 2, we introduced MARTs as having four major characteristics: *representation, availability, causal connection*, and *evolution*.

Throughout this dissertation, we have demonstrated that our models satisfy MARTs characteristics, therefore making them proper models at runtime. We have presented that our models: (1) satisfy the representation of users' task interactions that lead to the achievement of a particular personal goal; (2) are available to software systems; (3) maintain a connection with their reality (i.e., users' personal tasking and goals); and (4) are capable to evolve at execution time.

## 8.2 Experimental Evaluation

To demonstrate the feasibility of our contributions, we instantiated our MARTs, and implemented Primor for our online grocery shopping scenario. The details of our implementation are described in Appendix B. For the experimental analysis, we ran a set of experiments in terms of three quality attributes [Gor11]:

- *Accuracy* of the MART: assessed as the capability of our models to support the representation of relevant changes in the context.
- *Scalability* of the infrastructure: assessed as the capability of our framework to support growing models and new operations.
- *Performance* of the infrastructure: assessed as the time it takes to our infrastructure to perform CRUD operations on MARTs.

The following section introduces Edel, our UCSA's user for the scenario of online grocery shopping (cf. Section 4.2) in our experimental evaluation.

### 8.2.1 Case Scenario: Supporting Independent Living for the Elderly

Edel is an 85-year-old woman living in the city of Victoria. She lives by herself, and it is her wish and those in her family, to maintain her current living situation as long as possible. However, Edel's age comes with some cognitive challenges, such as memory loss, reduced processing speed, and decreased motivation for doing ordinary tasks. Maintaining her independent living is highly essential for Edel and all her family, mainly because of a present genetic heritage of Alzheimer's disease.

One of the activities that satisfy Edel's independence is her grocery shopping. Although Edel enjoys her trips to the grocery store, she has encountered some frustrating situations. A few times, her grocery shopping has been more than she could carry. Since she doesn't drive, her preferred stores are within walking distance from her home, thus complicating her way back. One time she had to take a taxi, which put her out of her budget. Another time she made several trips from her home to the store, which made her extremely tired with joint and muscle pain. Many times Edel misses on some good grocery deals because she has to choose one grocery store for her shopping. Making trips to different stores or in different days is not a possibility for Edel. A recurrent frustration occurs when she comes home to find out she forgot to buy something, or that an element in her kitchen needed replacement because it expired. In some cases, the option of waiting for the next grocery shopping is not possible, and Edel ends up making another trip to the grocery store. Finally, every time she has to get cleaning-related products, which are often bulky or heavy, she relies on her son to drive her to the grocery store. One time he was out of town for three weeks, thus affecting her grocery shopping activity.

One way to help Edel to reduce her frustrations would be to hire someone to do the grocery shopping for her, but that would have a negative impact on her independent living. To improve Edel's quality of life by supporting her independent living, she uses SUSGroceries, our UCSA to assist her in the achievement of her grocery shopping goal. SUSGroceries for Edel is possible since, despite her age, Edel is not foreign to technology. Edel's son and granddaughter had instrumented her home with sensors and actuators to collect various kinds of information, some connected to SUSGroceries. Edel's primary interface with SUSGroceries is her smartphone, which she always carries with her. Through her smartphone, she provides all sort of personal context information including preferences and location.

SUSGroceries collects information about Edel. For instance, her instrumented kitchen—smart camera on her fridge and intelligent mats on her pantry—is used to populate her grocery shopping list; her calendar and social connections are used to prepare the grocery shopping for food-related events; her location is used to track her in her preferred stores and to collect store-related information, such as deals and discounts. More importantly, SUSGroceries understands that Edel likes to visit the grocery store to get some of her groceries herself. Therefore, SUSGroceries assists her grocery shopping by suggesting her items she can get in the store to carry home comfortably, while effecting online grocery shopping for the remaining items to complement Edel's achievement of her personal goal.

### 8.2.2 Accuracy Analysis

To evaluate *accuracy*, our experiments involve having changes in the personal context as these reflect situations that might affect the execution of the tasking. The objective of this analysis is to assess the capability of our Galapagos Model of representing at runtime, the user's tasking and situations, as these change during the execution of the system. We focus on two variables of the user: *location* and *preferences*, which are representative pieces of context information that affect the achievement of personal goals.

Since *location* and *preferences* are general concepts of context, we focus on sub types as follows:

- **Location:** We experimented with three types of *location* that might affect the achievement of the online grocery shopping goal. In this case, new locations were (1) within the original location's radius; (2) outside the original location's radius; and (3) with no grocery stores within its new radius.

- **Preferences:** We experimented with three categories of preferences: (1) stores, (2) payment method, and (3) types of food.

  It is worth noting, that our representation of preferences follow the SmarterContext personal context sphere (PCS) structure by Villegas [Vil13]. Table B.2 presents the PCS for the user Edel in this case scenario.

### Analysis Process

For this analysis, we assumed that all changes in the *user's location* are permanent; and the user has authorized the system to effect changes in the model when the option becomes available. Moreover, we assumed that all changes in preferences are dynamic and can occur while the system is executing, and that all the preferences will modify the corresponding MART instance. We use functions in SUSGroceries to simulate external sensors changing the user's preferences.

Our analysis was conducted following four steps:

1. **Identification of model elements:** First, we identified the elements in the models that are affected by the context of *location* and *preference*. According to our Galapagos Metamodel, location—represented by the context entity `gc:LocationContext` from the SmarterContext ontology—is associated with the concepts of `Situation` and `InformationResource`. Consequently, our Galapagos Model represents *location* in the concepts of `Actor` and `InformationResource` (cf. Appendix A). For the *preference* context information, we focused on the three categories we wanted to examine. We identified that *stores* are the concept of `Actor` and `InformationResource`; *form of purchase* affect the concepts of `Task` as well as the *execution policies*; and *types of food* affect the concept of `Task`.

2. **Identification of policies** Second, we identified the policies that describe *location* and *preferences* as relevant pieces of context. According to our Galapagos Metamodel, policies are a type of `Condition` associated with the concepts of `TaskSequence`, `Task/Subtask` and `Situation`. Consequently, our Galapagos

Model represents *policies* as attributes of `Goal`, `Task`, and `Actor`. by the simulated changes in the context of the user.

3. **Experimental plan:** Third, we planned the scenarios in which the *location* and *preferences* variables would change, and predicted the resulting model instance based on the information available to our system SUSGroceries. Since our infrastructure does not support inference mechanisms, we defined alternative paths, and existing variability points based on our scenarios. To recreate the runtime environment of the system, we used time delays in the software component responsible to execute the personal tasking.

4. **Observation of the model instance:** Finally, after running the experiments, we observed the console output of the experiments to follow the execution of the CRUD operations, and the resulting model instance. As a result, we compared the predicted model instances after changes with the model instance that should be affected by changes in the aforementioned context variables, and predicted the model instance outcome.

### Tests and Results

In this section, we present three cases where context changes impact the user's situations that are relevant for the achievement of her personal goal. We present our results using our reporting pattern, which comprises the following sections:

- *Description*: Provides a synopsis of what the test is about.
- *Context change*: Describes the piece of context information that will have the change in this test.
- *Context source*: Describes the source that contains the pieces of context information and reports the context change.
- *Type of experiment* : Defines whether the test is a *situation-awareness* test, or a *CRUD operation* test.
- *Related policies*: Defines which policies in the objectives manager component are affected by this context change.
- *Effect*: Presents the designed functions to manage the change context event

- *Expected results*: Describes the expected results
- *Test results*: Presents the console output from the applications and services involved: SUSGroceries, Primor, and `JUnit`. The lines in the console include the following elements: (1) the action occurring in the system (i.e., *GET* message, context or model *CHANGE*, *SEND* message, display *INFO*); (2) the data information, such as the context changed, the message received or sent, or the change that took effect;

Our reporting pattern follows the essential elements of a pattern described by Gamma et al: (1) a *name* to describe the problem, (2) the *problem* and its context, (3) the *solution* in terms of the essential elements, and . (4) the *results* and trade-offs [GHJV95]. More importantly, our reporting pattern provides a mechanism to easily plan and compare other tests and evaluations of our infrastructure.

**Case 1. Location:** We modified a variable in SUSGroceries to determine a new set of coordinates for the preferred location of the user. The scenario is described as following:

> Edel, our online grocery shopper, is interested in grocery stores within a radius of 10 km around her preferred location. As a consequence there is a *task* associated with her personal goal of grocery shopping, to select grocery stores that offer online shopping within the specified radius. To be closer to her son, Edel changed her residency to a building in the same neighbourhood, which is 15 km from her previous home. A change in Edel's location should reflect on the change of the selection of available stores within her specified radius.

Table 8.5 summarizes the results of this test. As an expected result, our MART was able to support the representation of *location* as a piece of context information and the modification caused by the user's change in location. Through policies, changes were propagated in the model to other parts of the MART instance, such as the information about the location of the `Actor` that represents the user; the addition of new elements `Resource` to represent new grocery stores within the new radius; and the deletion of one element `Resource` that is no longer in compliance with the execution policies since it is outside the radius.

**Table 8.5:** Accuracy Case 1: Situation-awareness based on a change of the user's location

| | |
|---|---|
| Description: | A change in the location of the user creates a change in the user's situation |
| Context Change: | Location (`nLocat`) |
| Context Source: | Sensor (SUSGroceries) |
| Type of experiment : | Situation-awareness |
| Related Policies: | `sameAs(location, Actor(User), Actor(Store))` |
| Effect: | `flag(Node.class, Actor(Store), Context(location))`<br>`policy(Context(location), discardTasking)`<br>`policy(Context(location), executeTasking)`<br>`change(User.location,nLocat)`<br>`change(User.preferences.Store,(nLocat, settings.radius))`<br>`verify(Goal.executionPolicies)` |
| Expected Result: | 1. Change in the user's location setting<br>2. Change in the user's preferred stores adding others nearby on a radius of 10 km around the new location<br>3. Include the new stores in the tasking |

| Test Results: | |
|---|---|
| [SUSGroceries] | `> [Change] [Edel:User:Actor] :  Location[lat,long] --> [48.453844, -123.401481]`<br>`> [Change] [Edel:User:Actor] : Preferences.Store[lat,long,radius(mt)] --> [48.453844, -123.401481, 1000]`<br>`> [Send] [Primor.Request.Access] :  ucsa.goals.id(1)`<br>`> [Action] Executing tasking.  Personal goal --> [ucsa.goals.mainGoal["G_grocery_shopping"]]`<br>`> [Send] [Primor.Request.Adaptation] : ucsa.goals.id(1), Sentence[ADD, Resource("R_store5"), Task("T_match_items")]`<br>`> [Send] [Primor.Request.Adaptation] : ucsa.goals.id(1), Sentence[ADD, Resource("R_store6"), Task("T_match_items")]` |

| | |
|---|---|
| | `> [Send] [Primor.Request.Adaptation] : ucsa.goals.id(1), Sentence[DELETE, Resource("R_store3")]` |
| [PRIMOR] | `> [Get] [Request.Access.SUSGroceries] : [RETURN] ucsa.goals.id(1)`<br>`> [Get] [Request.Adaptation.SUSGroceries] : [Execute] ucsa.goals.id(1), Sentence[ADD, Resource("R_store5"), Task("T_match_items")]`<br>`> [Get] [Request.Adaptation.SUSGroceries] : [Execute] ucsa.goals.id(1), Sentence[ADD, Resource("R_store6"), Task("T_match_items")]`<br>`> [Get] [Request.Adaptation.SUSGroceries] : [Execute] ucsa.goals.id(1), Sentence[DELETE, Resource("R_store3")]`<br>`> [Info] ucsa.goals.id(1) CHANGED : [ADD] 2 Nodes, 2 Arcs - [DELETE] 1 Nodes, 2 Arcs - [UPDATE] 0 Nodes, 0 Arcs` |

**Case 2: Preference** We modified a variable in the user's personal context sphere replacing the entry for *Homogenized Milk* (cf. Line 6 in Table B.2) with a new one for *Soy Milk* instead. SUSGroceries employs user's preferences to determine the personalization of the user's tasking, in this case affecting those tasks related to a change in food type. The scenario is described as follows:

> During Edel's recent visit to the doctor, her physician determined she should remove lactose-based elements from her diet and replace them with a base of almond or soy instead. As a consequence, her health-care information, which is connected to her her personal context sphere by a third-party system, gets updated with this new dietary information. SUSGroceries realizes there is a change in the user's personal context. Moreover, such a change is relevant to the fulfilment of the user's goal of grocery shopping. As a result, the change on Edel's food preference implies reviewing her grocery shopping list and an addition of a new task to find lactose alternatives.

Table 8.6 summarizes the results of this test. As an expected result, the tasking of the user requires the addition of a new `Task` to filter the grocery items and discard those no longer permitted. During our test the *preference* change raised the affected model entities and policies to prepare for the change. Based on the policies the system acted accordingly—adding a new `Task` to the MART instance to accommodate the new situation of the user. Although another solution would have been to update the grocery list through external services and not though our MART—our approach supports adapting the user's tasking by exploiting knowledge autonomously. As mentioned before, so far our infrastructure does not support inference mechanisms. Therefore, the `Tasks "T_filterItems"` and `"T_findLactoseAlter"` for our scenario originally belonged to other instances of internet tasking that we stored in a knowledge repository. Our MART is capable of introducing shared tasks in order to compose new sequences.

**Table 8.6:** Accuracy Case 2: Situation-awareness based on a healthcare event related to the user's dietary restrictions

| | |
|---|---|
| Description: | An event in the user's healthcare system creates a change in the user's situation related to her eating restrictions |
| Context Change: | Food flag (foodType). |
| Context Source: | Healthcare System Sensor (Simulation) |
| Type of experiment : | Situation-awareness |
| Related Policies: | `relatedContext(flag.food, Goal("G_grocery_shopping"))` |
| Effect: | `flag(Goal.class, flag.food.type, Context(foodType))`<br>`policy(Context(foodType), updateTasking)`<br>`change(Goal.sequence, nTask)`<br>`verify(Goal.executionPolicies)` |
| Expected Result: | 1. Add an execution policy to discard grocery items in compliance with the user dietary restrictions<br>2. Add a new task |
| Test Results: | |
| [SUSGroceries] | `> [Send] [Primor.Request.Access] :  ucsa.goals.id(1)`<br>`> [Send] [Primor.Request.Adaptation] :`<br>`ucsa.goals.id(1), Sentence[ADD, Task("T_filterItems"),`<br>`Task("T_getGroceryList")]` |

| | |
|---|---|
| | > [Send] [Primor.Request.Adaptation] : ucsa.goals.id(1), Sentence[ADD, Task("T_findLactoseAlter"), Task("T_filterItems")] |
| [PRIMOR] | > [Get] [Request.Access.SUSGroceries] : [RETURN] ucsa.goals.id(1)<br>> [Get] [Request.Adaptation.SUSGroceries] : [Execute] ucsa.goals.id(1), Sentence[ADD, Task("T_filterItems"), Task("T_getGroceryList")]<br>> [Get] [Request.Adaptation.SUSGroceries] : [Execute] ucsa.goals.id(1), Sentence[ADD, Task("T_findLactoseAlter"), Task("T_filterItems")]<br>> [Info] ucsa.goals.id(1) CHANGED : [ADD] 2 Nodes, 2 Arcs - [DELETE] 0 Nodes, 0 Arcs - [UPDATE] 0 Nodes, 0 Arcs |

**Case 3: Preference** Our third case focused on a combination of two *preferences*: payment method and specific grocery stores. The goal of this test is to analyse the capability of our model to represent the user's tasking when there are two changes. Moreover, we want to analyse the capability of our models to represent the information necessary to resolve conflicts driven by changes that contradict each other. Here is our example scenario for Case 3:

> Edel defined three grocery stores and one payment method in her personal context sphere. She wants to add a fourth grocery store and update her credit card information to set a new payment method for her groceries. This change presents two situations that invalidate the achievement of Edel's grocery shopping goal. First, the new grocery store is not within the specified 10 km radius of her location, and second the new credit card information is not accepted by the three current stores.

Table 8.7 summarizes the results of this test. As an expected result, the model instance was able to represent information to support conflict resolution through the concept of *policies*. According to our Galapagos Model , *policies* are present in the concepts of `Goal` and `Task`. During our experiment, policies are described with a weight, which is a number that represents the priority level when an execution conflict arises. Policies also contain *flags* which represent alternative executions when the *Task* can't be executed. Although flags are customizable in the model instance, the two shown in the test case are: `Human in the loop (HIL)`, which implies requesting for human intervention to solve the conflict; `Replacement (REP)`, which implies requesting the inference engine to find an alternative `Task` as a replacement.

**Table 8.7:** Accuracy Case 3: Conflict resolution based on two context changes that invalidate the achievement of the personal goal

| | |
|---|---|
| Description: | The user registers a new preferred grocery store and a new form of payment. |
| Context Change: | Store flag (nStore). Payment flag (nCreditCard). |
| Context Source: | Personal Context Sphere (Local XML File) |
| Type of experiment : | Conflict resolution |
| Related Policies: | `relatedContext(flag.Store, Task("Search_Stores"))`<br>`relatedContext(flag.Payment, Task("Search_Stores"))`<br>`relatedContext(flag.Payment, Task("Proceed_Checkout"))`<br>`sameAs(location, Actor(User), Actor(Store))`<br>`sameAs(payment.type, Actor(Store),`<br>`Resource(nCreditCard.type)`<br>`policy(Goal.execution.priority,`<br>`Goal("G_grocery_shopping")) policy(Task.execution,`<br>`Task("Search_Stores")) policy(Task.execution,`<br>`Task("Proceed_Checkout"))` |
| Effect: | `execute(Task.class, searchStores) ---> Conflict (1)`<br>`execute(Task.class, proceedCheckout) ---> Conflict (3)`<br>`execute(Goal.class, validateTaskSequence) ---> Invalid` |
| Expected Result: | 1. The task sequence validation must show it is not possible to fulfil the goal<br>2. Display available information required to solve the conflict |

| Test Results: | |
|---|---|
| [SUSGroceries] | `> [Change] [Edel:User:Actor] :`<br>`Preferences.Resource.Payment --> nCreditCard` |

| | |
|---|---|
| | > [Change] [Edel:User:Actor] : Preferences.Store --> nStore > [Send] [Primor.Request.Access] : ucsa.goals.id(1) > [Send] [Primor.Request.Assessment] : ucsa.goals.id(1), execute(Validation.Type.Policy, ucsa.goals.mainGoal["G_grocery_shopping"]) > [Get] [Primor.Error] : Message --> ‘‘The goal cannot be achieved. Sequence execution incomplete (Thrown by Task.proceedCheckout(paymentInvalidException)). Possible Sequences (1). Conflicts reported (4).’’ |
| [PRIMOR] | > [Get] [Request.Access.SUSGroceries] : [RETURN] ucsa.goals.id(1) > [Get] [Request.Assessment.SUSGroceries] : [Execute] ucsa.goals.id(1), Validate(Validation.Type.Policy, Goal("G_grocery_shopping")) > [Conflict] [policy] : ucsa.resources.store(1).payment !=== user.payment.2 --> Flag.REP > [Conflict] [policy] : ucsa.resources.store(2).payment !=== user.payment.2 --> Flag.REP > [Conflict] [policy] : ucsa.resources.store(3).payment !=== user.payment.2 --> Flag.REP > [Info] [Relevant context ignored by policy] : >> User.Preferences.Store(4) by policy( sameAs (location,Store,User)) > [Conflict] [sameAs] : ucsa.resources.store(4).location !=== user.location --> Flag.HIL > [Send] [Error] : : Message --> T̈he goal cannot be achieved. Sequence execution incomplete (Thrown by Task.proceedCheckout(paymentInvalidException)). Possible Sequences (1). Conflicts reported (4).¨ > [Info] ucsa.goals.id(1) CHANGED : [ADD] 0 Nodes, 0 Arcs - [DELETE] 0 Nodes, 0 Arcs - [UPDATE] 0 Nodes, 0 Arcs |

### 8.2.3 Scalability and Performance Analysis

We recognise that MARTs for our online grocery shopping case scenario are likely to be small (i.e., graphs around 100 nodes), which makes scalability and performance no much of a concern. However, UCSAs can be extended to other applications with bigger MARTs, such as smart cities. Intelligent traffic control is a case in the smart cities domain, which is characterized for its highly dynamic socio-technical ecosystem with automotive elements, pedestrians, environmental conditions, and unexpected road situations. Elements of the intelligent traffic control system comprise their own group of sensors, whether it is a car or a person, context information moves along the traffic lines from one sensor to another, thus affecting the representation of the current state. MARTs are proper representations for the realization of intelligent traffic control CPHSs, resulting in graphs that might be in the order of thousands of nodes and arcs. Given the possibility of a scenario with bigger MARTs, we analysed the capability of PRIMOR to support growing models and CRUD operations efficiently.

For the evaluation of the *scalability* and *performance* of PRIMOR, we used a variety of JUnit tests and adaptation requests sent from SUSGROCERIES to execute CRUD operations. We analysed the results of PRIMOR after their execution to assess PRIMOR's capacity of understanding, translating and applying CRUD operations, as well as to manage exceptions during the execution of the user's tasking.

#### Analysis Process

Our analysis was conducted following four steps:

1. **Creation of valid instances of our Galapagos Model:** We created two additional instances of our MART as depicted in Figures 8.4 and 8.5 as the base graphs for our evaluation. As a result, we have three available instances of the same MART as input data to increment the variability of our tests, and to reduce the bias of our experiments. Each of these three instances represents a valid sequence to achieve the same personal goal. The variability of the sequence is determined by the preferences of the user.

2. **Plan the growth of the MARTs instances:** We created *dummy* elements for `Goal, Task, Resource` and `Actor`, with the purpose of generating bigger

and more complex Galapagos Model types of graphs. We included a single activity in each *dummy task* with a time delay to simulate the time a task will take to be completed. We set the time to be a random number between 1 and 10, following the three limits for response time proposed by Nielsen [Nie93]. Since our scenario takes place on the internet, we take Nielsen's approach assuming that most web services are likely to follow this range.

3. **Experimental plan:** We defined fifteen types of unit tests described with the following sentences: (1) Delete-Goal, (2) Add-Goal-Goal, (3) Add-Task-Goal, (4) Add-Task-Task, (5) Add-Resource-Task, (6) Add-Actor-Goal, (7) Add-Actor-Task, (8) Update-Goal, (9) Update-Task, (10) Update-Actor, (11) Update-Resource, (12) Delete-Goal, (13) Delete-Task, (14) Delete-Actor, and (15) Delete-Resource. For the ADD operation, we generated a *dummy element* with a numeric id for identification. For the DELETE and UPDATE operations, we used existing elements.

4. **Observing results:** For each test, we focused on the expected result and the time it took Primor to effect such change. In addition, we examined whether the resulting operation was an expected operation.

It is worth noting that during this analysis, we did not evaluate the time the infrastructure might spend in conflict resolution since it requires additional inference capabilities not currently supported by Primor.

### Tests and Results

In these tests, we evaluated the capability of Primor to handle CRUD operations. Moreover, we evaluated the robustness of our operational framework in terms of the specification and runtime semantics of the MART supported operations. These tests were used in the refinement process of our operational framework and the functionalities of Primor.

**Scalability** In this analysis, *scalability* refers to the capability of our framework to support growing models and new operations. For this test, we added elements to the model with the following distribution:

- Goal : 10%

- Task : 50%
- Resource : 20%
- Actor : 20%
- Dependency : 20%

Other concepts, such as `Decomposition, Achievement` and `Responsibility` are consequence of the addition of their corresponding nodes, therefore the elements are created in the same distribution following the rules of the model.

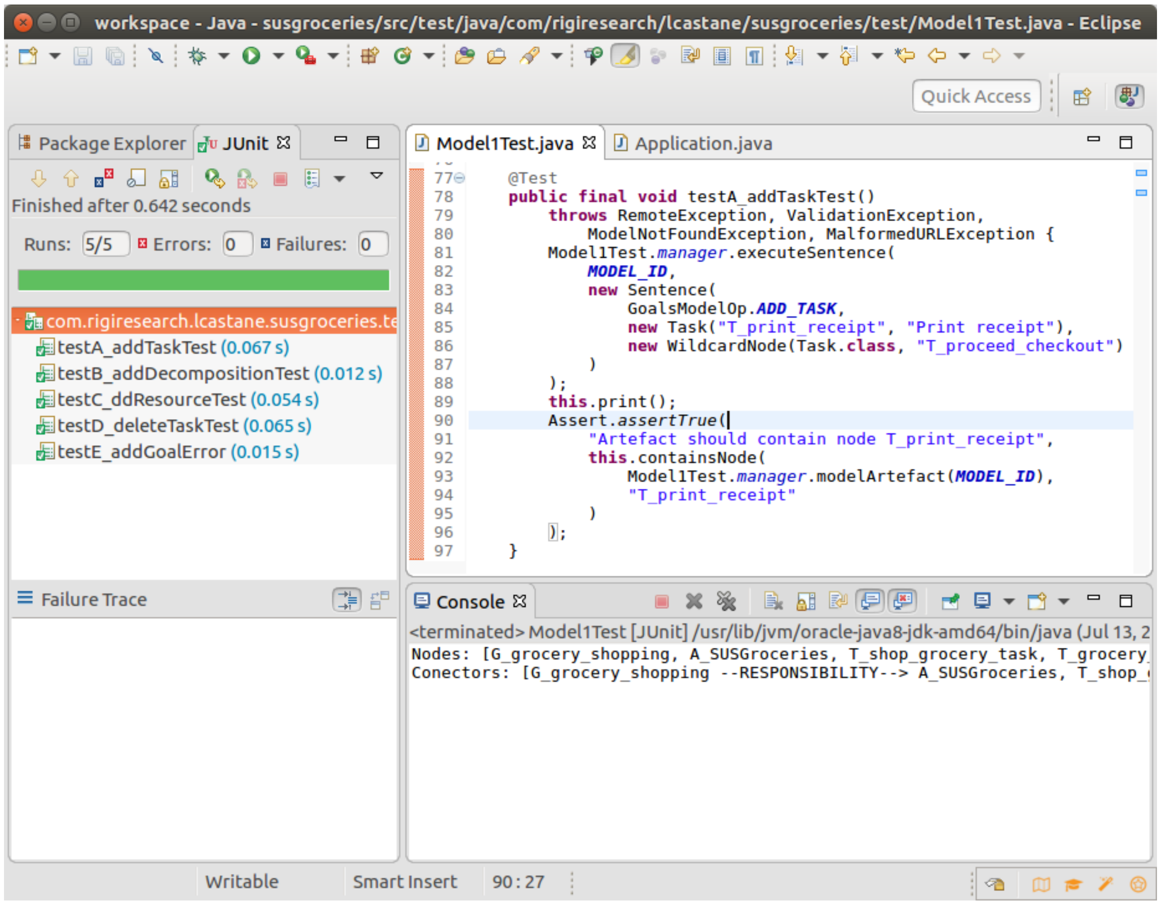
We ran the experiment 100 times using JUnit for each of our instances, for a total of 300 iterations. We used the console standard output to analyse the resulting models and to keep track of the changes as these occur. Figure 8.1 depicts a screenshot of our unit test environment using JUnit, showing an example of our JUnit test to execute a CRUD operation of adding a `Task`.[1]

Since we created random connections among the elements of the graph, every execution of the test presented different graph topographies. Table 8.8 presents an example of the model instance growth used in our test.

**Table 8.8:** Scalability Case: Model Instance A

| | **Base (B)** | **B+10** | **B+100** | **B+1000** | **B+5000** |
|---|---|---|---|---|---|
| **Nodes** | | | | | |
| Goals | 1 | 1 | 8 | 87 | 445 |
| Task | 8 | 12 | 57 | 507 | 2507 |
| Actors | 1 | 2 | 12 | 200 | 1000 |
| Resources | 1 | 2 | 20 | 200 | 1000 |
| *Total:* | *11* | *17* | *97* | *994* | *4952* |
| **Arcs** | | | | | |
| Decomposition | 8 | 12 | 57 | 507 | 2507 |
| Achievement | 1 | 1 | 7 | 63 | 200 |
| Dependency | 5 | 8 | 44 | 404 | 2004 |
| Responsibility | 5 | 5 | 5 | 5 | 5 |
| *Total:* | *19* | *26* | *113* | *979* | *4716* |

[1]The complete implementation of our tests are available through the project's repository in `http://www.rigireserch.com/research/pit`



workspace - Java - susgroceries/src/test/java/com/rigiresearch/lcastane/susgroceries/test/Model1Test.java - Eclipse

Finished after 0.642 seconds

Runs: 5/5 Errors: 0 Failures: 0

```
com.rigiresearch.lcastane.susgroceries.te
    testA_addTaskTest (0.067 s)
    testB_addDecompositionTest (0.012 s)
    testC_ddResourceTest (0.054 s)
    testD_deleteTaskTest (0.065 s)
    testE_addGoalError (0.015 s)
```

Model1Test.java

```java
    @Test
    public final void testA_addTaskTest()
        throws RemoteException, ValidationException,
            ModelNotFoundException, MalformedURLException {
        Model1Test.manager.executeSentence(
            MODEL_ID,
            new Sentence(
                GoalsModelOp.ADD_TASK,
                new Task("T_print_receipt", "Print receipt"),
                new WildcardNode(Task.class, "T_proceed_checkout")
            )
        );
        this.print();
        Assert.assertTrue(
            "Artefact should contain node T_print_receipt",
            this.containsNode(
                Model1Test.manager.modelArtefact(MODEL_ID),
                "T_print_receipt"
            )
        );
    }
```

Console

```
<terminated> Model1Test [JUnit] /usr/lib/jvm/oracle-java8-jdk-amd64/bin/java (Jul 13, 2
Nodes: [G_grocery_shopping, A_SUSGroceries, T_shop_grocery_task, T_grocery
Conectors: [G_grocery_shopping --RESPONSIBILITY--> A_SUSGroceries, T_shop_
```

**Figure 8.1:** Eclipse environment for our JUnit implementation

The goal of our analysis was to assess the capability of the framework to support the growth of the model with new elements and operations. As mentioned before, our running infrastructure PRIMOR employs the runtime semantics described in the operational framework to effect such modification.

The following step in our test was to create a new type of Node called `MiniTask` and run the experiments again. Table 8.9 presents a summary of the experiment for the model instance with the new type of node. In comparison, both the infrastructure and the model were able to support the creation of a new element in the model and execute the CRUD operations. By comparing both tables, we conclude that the distribution of the nodes was affected, since the new node had to share its distribution with her parent node (i.e., `Task`).

**Table 8.9:** Scalability Case: Model Instance A with a new type of `Node`

| | **Base (B)** | **B+10** | **B+100** | **B+1000** | **B+5000** |
|---|---|---|---|---|---|
| **Nodes** | | | | | |
| Goals | 1 | 1 | 8 | 87 | 445 |
| MiniTask | 8 | 8 | 39 | 407 | 2007 |
| Task | 8 | 8 | 30 | 303 | 1507 |
| Actors | 3 | 3 | 3 | 3 | 3 |
| Resources | 1 | 2 | 20 | 200 | 1000 |
| **Arcs** | | | | | |
| Decomposition | 8 | 11 | 47 | 407 | 2007 |
| Achievement | 1 | 1 | 1 | 1 | 1 |
| Dependency | 5 | 8 | 44 | 404 | 2004 |
| Responsibility | 5 | 5 | 5 | 5 | 5 |

Although the test shows that the model is capable of growing in terms of adding new elements at execution time (both existing and new concepts), the elements need to be defined in the Galapagos Model. Growing the model from the conceptual point of view, implies modifications to the operational framework to define the mapping between the notation and the artefact, as well as the definition of the runtime semantics. Our infrastructure is designed in terms of a multilayer-definition of Java classes and interfaces, which allows to work at different levels of abstraction of the model. As a result, a model is capable of propagating changes to its layers of abstraction with our design.

**Performance** With the previous analysis, we assessed that the model instance, and the infrastructure support growing models. However, the main challenge for this analysis was to recreate the runtime environment and analyse how the CRUD operations are affected by the size of the model and the complexity of the operations. To recreate the runtime scenario, we created a function in SUSGroceries to schedule five CRUD operation requests to Primor, scheduled every 0.01 ms. At the same time, we run a separated process to request the growth of the model in Primor using the environment for our *scalability* analysis.

Source Code 8.1 depicts simplified sections of the five CRUD operations we used in the performance test: (1) add `Task`, (2) add `Decomposition`, (3) add `Resource`, (4) delete `Task`, and (5) add `Goal`. We selected these five from our catalogue of operations to be the ones with more time consuming activities, such as validations, preconditions, or postconditions that might affect the performance of PRIMOR.

**Source Code 8.1:** Simplified view of the source code for the five CRUD requests sent from SUSGROCERIES for this experiment

```

public final void CRUD1(){
  this.primor.sendRequest(
    request.type.ADAPTATION(
      manager.modelArtefact(modelId),
      new Sentence(
        GoalsModelOp.ADD_TASK,
        new Task("T_print_receipt", "Print receipt"),
        new WildcardNode(Task.class, "T_proceed_checkout")
      )
    )
  );
}

public final void CRUD2(){
  this.primor.sendRequest(
    request.type.ADAPTATION(
      manager.modelArtefact(modelId),
      new Sentence(
        GoalsModelOp.ADD_DECOMPOSITION,
        new WildcardNode(Task.class, "T_proceed_checkout"),
        new WildcardNode(Task.class, "T_print_receipt")
      )
    )
  );
}

public final void CRUD3(){
  this.primor.sendRequest(
    request.type.ADAPTATION(
      manager.modelArtefact(modelId),
      new Sentence(
        GoalsModelOp.ADD_RESOURCE,
        new Resource(
          "R_printer_1",
          "Printer",
          new URL("http://localhost/printer1"),
          "printer",
          false
        ),
        new WildcardNode(Task.class,"T_print_receipt")
      )
    )
  );
}

public final void CRUD4(){
  this.primor.sendRequest(
    request.type.ADAPTATION(
      manager.modelArtefact(modelId),
      new Sentence(
        GoalsModelOp.DELETE_TASK,
        new WildcardNode(Task.class, "T_print_receipt")
      )
    )
  );
}


public final void CRUD5(){
  this.primor.sendRequest(
    request.type.ADAPTATION(
      manager.modelArtefact(modelId),
      new Sentence(
        GoalsModelOp.ADD_GOAL,
        new Goal(
          "G_unvalidGoal",
          "this should not be added"
        )
      )
    )
  );
}
```

We ran our scenario 100 times to collect a significant amount of samples, and then we ran the scenario 900 more times in sets of 10. We decided to stop in 1,000 total of executions since the executions were not presenting significant changes in the results. Table 8.10 presents a summary of maximum time execution, average and standard deviation for our 1,000 executions per model instances (i.e., A, B and C) for each one of our operations. We used these results for the next comparisons combining CRUD operations and model growth. Our three instances are different in their composition (i.e., their tasks , but similar in their size (i.e., the number of nodes and arcs in the graph). There is not much difference in the results of the test, therefore, we need to test PRIMOR by incrementing the complexity of the connection and the size of the models. Figure 8.2

**Table 8.10:** Performance Case 1: CRUD operations time of execution on the model instances A, B, C for the base size (no growth). Time measured in seconds

| | **CRUD 1** | **CRUD 2** | **CRUD 3** | **CRUD 4** | **CRUD 5** |
|---|---|---|---|---|---|
| **Instance A** | | | | | |
| **Max** | 21.00 | 19.00 | 23.00 | 38.00 | 13.00 |
| **Average** | 3.71 | 3.84 | 6.35 | 11.28 | 2.17 |
| **StdDev** | 0.004 | 0.004 | 0.005 | 0.006 | 0.003 |
| **Instance B** | | | | | |
| **Max** | 65.00 | 26.00 | 47.00 | 95.00 | 31.00 |
| **Average** | 4.41 | 4.42 | 6.23 | 13.33 | 2.68 |
| **StdDev** | 0.007 | 0.005 | 0.006 | 0.012 | 0.004 |
| **Instance C** | | | | | |
| **Max** | 32.00 | 23.00 | 23.00 | 36.00 | 12.00 |
| **Average** | 3.88 | 3.74 | 6.35 | 13.61 | 2.28 |
| **StdDev** | 0.004 | 0.003 | 0.004 | 0.007 | 0.002 |

For this experiment, we ran the test 1,000 times with the aid of JUnit. Every test executed five CRUD operations and a growth request (B+10, B+100, B+1000, B+5000). A total of 4,000 experiments were executed for every model instance (three model instances). Table 8.11 presents a summary of the results of this experiment in terms of the maximum values, average and standard deviation for the experiments on

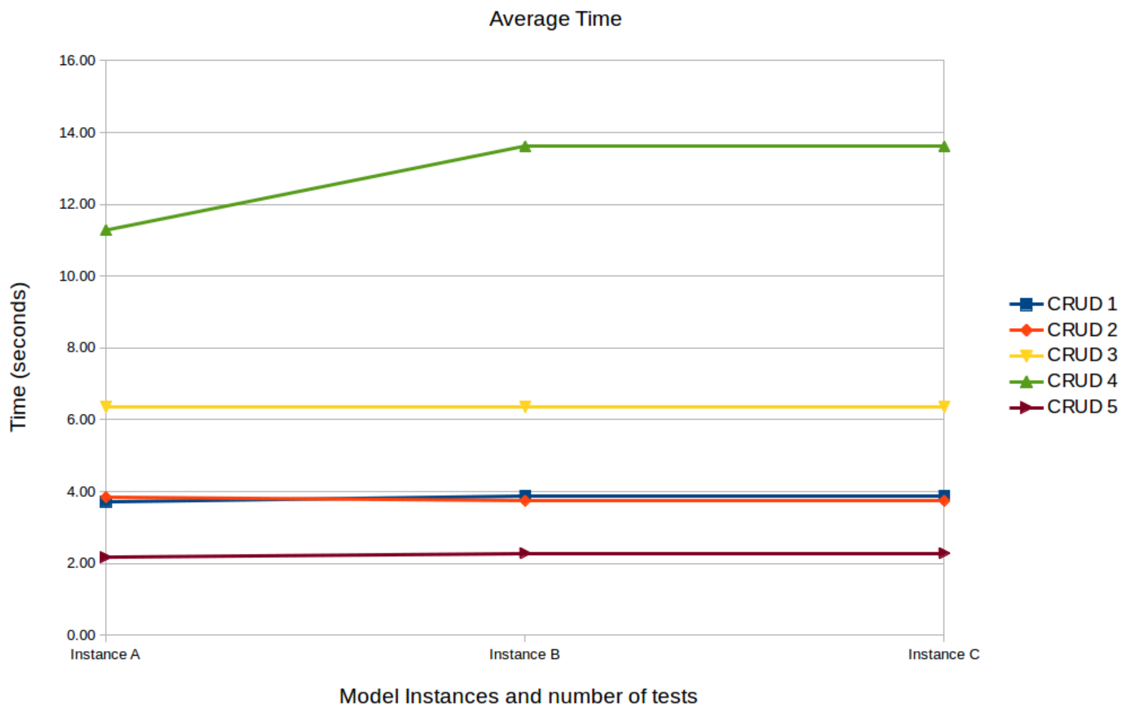


**Figure 8.2:** Plot depicting the average time for the experiments

one base model instance (i.e., Instance A). As expected, the time it takes PRIMOR to execute CRUD operation becomes bigger as the graph grows in size and complexity. Since the operations were the same in every test, we selected the five CRUD operations with more validation rules according to our operational framework. Our underlying assumption is that validations take longer, therefore the CRUD operation execution time might be bigger. Figure 8.3 depicts the average time for the experiments showing a linear behaviour.

**Table 8.11:** Performance Case 2: Maximum time, Average and Standard Deviation. The base is Model Instance A

| | **CRUD 1** | **CRUD 2** | **CRUD 3** | **CRUD 4** | **CRUD 5** |
|---|---|---|---|---|---|
| **Average** | | | | | |
| **B+10** | 1.733 | 1.74 | 3.228 | 6.907 | 1.121 |
| **B+100** | 2.546 | 2.545 | 4.831 | 11.473 | 0.911 |
| **B+1,000** | 12.628 | 12.458 | 24.532 | 61.54 | 0.791 |
| **B+5000** | 60.509 | 60.243 | 119.629 | 303.562 | 0.934 |

| | CRUD 1 | CRUD 2 | CRUD 3 | CRUD 4 | CRUD 5 |
|---|---|---|---|---|---|
| **Standard Deviation** | | | | | |
| **B+10** | 1.81 | 2.13 | 3.21 | 5.82 | 1.92 |
| **B+100** | 3.68 | 2.87 | 6.57 | 12.52 | 0.67 |
| **B+1,000** | 2.24 | 2.06 | 2.81 | 6.93 | 1.54 |
| **B+5,000** | 15.71 | 14.70 | 24.78 | 53.64 | 0.71 |
| **Max (seconds)** | | | | | |
| **B+10** | 20 | 36 | 37 | 89 | 42 |
| **B+100** | 50 | 31 | 27 | 82 | 18 |
| **B+1,000** | 68 | 49 | 144 | 314 | 11 |
| **B+5,000** | 316 | 163 | 272 | 752 | 10 |

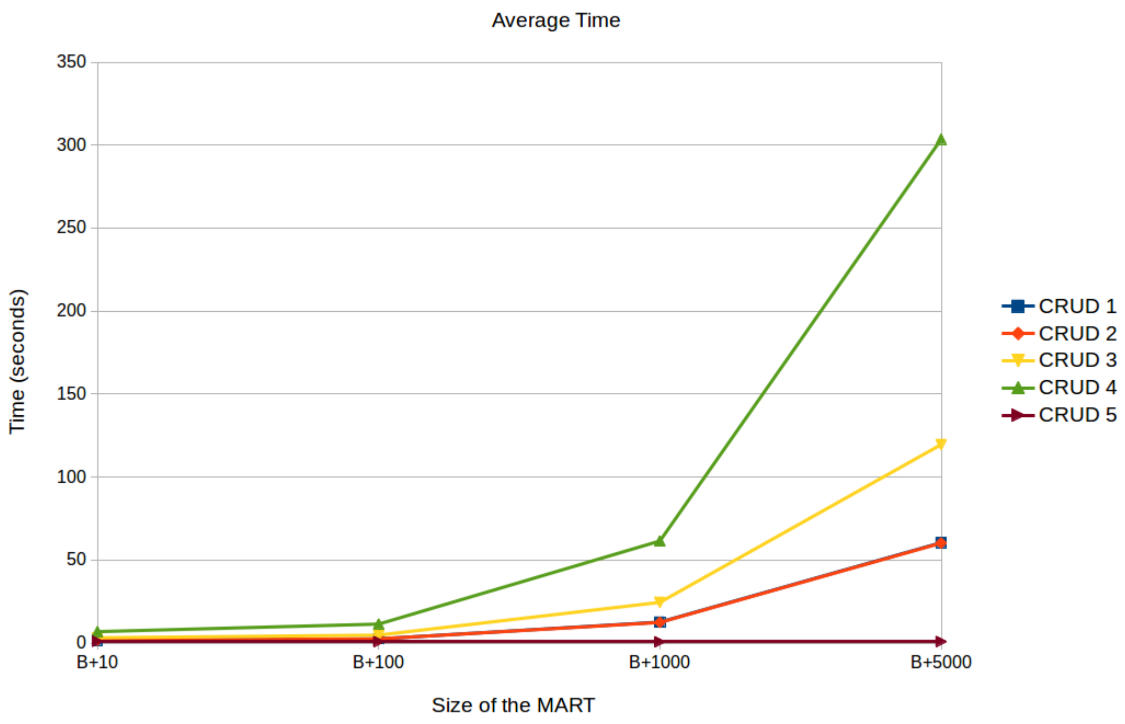


**Figure 8.3:** Plot depicting the average time for the experiments in growing MARTs

In terms of *performance* of the infrastructure, we assessed the time that our infrastructure takes to perform CRUD operations on MARTs. Our scenario with the biggest graph was the one with a maximum of 5,000 nodes and approximately 3,000 arcs (B+5,000). CRUD 4, which is the operation to delete a `Task`, takes an average of 304 seconds to be executed. However, with a standard distribution of 53.64, the results of this test show that the data points are widely distributed.

We attribute the time of execution to various factors:

- **Size**: The size of the graph affects the time due to the process it takes to find the place to insert a new element (node or arc). We use the Breadth-first search (BFS) algorithm for traversing the graph and follow an Object-Oriented paradigm implementation.

- **Validation:** Some of our model validations include the verification of the task sequence in terms of the accomplishment of the specified goal. *Assurance* is highly valued in the research area of Models at Runtime. In our contribution, our operational framework, lays the structure for the definition of validation rules applied to CRUD operations, and policies associated with the execution of the personal tasking. However, the process of validation increases the processing time of the CRUD operations. The alternative we implemented to reduce this time was the use of separate agents to execute the validation as a mechanism to support parallel processing. Given the main focus of this dissertation, we did not implement extensive tests with our agents. However, our infrastructure design is meant to be extended in other applications scenarios focused on assurance.

Given that CRUD 4 is an operation on a `Task`, by our Galapagos Model we can foresee that removing a `Task` implies a set of validations and conditions that are more extensive compared with the other elements in the MART.

In contrast, CRUD 5 in B+5,000, which is the operation to add a `Goal` presents a standard deviation significantly smaller. The reason lies in the nature of a `Goal` in the definition of the Galapagos Metamodel. Since a `Goal` usually is at the beginning of the graph and their validations to add are significant but few, the operation is executed efficiently.

It is worth noting that the standard distributions for B+10, B+100 and B+1,000 are relatively small. Based on the premise that our Galapagos Model represents the tasking of the user, we can predict that a tasking of 1,000 nodes (approximately 500 might be `Task`) is a rare case scenario for a personal task. In this scenario, CRUD 4 took an average of 61 seconds. Based on the experiences during the course of this research, a personal tasking of the user is probably modelled with a B+100 type of graph size. In this case, CRUD 4 took an average of 11 seconds. That being said, we conclude that our model, operational framework, and infrastructure, are capable to support growing models and CRUD operations at execution time with a performance close to the upper limits for response time proposed by Nielsen of 10 seconds [Nie93].

## 8.3 Chapter Summary

This chapter presented the evaluation of the contributions of this dissertation. First, we presented a qualitative analysis to demonstrate the novelty of our MARTs by comparing it with related approaches. As a result, we concluded that our models represent the dynamic nature of CPHSs while making users the principal element during the execution of their tasking. More importantly, our models as runtime models provide capabilities to evolve at execution time, which contributes to the growing socio-technical ecosystem of CPHSs.

Second, we presented an experimental analysis to demonstrate the applicability of our MARTs and Primor. We conducted several experiments and scenarios to analyse our contributions in terms of three quality attributes: *accuracy, scalability*, and *performance* [Gor11]. The experimental results of this analysis provides evidence that our MARTs are capable of representing changing user situations, dynamic tasking and the achievement of a personal goal. Moreover, our operational framework and infrastructure (i.e., Primor) support the growth of MARTs during execution time, and are designed to be extended to other MARTs and application domains given its modular and flexible design.

The next chapter of this dissertation summarizes the research and contributions of this dissertation, presents the conclusions and discusses potential future work.

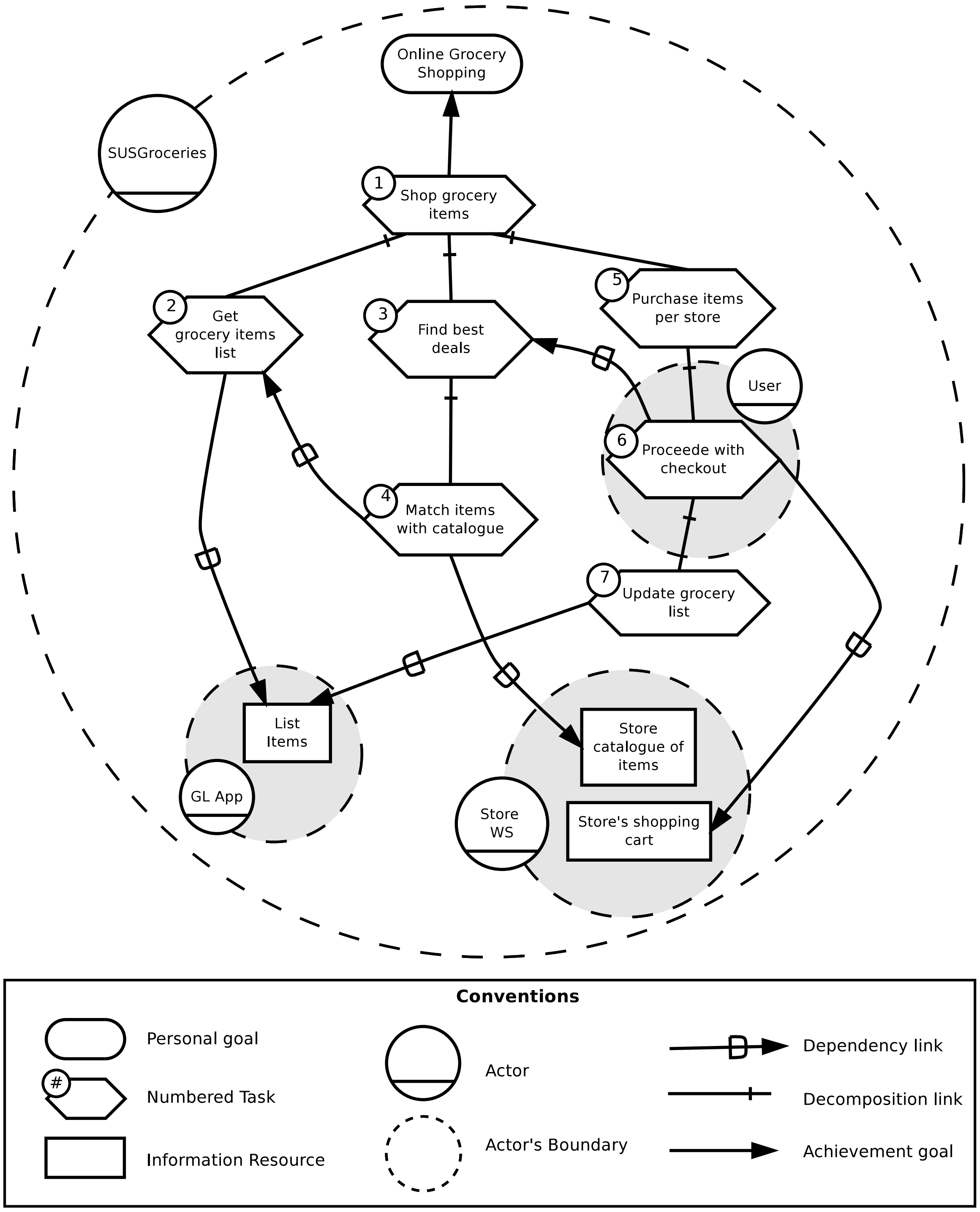


**Figure 8.4:** Simplified instance of the GALAPAGOS MODEL for the online grocery shopping example (Instance B)

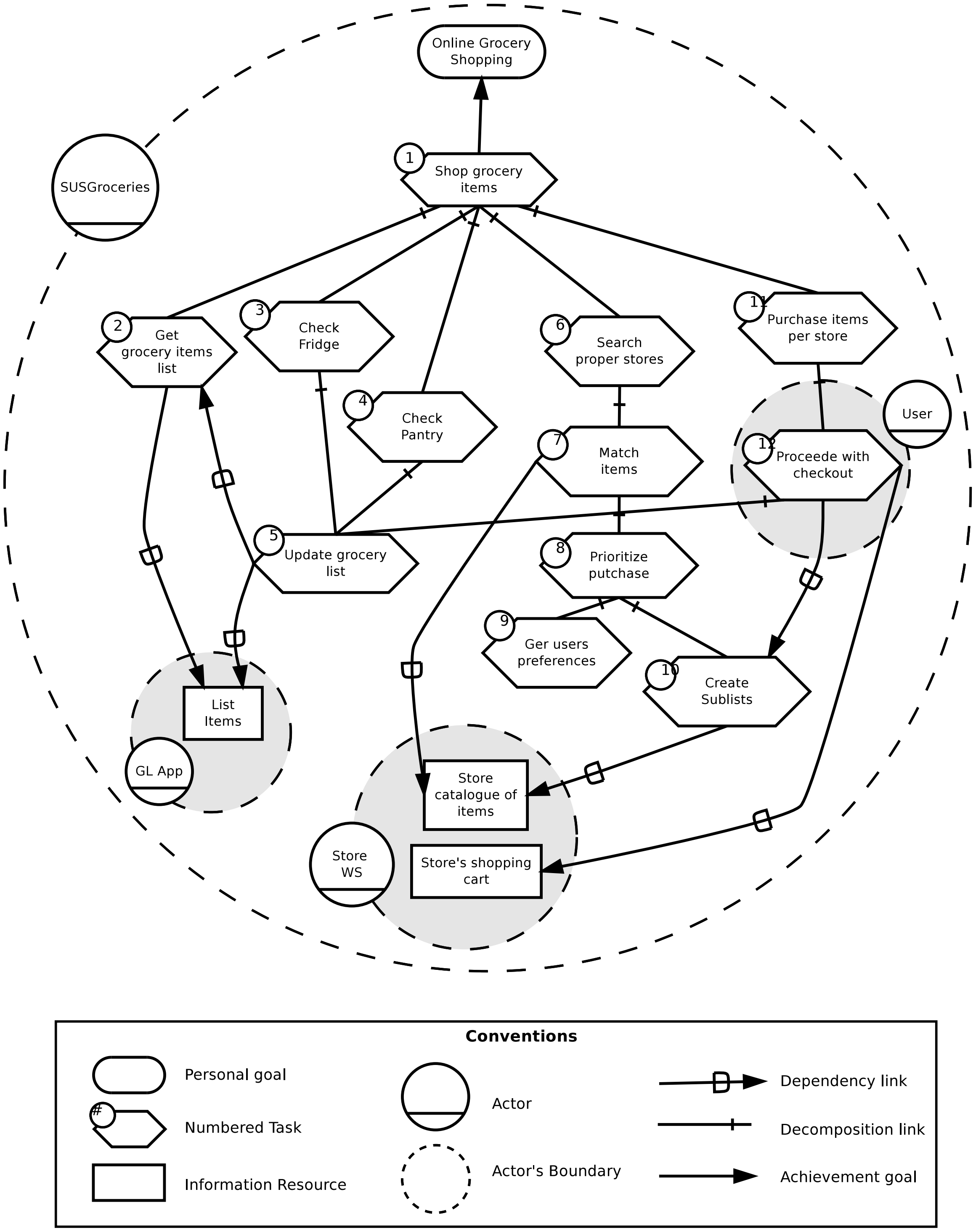


**Figure 8.5:** Simplified instance of the GALAPAGOS MODEL for the online grocery shopping example (Instance C)

# Chapter 9

# Summary, Discussion and Future Work

This chapter summarizes this dissertation by revisiting our research challenges, goals and contributions. We also point our selected limitations in the realization of runtime models and infrastructures for User-Centric Smart Cyber-Physical-Human Applications (UCSAs), as well as in the application of PRIMOR in other domains. Finally, this chapter concludes this dissertation with a discussion of future work.

## 9.1 Dissertation Summary

Cyber-Physical-Human Systems (CPHSs) integrate *cyber*, *physical*, and *human* components to work together towards the achievement of the objectives of the system [SSZ+16]. CPHSs will become larger, more complex and users will be deeply involved. Users constantly rely on their technology to fulfil personal goals, thus becoming active, relevant, and necessary components of the designed system.

However, humans are highly dynamic, their decisions might not always be predictable, and they expose themselves to unforeseeable situations that might impact their interactions with their physical and cyber elements. With the human in the loop, CPHSs need to understand and respond to the dynamic environment introduced by users, while assisting them in the achievement of their personal goals. The design of smart CPHSs requires a user-centric vision and runtime adaptation capabilities. Models at Runtime (MARTs) are up-to-date representations of the system, environment and users. MARTs enable CPHSs to represent changing situations and context information while the system executes, empowering the CPHSs to reason and adapt on runtime information.

The two motivations that drove this research concerns the need for (1) empowering CPHSs with situation-awareness to understand users' context and changing situations; and (2) runtime models and infrastructures to represent and manage CPHSs dynamic requirements based on user-centric concerns and situations.

The research problem addressed in this dissertation was to investigate how to design runtime models and infrastructures to support CPHSs' user-centric requirements, such as (1) understanding users, their personal goals and changing situations, (2) causally connecting the cyber, physical and human components involved in the achievement of users' personal goals, and (3) supporting runtime adaptation to respond to relevant changes in the users' situations.

### 9.1.1 Addressed Challenges

We classified our research challenges into two groups: *situation-awareness* and *runtime adaptation*. The challenges that we addressed in this dissertation are summarized as follows:

#### Situation-awareness

CH1. Specifications that explicitly connect users with personal goals and relevant context.

CH2. Comprehensive representations of user's tasks and sequences and measurable outcomes.

CH3. Representations and reasoning techniques to infer emerging situations.

#### Runtime adaptation

CH4. Runtime models to make explicit the components of CPHSs and their interactions.

CH5. Architectural and functional requirements of CPHSs to support runtime user-centric awareness and adaptation.

CH6. Runtime adaptation techniques to support dynamic changes in the specification of the CPHSs' runtime models.

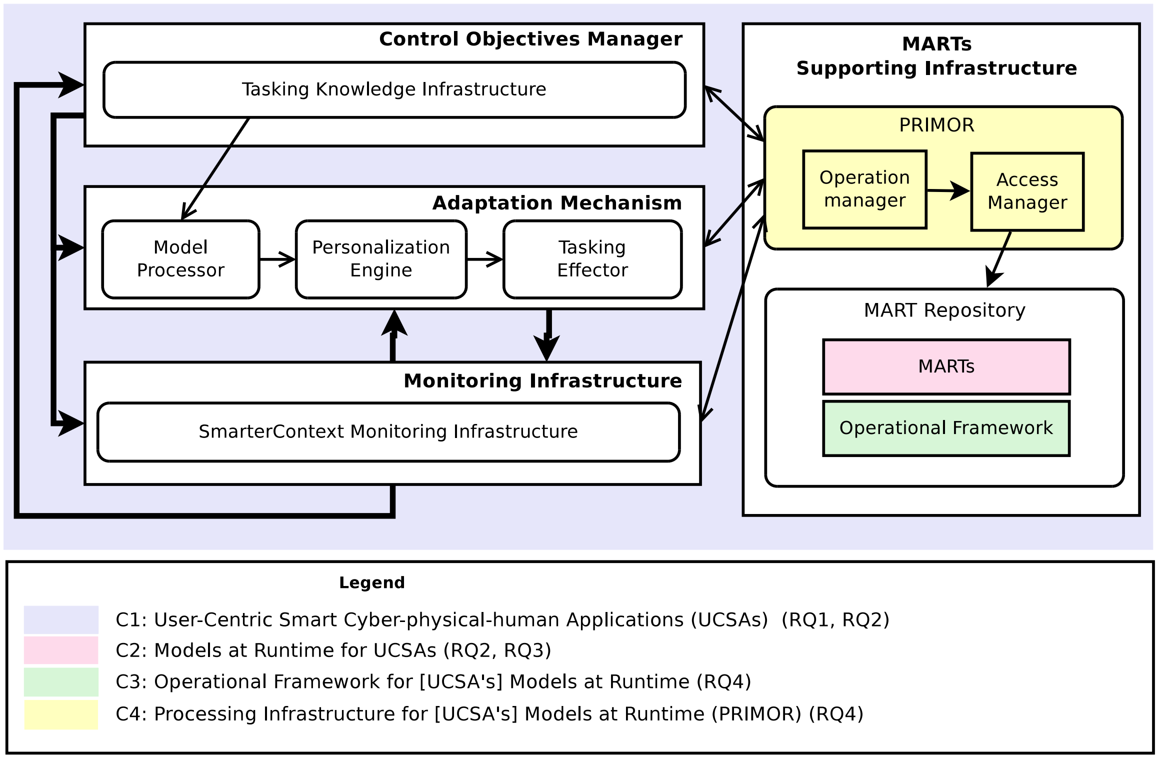

Control Objectives Manager
Tasking Knowledge Infrastructure
Adaptation Mechanism
Model Processor
Personalization Engine
Tasking Effector
Monitoring Infrastructure
SmarterContext Monitoring Infrastructure
MARTs Supporting Infrastructure
PRIMOR
Operation manager
Access Manager
MART Repository
MARTs
Operational Framework
Legend
C1: User-Centric Smart Cyber-physical-human Applications (UCSAs) (RQ1, RQ2)
C2: Models at Runtime for UCSAs (RQ2, RQ3)
C3: Operational Framework for [UCSA's] Models at Runtime (RQ4)
C4: Processing Infrastructure for [UCSA's] Models at Runtime (PRIMOR) (RQ4)

**Figure 9.1:** This dissertation's contributions

### 9.1.2 Contributions

This section summarizes our contributions. Figure 9.1 depicts our contributions in relation with each other through the architecture of User-Centric Smart Cyber-Physical-Human Applications (UCSAs). Contribution *C1* corresponds to our characterization and architectural design of UCSAs, a set of cyber, physical and human components that assist users in the fulfilment of personal goals. Contributions *C2* and *C3* constitute our MART (i.e., Galapagos Metamodel and Galapagos Model), and their operational framework, which defines UCSAs' runtime representations and CRUD operations. Finally, contribution *C4* is our Processing Infrastructure for Models at Runtime (PRIMOR), which is responsible for controlling the access to the MARTs and managing their CRUD operations.

#### C1: User-Centric Smart Cyber-Physical-Human Applications

We focused on two major components of CPHSs: (1) humans as first class elements, which define the system's objectives to be the achievement of users' personal goals while exploiting personal context; and (2) the software component that is responsible for the orchestration of the cyber, physical and human components of the system, as well the adaptation requirements under changing conditions. With growing socio-technical ecosystems and the need for smarter internet applications, we defined User-Centric Smart Cyber-Physical-Human Applications (UCSAs) (cf. Definition 4.1) as follows:

> *a UCSA as an orchestrated set of cyber, physical, and human components (along with their interconnections) that assist users in the fulfilment of their personal goals. A UCSA manages the smart interaction among the components dynamically, understands and acts upon users' changing situations, and has capabilities to evolve at runtime.*

UCSAs' are characterized by: (1) *user awareness*, which is the explicit identification of the *human dimension* as a sphere of information containing users' concerns and personal data; (2) *runtime modelling support*, which is the capability of the UCSA to store and manage runtime models; and (3) *runtime adaptation support*, which is the capability to propagate changes across the models at runtime.

Our architectural design for UCSAs (cf. Figure 9.1) defines a self-adaptive context driven software system based on the DYNAMICO reference model [VTM+13]. We defined five components for UCSAs: (1) the *tasking knowledge infrastructure* responsible for understanding and representing personal goals based on the analysis of users' task interactions; (2) the *model processor*, in charge of translating the information about users' tasking into runtime models; (3) the *personalization engine* exploits static and dynamic personal context to tailor an appropriate task sequence; (4) the *tasking effector* that executes the tasks on behalf of the user while providing runtime adaptation support and (5) the *MARTs supporting infrastructure* responsible for managing and hosting the corresponding MARTs.

## C2: MARTs for User-Centric Smart CPH Applications

A UCSA manages the smart interactions among the cyber, physical and human components dynamically and acts upon users' changing situations by evolving at runtime. Models at Runtime (MARTs) are fundamental for UCSAs to represent and understand users' personal goals and changing situations at runtime.

We define two MARTs for User-Centric Smart Cyber-Physical-Human Applications (UCSAs): (1) our GALAPAGOS METAMODEL, and (2) our GALAPAGOS MODEL.

First, our Galapagos Metamodel defines the concepts of UCSA by abstracting the three dimensions of CPHSs as well as the smart interactions among them. Our Galapagos Metamodel represents situation-awareness components through the *Situation* entity, and time and space descriptors. More importantly, in our Galapagos Metamodel, personal goals and situations are connected in two manners: (1) situations influencing the achievement of personal goals, and (2) personal goals associated with situations.

Second, our Galapagos Model is specified using *G-iStar*, our extension and adaptation of the iStar framework to support dynamic personal goals and task interactions. Our model supports the specification of evolving tasking goals, personal interactions, and the relevant contexts. In particular, we extended the iStar atomic notions of actor, goals, task and resources, to support the specification of tasking goals, task sequences, and relationships among goals, tasks, actors, and resources. Our Galapagos Model provides the representation of a personal goal that is fulfilled through the execution of an ordered sequence of internet tasks. Moreover, our model can represent the sequence as well as the dependency among tasks, and the decomposition into subtasks. Tasks often require to access information in the cyber or physical dimensions, represented in our model as information resources. Our Galapagos Model provides a mechanism to represent the composition and execution of UCSAs.

For the evaluation of this contribution, we performed a qualitative analysis to demonstrate the novelty of our approach by comparing it with related approaches. Throughout this dissertation, we demonstrated that our models satisfy MARTs characteristics, therefore making them proper models at runtime. Therefore, our models: (1) satisfy the representation of users' task interactions that lead to the achievement of a particular personal goal; (2) are available to software systems; (3) maintain a connection with their reality (i.e., users' personal tasking and goals); and (4) are capable to evolve at execution time.

In addition, we instanced our Galapagos Model to evaluate its feasibility during the implementation of our online grocery shopping scenario. During this part of the evaluation, we analysed the *accuracy* of our MARTs, assessed through the capability of our models to support the representation of relevant changes in the context. During our experiments, we modified two pieces of context information about the user: *location* and *preference*. According to our results, our MART is capable to support the representation of changing context based on the users' personal situations.

## C3: Operational Framework for Models At Runtime

As part of the runtime realization of MARTs, we specified runtime operations that are supported by the model. We defined that such information must exist in the operational framework of the MART. Our framework defines model equivalences between human-readable and machine-readable, available runtime operations and semantics, to manage runtime operations on MARTs. Our operational framework comprises four components: (1) a notation-artefact mapping between the MART human-readable and machine-readable views; (2) a catalogue of operations that describes the considerations and restrictions of every element in the model for runtime CRUD operations; (3) the runtime semantics to execute the operations in the MARTs; and (4) the *causal links* component, which is shared among the MARTs and describes causal connections among them.

The mapping is the main element for the translation of the MARTs in the two ways: human readable and machine readable. The catalogue of runtime operations defines for every element what the limitations, restrictions, and other considerations are when performing runtime operations. We defined three operations, however the catalogue is flexible in that it can adapt to any supported runtime operation of MARTs. The runtime semantics are specified in a programming language and are implemented by the smart infrastructure to execute its operations (i.e., software commands) at runtime. Finally, the operational framework describes the causal connections between the MARTs through the *causal links* component, which is shared by all MARTs.

For the evaluation of this contribution, we performed an experimental analysis focused on the runtime operations specified in the framework as supported by the corresponding MART. During this part of the evaluation, we analysed the *scalability* of our approach, assessed as the capability of our framework to support growing models and new operations. During our experiments, we executed a diversity of CRUD operations and analysed the capability of the framework to support the corresponding specifications based on the runtime semantics available for the MART. Our framework showed to be sufficient to execute basic CRUD operations based on the catalogue of operations. More importantly, its design facilitates the process of including new operations and semantics for other application domains.

## C4: Processing Infrastructure for Models at Runtime (PRIMOR)

Our last contribution is our Processing Infrastructure for Models at Runtime (PRIMOR) to manage operations on MARTs for UCSA. Primor is a component of the UCSA's models at runtime supporting infrastructure. More importantly, Primor is a component-based system designed to be extensible to other domains.

Primor manages runtime operations for our two MARTs: the Galapagos Metamodel and the Galapagos Model. Primor's services are to provide reading access from software components to the MARTs, execute model-related runtime operations, and manage the propagation of changes among interconnected MARTs and their realities. For these purposes, Primor receives two types of requests: an *Access Request* and an *Adaptation Request*. An *Access Request* asks for reading capabilities on MARTs, which is handled by the *Access Manager* component. An *Adaptation Request* asks the execution of changes on the MARTs, which is handled by the *Operation Manager* component.

Handling the *Adaptation Request* consists of four activities executed by the *Operation Manager*: (1) *operation identification*, which analyses the Sentence in the *Adaptation Request* based on the operational framework of the MART; then (2) *causal link analysis*, which prepares the internal and external propagation of changes by analysing the causal links of the MART and creating new sentences for the additional operations; (3) *operations plan, execution and assurance*, which composes an execution plan, coordinates its execution, and controls runtime exceptions by taking corrective or informative actions; and (4) *MARTs' deployment*, which deploys the new versions of the MARTs in the repository, and guarantees both views (notation and software artefact) remain truthful to each other.

For the evaluation of this contribution, we performed an experimental analysis focused *performance* of our infrastructure, assessed as the time it takes our infrastructure to perform CRUD operations on MARTs. More importantly, we used the capabilities of our operational framework to produce growing instances of our Galapagos Model to analyse how the size of the model would affect the performance of Primor when executing CRUD operations. In addition, we defined five CRUD operations that would represent different expected behaviours in our experiments to increase the degree of variability. The results of our experiments demonstrated the practical feasibility of Primor to manage CRUD operations and growing MARTs with an acceptable performance with some opportunities for improvement.

## 9.2 Limitations

This section discusses the aspects that may constitute limitations on the applications of our MARTs and Primor to an actual setting in the short-term. Most of these limitations are in the concept of the socio-technological ecosystem described in this dissertation. Nevertheless, we present these limitations as opportunities for future research and collaborative work.

### Availability of Context Sources to "Get to Know" the User

To represent user's personal goals, tasking and changing situations effectively, UCSAs need to access relevant information about users to understand them. The *knowledge tasking infrastructure* component of the UCSA's architectural design is responsible for this specific task. Unfortunately, the current configuration of CPHSs limits the operation of this component.

There are several aspects regarding the availability of the context sources that need to be solved to realize the full potential UCSAs. Ordinarily, current systems do not expose themselves to third-party applications to collect information they have already gathered about the user. The reasons behind this behaviour are mostly commercial. Most software and service companies are competing to get to know the user and provide a better-personalized experience through their applications, thus, marketing themselves as providers of personalized. Over time, users find themselves invested in applications where they have already provided much information, thus making it very difficult for them to move to a new and probably better service. The decision lies between staying with the service that already knows them, or starting all over again with a risk of future disappointment if the service does not meet users' expectations.

The implementation of UCSAs removes that concern from the users and software providers, given that the personal information travels with the user. UCSAs can focus on mechanisms and innovative ways to exploit personal context, understand the user, and enhance the experience of the user by connecting services and applications inside and outside their personal sphere. More importantly, our current socio-technical ecosystem is quickly moving towards a technological platform where applications need to cooperate to meet users services demands. Application who have already made it through the integration of services and applications have succeeded to gain popularity among users.

Even when applications decide to cooperate, there is the issue of interoperability. To be able to share information, the transmission mechanisms and the format of the data needs to be compatible with the systems. Our UCSAs propose the use of MARTs, which are representations at different levels of abstraction, one of which might be the data specification about the format and the transmission. MARTs can become a language that systems speak, since these dynamic models exist at execution time, are available to software systems, and can evolve at runtime. With the limitation of available context information to understand the user, this dissertation focused on the implementation of MARTs to prepare CPHSs for the moment in time where cooperating and sharing will no longer be an issue.

## Growing Size of Context Data Results in Complex MARTs

Representing information at execution time is effective using MARTs. However, the rapid growth of context information implies that the MARTs also increase their size and complexity. UCSAs rely on MARTs to represent the evolving task of users to achieve a personal goal. The information that represents all the tasks, policies, resources, sequences, and personal preferences of the user, need to be defined inside the instance of the model. Thus, the MART represents all the necessary information for the UCSA to execute the tasking on behalf of the user. Although MARTs can support the representation, the growth of the context data incurs in limitation for the techniques, methods and time required to execute important tasks of the MARTs, such as discovering information, verification and validation of the task sequence, and assurance of the adaptations requirements. Even ordinary tasks like searching for an element in the model or making any modification to the model's structure becomes an issue when the MART is big and complex.

## Privacy and Confidentiality

The user-centric vision of UCSAs relies on the capability to understand users and exploit the personal context, specifically their internet interactions and preferences. A major concern from the perspective of users is privacy and confidentiality when it comes to systems trying to know users. The core principle of UCSAs is that users own and control their personal information through the implementation of the Personal Context Sphere (PCS) proposed by Villegas [Vil13]. Moreover, since UCSAs live in the domain of the user, they work towards the assistance of helping users achieve personal goals by using the information that users have authorized UCSAs to use.

The implementation of UCSAs in the short-term is possible with single repositories of personal context in compliance with the PCS structure. Agents that act as PCS collectors and managers can be created on the users' cloud profile the same way internet identities, such as email currently work.

## 9.3 Future Work

This dissertation concludes with a presentation of selected future work opportunities emerging from our research.

### Tasking Knowledge Global Repository

UCSAs are applications that assist users in the achievement of personal goals. For many cases, *personal goals* and *tasks* are common among users. We are interested in creating a shared repository in the cloud for personal goals and tasks where UCSAs can access a knowledge base to compose task sequences. With the implementation of MARTs, goals and tasks become pieces of information that can be understood, shared and reused from different UCSAs. This repository might be useful for discovering new sequences of tasks that achieve a personal goal, alternative tasks to improve a current sequence or publish updated services that might impact users' situations.

Since the personal information, what makes the execution of the UCSA unique, resides on the side of the user, there is no risk of sharing personal information in this repository. In a way, this repository would improve users' achievements of personal goals by having access to the knowledge of how other users with similar interests achieve the same goal. Just like humans do in their daily lives.

### Personal Context Sphere Agents in the Cloud

We have mentioned before that the current socio-technical ecosystem is a platform where personal information is constantly exchanged. We are interested in the development of *Personal Context Sphere Agents (PCSAs)*, which are smart virtual sensors that live in the cloud and assist users in the composition of their personal context sphere (PCS) by collecting and representing personal context information. These agents record users' preference, interactions and relevant context, and store them in MARTs.

Since PCSAs are smart virtual sensors some research challenges include the investigation of non-intrusive mechanisms to collect information, privacy and security assurances, and communication interfaces to connect PCSAs with services, applications, and UCSAs. A UCSA communicates with a user's PCSA to understand her preferences and exploit personal information to improve the personalization of her tasking.

## Situation-Awareness and Internet Personalities

Situation-awareness is the understanding of users' changing situations based on the analysis of the personal context. Personal goals can be associated with the various personalities of users while interacting with their UCSAs. In other words, users' situations depend on users' personalities while achieving personal goals. For example, the achievement of the grocery shopping goal is different for the same user when executing the task as a family member or as an employee in charge of an office social event. UCSAs are viable implementations for various domains where users' personal goals are the objective of the system, such as e-commerce, healthcare or transportation.

Since UCSAs assist users in the achievement of their goals, adding capabilities to the *personalization engine* component to differentiate the personalities executing the task might enhance the degree of task personalization. More importantly, it can create personal context sub-spheres associated with the users' personalities that can be updated independently.

## PRIMOR for Smart Systems Deployment

We designed and implemented PRIMOR to process CRUD operations in MARTs independently of the application domain. PRIMOR can be implemented in software engineering approaches using MARTs, such as systems' deployment, software development environments, or technical training. For example, the realization of smart applications deployment might use MARTs to define high-level configurations of the target deployment environment and procedures. Current deployment design-time specifications use UML, informal diagrams, or textual models. However, self-adaptive systems, the dynamic nature of the cloud and unexpected changes in the network, modify the execution environment dynamically. As a result, traditional deployment specifications become obsolete. With PRIMOR, changes in the deployment environment, specified as high-level MARTs, and through their causal connection, are propagated among the levels of abstraction, thus adapting the MARTs at execution time maintaining the information updated. Similarly, changes in the deployment specified in the MARTs, which will be taken by smart infrastructures to deploy a system.

## Software Engineering in CPHSs and Models at Runtime

As discussed by Lee [LS15], the CPS term differentiates from others in the sense that, unlike others, CPS is foundational and does not depend on an implementation approach, nor to a particular application. CPHSs adds the human component and the uncertainty that users' changing situations introduce to the system. CPHSs' research focuses on the intellectual challenges that arise when the cyber, physical and human worlds coexist. As described in Lee's definition, the development of CPSs (therefore CPHSs) requires an understanding of the interaction of the elements within the system. No doubt users will continue to demand integrated services, personalized features, and rely on their technology to achieve personal goals. Consequently, understanding and designing for uncertainty continue to be a milestone for software engineers that will remain relevant for the upcoming waves of technological changes.

Traditional software development takes into account all known information about the future system and its environments, such as components, interactions, data, users (and other stakeholders) and processes. However, unforeseeable changes in the users and the environment that affect the CPHSs applications described in this dissertation demonstrates a need for new paradigms of software engineering. Modern software development methods might have to consider execution time and all the uncertainty that will appear in the future production environment, to enable systems to remain available and relevant even when unexpected changes occur.

As uncertainty is unforeseeable, and predicting it opposes its reason to be, preparing software applications to deal with runtime uncertainty needs to enable the system with mechanisms to be aware, represent, understand, and act upon unknown situations that might affect its execution. In other words, systems should be aware and capable to not ignore unknown events and information before assessing them as irrelevant.

# Glossary

**Causal Links** Connections between two MARTs that will be affected in the case where one of them changes. Causal links are used by the MART management infrastructure to propagate changes.

**Cyber-Physical-Human System (CPHS)** The integration, mostly focused on the interactions, of cyber, physical and humans elements that work together towards the achievement of the objectives of the system [SSZ$^+$16, LS15].

**Galapagos Model** Our MART to represent users' evolving tasking goals, personal interactions, and the relevant contexts. It is specified using the *G-iStar* notation, our *i** extension for modelling UCSAs. [CVM14c].

**Galapagos Metamodel** Our MART that defines the concepts of UCSAs by abstracting the three dimensions of CPHSs as well as the smart interactions among them [CVM14c].

**Models at Runtime (MARTs)** Up-to-date representations about the system and its environment that can be manipulated and adapted at execution time [BBF09]. Also found in literature as *Models@run.time*, M@RT, Execution models, or Models at execution time.

**Operational Framework** Our framework defines model equivalences between human-readable and machine-readable, available runtime operations and semantics, to manage runtime operations on MARTs.

**Personal Tasking** An ordered sequence of tasks that assist users in the fulfilment of a personal goal. During personal tasking, users' context sources are exploited at runtime to understand changing situations and improve personalized functionalities [CVM13, CVM14b, CVM14c].

**Processing Infrastructure for Models at Runtime (PRIMOR)** Our infrastructure responsible for managing CRUD operations on MARTs. It is designed as a component-based system, implemented in Java, and designed to be extensible to other domains..

**Runtime adaptation** System's capability of performing adaptations to regulate their requirements satisfaction under changing conditions.

**Situation-aware self-adaptive system** A software system able to understand its environment to adapt itself with the goal of addressing changing requirements and context situations [Vil13].

**Smart internet** A new generation of the internet where web entities, represented by online services and content, are discovered, aggregated and delivered dynamically, automatically, and interactively according to users' needs and situations [NCCY10a].

**User-Centric Smart Cyber-Physical-Human Application** An orchestrated set of cyber, physical, and human components (along with their interconnections) that assist users in the fulfilment of their personal goals. A UCSA manages the smart interaction among the components dynamically, understands and acts upon users' changing situations, and has system capabilities to evolve at runtime.

**User-centric vision** Services and contents are dynamically and automatically composed of multiple sources to fit users' needs [NCCY10c].

# Acronyms

|  |  |
|---|---|
| **CPHS** | Cyber-Physical-Human System. |
| **MART** | Model at Runtime. |
| **PRIMOR** | Processing Infrastructure for Models at Runtime. |
| **UCSA** | User-Centric Smart Cyber-Physical-Human Application. |

# References[1]

[ABCF12] U. Aßmann, N. Bencomo, B. H. C. Cheng, and R. B. France. Models@run.time (Dagstuhl Seminar 11481). *Dagstuhl Reports*, 1(11):91–123, 2012. 27

[ADB+99] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles. Towards A Better Understanding of Context and Context-Awareness. In *Proceedings 1st International Symposium on Handheld and Ubiquitous Computing (HUC 1999)*, pages 304–307. Springer, 1999. 2, 48

[AGJ+14] U. Aßmann, S. Götz, J.-M. Jézéquel, B. Morin, and M. Trapp. A reference architecture and roadmap for Models@run.time systems. In *Models@run.time*, pages 1–18. Springer, 2014. x, 30, 31, 33, 36

[AP12] G. H. Alférez and V. Pelechano. Dynamic Evolution of Context-Aware Systems with Models at Runtime. In *Proceedings 15th International Conference on Model Driven Engineering Languages and Systems (MODELS 2012)*, volume 7590 of *Lecture Notes in Computer Science*, pages 70–86. Springer, 2012. 27

[BB13] N. Bencomo and A. Belaggoun. Supporting Decision-Making for Self-Adaptive Systems: From Goal Models to Dynamic Decision Networks. In *Requirements Engineering: Foundation for Software Quality*, volume 7830 of *Lecture Notes in Computer Science*, pages 221–236. Springer, 2013. 3

[BBF09] G. Blair, N. Bencomo, and R. B. France. Models@Run.Time. *IEEE Computer*, 42(10):22–27, 2009. 27, 157

[1] The numbers at the end of each bibliography item are the backward references to the pages where it was cited.

[BBG$^+$13] N. Bencomo, A. Bennaceur, P. Grace, G. Blair, and V. Issarny. The Role of Models@Run.Time in Supporting On-The-Fly Interoperability. *Computing*, 95(3):167–190, 2013. 3

[BBI13] N. Bencomo, A. Belaggoun, and V. Issarny. Dynamic Decision Networks for Decision-making in Self-adaptive Systems: A Case Study. In *Proceedings of the 8th ACM/IEEE International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2013)*, pages 113–122. IEEE, 2013. 3, 28

[BFT$^+$14] A. Bennaceur, R. France, G. Tamburrelli, T. Vogel, P. J. Mosterman, W. Cazzola, F. M. Costa, A. Pierantonio, M. Tichy, and M. Akşit. Mechanisms for leveraging models at runtime in self-adaptive software. In *Models@run.time*, pages 19–46. Springer, 2014. x, xi, 31, 32, 35, 63, 64, 72

[BGM09] V. Bryl, P. Giorgini, and J. Mylopoulos. Designing Socio-Technical Systems: From Stakeholder Goals To Social Networks. *Requirements Engineering*, 14(1):47–70, 2009. 44

[BM99] G. Bedny and D. Meister. Theory of Activity and Situation Awareness. *International Journal of Cognitive Ergonomics*, 3(1):63–72, 1999. x, 24, 26

[BM03] D. Bolchini and J. Mylopoulos. From Task-Oriented To Goal-Oriented Web Requirements Analysis. In *Proceedings 4th IEEE International Conference on Web Information Systems Engineering*, pages 166–175. IEEE, 2003. 38, 46, 71, 114, 116, 117

[Bré07] P. Brézillon. Context Modeling: Task Model and Practice Model. In *Modeling and Using Context*, volume 4635 of *Lecture Notes in Computer Science*, pages 122–135. Springer, 2007. 38, 46, 116, 117

[BTJ$^+$13] A. Bergen, N. Taherimakhsousi, P. Jain, L. Castañeda, and H. A. Müller. Dynamic Context Extraction in Personal Communication Applications. In *Proceedings Conference of The Center for Advanced Studies on Collaborative Research (CASCON 2013)*, pages 261–273. ACM, 2013. 12

[Car88] S. Carberry. Modeling The User's Plans and Goals. *Computing Linguistic*, 14(3):23–37, 1988. x, 38, 41, 42, 114

[Caz14] W. Cazzola. Evolution as Reflections on the Design. In *Models@run.time*, pages 259–278. Springer, 2014. 30

[CDGM10] A. K. Chopra, F. Dalpiaz, P. Giorgini, and J. Mylopoulos. Modeling and Reasoning About Service-Oriented Applications Via Goals and Commitments. In *Proceedings 22nd International Conference on Advanced Information Systems Engineering (CAISE 2010)*, pages 113–128. Springer, 2010. 38, 44, 114

[CE11] L. Coetzee and J. Eksteen. The Internet of Things - Promise for The Future? An Introduction. In *IST-Africa Conference Proceedings*, pages 1–9. IEEE, 2011. 3, 57

[CEG$^+$14] B. H. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, P. Pelliccione, A. Perini, N. A. Qureshi, B. Rumpe, D. Schneider, F. Trollmann, and N. M. Villegas. Using Models at Runtime to Address Assurance for Self-Adaptive Systems. In *Models@run.time*, volume 8378 of *Lecture Notes in Computer Science*, pages 101–136. Springer, 2014. 30

[CLG$^+$09] B. H. Cheng, R. Lemos, H. Giese, P. Inverardi, J. Magee, J. Andersson, B. Becker, N. Bencomo, Y. Brun, B. Cukic, G. Marzo Serugendo, S. Dustdar, A. Finkelstein, C. Gacek, K. Geihs, V. Grassi, G. Karsai, H. M. Kienle, J. Kramer, M. Litoiu, S. Malek, R. Mirandola, H. A. Müller, S. Park, M. Shaw, M. Tichy, M. Tivoli, D. Weyns, and J. Whittle. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-Adaptive Systems*, volume 5525 of *Lecture Notes in Computer Science*, pages 1–26. Springer, 2009. 2, 44, 59

[CLPS11] L. Console, I. Lombardi, C. Picardi, and R. Simeoni. Toward A Social Web of Intelligent Things. *AI Communications*, 24(3):265–279, 2011. 3, 57

[CLWG04] J. Cui, J. Liu, Y. Wu, and N. Gu. An Ontology Modeling Method in Semantic Composition of Web Services. In *Proceedings IEEE International Conference on E-Commerce Technology for Dynamic E-Business (CEC-EAST 2004)*, pages 270–273. IEEE, 2004. 38, 47, 116

[Cre13] J. W. Creswell. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications, 2013. 6

[CVM13] L. Castañeda, N. M. Villegas, and H. A. Müller. Towards Personalized Web-Tasking: Task Simplification Challenges. In *Proceedings 1st Workshop on Personalized Web-Tasking (PWT 2013) At Ninth IEEE World Congress on Services (SERVICES 2013)*, pages 147–153, 2013. 8, 12, 22, 45, 52, 66, 157

[CVM14a] L. Castañeda, N. M. Villegas, and H. A. Müller. Exploiting Social Context in Personalized Web-Tasking Applications. In *Proceedings 2014 Conference of The Center for Advanced Studies on Collaborative Research (CASCON 2014)*, pages 134–147. ACM, 2014. xi, 8, 12, 76

[CVM14b] L. Castañeda, N. M. Villegas, and H. A. Müller. Personalized Web-Tasking Applications: An Online Grocery Shopping Prototype. In *Proceedings 1st Workshop on Personalized Web-Tasking (PWT 2014) At Tenth IEEE World Congress on Services (SERVICES 2014)*, pages 24–29, 2014. 8, 12, 52, 157

[CVM14c] L. Castañeda, N. M. Villegas, and H. A. Müller. Self-Adaptive Applications: on The Development of Personalized Web-Tasking Systems. In *Proceedings ACM/IEEE 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2014)*, pages 49–54, 2014. xi, 12, 22, 52, 59, 71, 74, 157

[DFH16] F. Dalpiaz, X. Franch, and J. Horkoff. iStar 2.0 Language Guide. *arXiv preprint arXiv:1605.07767*, 2016. 77

[Dix08] A. Dix. Tasks = data + action + context: Automated task assistance through data-oriented analysis. In *Proceedings 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams*, (HCSE-TAMODIA 2008), pages 1–13. Springer, 2008. 39, 45, 71, 117, 119

[End95] M. R. Endsley. Toward A Theory of Situation Awareness in Dynamic Systems: Situation Awareness. *Human Factors*, 37(1):32–64, 1995. x, 2, 24, 25

[ESSD08] S. Easterbrook, J. Singer, M.-A. Storey, and D. Damian. Selecting Empirical Methods for Software Engineering Research. In *Guide To Advanced Empirical Software Engineering*, pages 285–311. Springer, 2008. 6

[GBP$^+$14] H. Giese, N. Bencomo, L. Pasquale, A. J. Ramirez, P. Inverardi, S. Wätzoldt, and S. Clarke. Living with uncertainty in the age of runtime models. In *Models@run.time*, pages 47–100. Springer, 2014. 30

[GBS08] S. Goschnick, S. Balbo, and L. Sonenberg. From Task To Agent-Oriented Meta-Models, and Back Again. In *Proceedings 2nd Conference on Human-Centered Software Engineering and 7th International Workshop on Task Models and Diagrams (HCSE-TAMODIA 2008)*, pages 41–57. Springer, 2008. 39, 114

[GFF$^+$07] M. Giersich, P. Forbrig, G. Fuchs, T. Kirste, D. Reichart, and H. Schumann. Towards An Integrated Approach for Task Modeling and Human Behavior Recognition. In *Proceedings 12th International Conference on Human-Computer Interaction: Interaction, Design and Usability (HCI 2007)*, pages 1109–1118. Springer, 2007. 39, 45, 116, 117

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley, 1995. 124

[GKV$^+$07] M. Golemati, A. Katifori, C. Vassilakis, G. Lepouras, and C. Halatsis. Creating An Ontology for The User Profile: Method and Applications. In *Proceedings 1st International Conference on Research Challenges in Information Science*, (RCIS 2007), pages 407–412, 2007. 49

[Gor11] I. Gorton. *Software Quality Attributes*, pages 23–38. Springer, Berlin, Heidelberg, 2011. 113, 120, 142

[GSB$^+$08] H. Goldsby, P. Sawyer, N. Bencomo, B. Cheng, and D. Hughes. Goal-Based Modeling of Dynamically Adaptive System Requirements. In *15th*

*IEEE Annual International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2008)*, pages 36–45. IEEE, 2008. 27

[JBCM13] P. Jain, A. Bergen, L. Castañeda, and H. A. Müller. PALTask chat: A personalized automated context aware web resources listing tool. In *Proceedings 1st Workshop on Personalized Web-Tasking (PWT 2013) At Ninth IEEE World Congress on Services (SERVICES 2013)*, pages 154–157. IEEE, 2013. 12

[JCP08] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. Reporting Experiments in Software Engineering. In *Guide To Advanced Empirical Software Engineering*, pages 201–228. Springer, 2008. 6

[JHG13] E. Y. Jennifer Horkoff and G. Grau. IStar Guide. Online: http://istar.rwth-aachen.de/tiki-index.php?page=istarquickguide. Oct. 2013, 2013. 77

[JVM$^+$02] B. John, A. Vera, M. Matessa, M. Freed, and R. Remington. Automating CPM-GOMS. In *Proceedings SIGCHI Conference on Human Factors in Computing Systems*, (CHI 2002), pages 147–154, 2002. 39, 48, 117

[KC07] B. Kitchenham and S. Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering. Technical Report EBSE 2007-001, Keele University and Durham University Joint Report, 2007. 6

[KFM$^+$13] K. Kenda, C. Fortuna, A. Moraru, D. Mladenić, B. Fortuna, and M. Grobelnik. Mashups for The Web of Things. In B. Endres-Niggemeyer, editor, *Semantic Mashups*, pages 145–169. Springer, 2013. 3, 57

[KK05] T. Klug and J. Kangasharju. Executable Task Models. In *Proceedings 4th International Workshop on Task Models and Diagrams (TAMODIA 2005)*, pages 119–122. ACM, 2005. 39, 46, 116, 117

[KKJ13] H. Kim, S. I. Kim, and W. Jung. Ontology Modelling for REST Open Apis and Web Service Mash-Up Method. In *Proceedings 2013 International Conference on Information Networking (ICOIN 2013)*, pages 523–528. IEEE Computer Society, 2013. 39, 47

[KM06] S. K. Khaitan and J. D. McCalley. Design Techniques and Applications of Cyberphysical Systems: A Survey. *IEEE Systems Journal*, 9(2):350–365, 2015-06. 18

[KPBB+09] B. Kitchenham, O. Pearl Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic Literature Reviews in Software Engineering - A Systematic Literature Review. *Information and Software Technology*, 51(1):7–15, 2009. 13, 38, 50

[KVD+08] A. Katifori, C. Vassilakis, I. Daradimos, G. Lepouras, Y. Ioannidis, A. Dix, A. Poggi, and T. Catarci. Personal Ontology Creation and Visualization for A Personal Interaction Management System. In *Proceedings of Personal Information Management Workshop*, (CHI 2008), 2008. 49

[Lee10] E. A. Lee. CPS Foundations. In *Proceedings of the 47th Design Automation Conference (DAC 2010*, pages 737–742, New York, NY, USA, 2010. ACM. 1, 17, 18

[LGM+13] R. Lemos, H. Giese, H. A. Müller, M. Shaw, J. Andersson, M. Litoiu, B. Schmerl, G. Tamura, N. M. Villegas, T. Vogel, D. Weyns, L. Baresi, B. Becker, N. Bencomo, Y. Brun, B. Cukic, R. Desmarais, S. Dustdar, G. Engels, K. Geihs, K. M. GÖschka, A. Gorla, V. Grassi, P. Inverardi, G. Karsai, J. Kramer, A. Lopes, J. Magee, S. Malek, S. Mankovskii, R. Mirandola, J. Mylopoulos, O. Nierstrasz, M. PezzÈ, C. Prehofer, W. SchÄfer, R. Schlichting, D. B. Smith, J. A. P. Sousa, L. Tahvildari, K. Wong, and J. Wuttke. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 1–32. Springer, 2013. 2, 59

[LLJM11] S. Liaskos, M. Litoiu, M. D. Jungblut, and J. Mylopoulos. Goal-Based Behavioral Customization of Information Systems. In *Proceedings 23rd International Conference on Advanced Information Systems Engineering (CAISE 2011)*, pages 77–92. Springer, 2011. x, 39, 44, 50, 71, 115, 117

[LMSM10] S. Liaskos, S. A. Mcilraith, S. Sohrabi, and J. Mylopoulos. Integrating Preferences Into Goal Models for Requirements Engineering. In *Pro-*

*ceedings 18th IEEE International, Requirements Engineering Conference (RE 2010)*, pages 135–144, 2010. 39, 44, 50, 71, 115, 117

[LMSM11] S. Liaskos, S. A. Mcilraith, S. Sohrabi, and J. Mylopoulos. Representing and Reasoning About Preferences in Requirements Engineering. *Requirements Engineering*, 16(3):227–249, 2011. 71

[LS15] E. A. Lee and S. A. Seshia. *Introduction to Embedded Systems, A Cyber-Physical Systems Approach, Second Edition*. LeeSeshia.org, 2015. 1, 17, 156, 157

[MBJ+09] B. Morin, O. Barais, J.-M. Jezequel, F. Fleurey, and A. Solberg. Models@Run.Time To Support Dynamic Adaptation. *IEEE Computer*, 42(10):44–51, 2009. 28

[MCY99] J. Mylopoulos, L. Chung, and E. Yu. From Object-Oriented To Goal-Oriented Requirements Analysis. *Communications of the ACM*, 42(1):31–37, 1999. 39, 43, 44, 50, 71, 115, 117

[MFKC+17] J. Muñoz-Fernández, A. Knauss, L. Castañeda, M. Derakhshanmanesh, R. Heinrich, M. Becker, and N. Taherimakhsousi. Capturing Ambiguity in Artifacts to Support Requirements Engineering for Self-Adaptive Systems. In *RESACS: 3rd International Workshop on Requirements Engineering for Self-Adaptive & Cyber Physical System*, 2017. 12

[MKS09] H. A. Müller, H. M. Kienle, and U. Stege. Autonomic Computing: Now You See It, Now You Don't. Design and Evolution of Autonomic Software Systems. In *Lecture Notes in Computer Science*, volume 5413, pages 32–54. Springer, 2009. 2, 59

[MMS15] I. Malavolta, H. Muccini, and M. Sharaf. A Preliminary Study on Architecting Cyber-Physical Systems. In *Proceedings of the 2015 European Conference on Software Architecture Workshops (ECSAW 2015)*, pages 20:1–20:6, New York, NY, USA, 2015. ACM. 18

[MNS+13] H. A. Müller, J. Ng, E. Stroulia, K. Kontogiannis, N. M. Villegas, and D. Lau. Personalized Web-Tasking Workshop Organizers' Message. In *Proceedings 1st Workshop on Personalized Web-Tasking (PWT 2013) at*

*Ninth IEEE World Congress on Services (SERVICES 2013)*, page xvii, 2013. 22

[MSW16] H. Muccini, M. Sharaf, and D. Weyns. Self-adaptation for Cyber-physical Systems: A Systematic Literature Review. In *Proceedings of the 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, SEAMS 2016, pages 75–81, New York, NY, USA, 2016. ACM. 3

[MV13] H. Müller and N. Villegas. Runtime Evolution of Highly Dynamic Software. In *Evolving Software Systems*, pages 229–264. Springer, 2013. 3, 27, 30, 58

[NCCY10a] J. W. Ng, M. Chignell, J. R. Cordy, and Y. Yesha. Overview of The Smart Internet. In *The Smart Internet*, pages 49–56. Springer, 2010. 20, 158

[NCCY10b] J. W. Ng, M. Chignell, J. R. Cordy, and Y. Yesha. Smart Interactions. In *The Smart Internet*, pages 59–64. Springer, 2010. 17, 20, 22

[NCCY10c] J. W. Ng, M. Chignell, J. R. Cordy, and Y. Yesha. *The Smart Internet*. Springer, 2010. 20, 158

[NFG⁺06] L. Northrop, P. Feiler, R. P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, D. Schmidtd, K. Sullivan, and K. Wallnau. *Ultra-Large-Scale Systems: The Software Challenge of The Future.* Carnegie Mellon University, 2006. http://www..Sei.Cmu.Edu/Uls/. Published: June 2006. (Accessed: May 2013). 1

[Ng10] J. W. Ng. The Personal Web: Smart Internet for Me. In *Proceedings 2010 Conference of The Center for Advanced Studies on Collaborative Research (CASCON 2010)*, pages 330–344. ACM, 2010. 20

[Nie93] J. Nielsen. *Usability Engineering.* Morgan Kaufmann, 1993. 132, 142

[PZCG13] C. Perera, A. B. Zaslavsky, P. Christen, and D. Georgakopoulos. Context Aware Computing for The Internet of Things: A Survey. *IEEE Communications Surveys & Tutorials*, 16(1):414–454, 2013. 3, 57

[RCBS12] A. J. Ramirez, B. H. Cheng, N. Bencomo, and P. Sawyer. Relaxing Claims: Coping with Uncertainty While Evaluating Assumptions at Run Time. In *Model Driven Engineering Languages and Systems*, volume 7590 of *Lecture Notes in Computer Science*, pages 53–69. Springer, 2012. 27

[SBCC13] H. Song, S. Barrett, A. Clarke, and S. Clarke. Self-adaptation with End-User Preferences: Using Run-Time Models and Constraint Solving. In *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 555–571. Springer, 2013. 27

[SHC$^+$05] H. Song, G. Huang, F. Chauvel, Y. Sun, and H. Mei. SM@RT: representing run-time system data as MOF-compliant models. In *Proceedings 32nd International Conference on Software Engineering*, volume 2, pages 303–304. ACM/IEEE, 2010-05. 27

[SLV02] N. Souchon, Q. Limbourg, and J. Vanderdonkt. Task Modelling in Multiple Contexts of Use. In *Proceedings 9th International Workshop on Interactive Systems: Design, Specification, and Verification (DSV-IS 2002)*, pages 59–73. Springer, 2002. 40, 46, 71, 116, 118, 119

[SS08] T. Stoitsev and S. Scheidl. A Method for Modeling Interactions on Task Representations in Business Task Management Systems. In *Engineering Interactive Systems*, volume 5247 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2008. 40, 48, 116

[SSLRM11] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness Requirements for Adaptive Systems. In *Proceedings of the 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, (SEAMS 2011), pages 60–69. ACM, 2011. 27

[SSZ$^+$16] S. K. Sowe, E. Simmon, K. Zettsu, F. de Vaulx, and I. Bojanova. Cyber-Physical-Human Systems: Putting People in the Loop. *IT Professional*, 18(1):10–13, Jan 2016. 1, 2, 17, 18, 145, 157

[SWYS11] J. Shi, J. Wan, H. Yan, and H. Suo. A survey of Cyber-Physical Systems. In *2011 International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2011. 18

[SZ13] M. Szvetits and U. Zdun. Systematic literature review of the objectives, techniques, kinds, and architectures of models at runtime. *Software & Systems Modeling*, pages 1–39, 2013. 27

[SZF$^+$14] H. Song, X. Zhang, N. Ferry, F. Chauvel, A. Solberg, and G. Huang. Modelling Adaptation Policies as Domain-Specific Constraints. In *Model-Driven Engineering Languages and Systems*, volume 8767 of *Lecture Notes in Computer Science*, pages 269–285. Springer, 2014. 27

[TLI$^+$11] L. H. Thom, I. M. Lazarte, C. Iochpe, L.-M. Priego, C. Verdier, O. Chiotti, and P. D. Villarreal. on the Capabilities of BPMN for Workflow Activity Patterns Representation. In *Business Process Model and Notation*, volume 95 of *Lecture Notes in Business Information Processing*, pages 172–177. Springer, 2011. 40, 48, 118

[TS14] M. Trapp and D. Schneider. Safety assurance of open adaptive systems–a survey. In *Models@run.time*, pages 279–318. Springer, 2014. 30

[Vil13] N. M. Villegas. *Context Management and Self-Adaptivity for Situation-Aware Smart Software Systems*. PhD Thesis, Department of Computer Science, University of Victoria, 2013. x, 7, 8, 25, 40, 48, 49, 50, 60, 71, 76, 118, 119, 122, 154, 158, 180, 183

[VM10] N. M. Villegas and H. A. Müller. Managing Dynamic Context To Optimize Smart Interactions and Services. In *The Smart Internet*, volume 6400 of *Lecture Notes in Computer Science*, pages 289–318. Springer, 2010. 49

[VTM$^+$13] N. M. Villegas, G. Tamura, H. A. Müller, L. Duchien, and R. Casallas. DYNAMICO: A reference model for governing control objectives and context relevance in self-adaptive software systems. In *Software Engineering for Self-Adaptive Systems II*, volume 7475 of *Lecture Notes in Computer Science*, pages 265–293. Springer, 2013. 8, 10, 59, 148

[Wil99] T. D. Wilson. Models in Information Behaviour Research. *Journal of Documentation*, 55(3):249–270, 1999. x, 40, 47, 115, 118

[YLL$^+$08] Y. Yu, A. Lapouchnian, S. Liaskos, J. Mylopoulos, and J. C. S. P. Leite. From Goals To High-Variability Software Design. In *Proceedings 17th*

*International Conference on Foundations of Intelligent Systems (ISMIS 2008)*, pages 1–16. Springer, 2008. x, 40, 44, 45, 118

[YM94] E. Yu and J. Mylopoulos. Understanding ”Why” in Software Process Modelling, Analysis, and Design. In *Proceedings 16th International Conference on Software Engineering (ICSE 1994)*, pages 159–168. IEEE Computer Society Press, 1994. x, 40, 42, 43, 71, 115, 117, 119

[Yu93] E. Yu. Modeling Organizations for Information Systems Requirements Engineering. In *Proceedings IEEE International Symposium on Requirements Engineering (RE 1993)*, pages 34–41, 1993. 40, 41, 50, 71, 115, 117

[ZCZ$^+$13] X. Zhang, X. Chen, Y. Zhang, Y. Wu, W. Yao, G. Huang, and Q. Lin. Runtime Model Based Management of Diverse Cloud Resources. In *Model-Driven Engineering Languages and Systems*, volume 8107 of *Lecture Notes in Computer Science*, pages 572–588. Springer, 2013. 27

# Appendix A

# MART Specifications

This appendix presents the specification of our two Models at Runtime (MARTs) for User-Centric Smart Cyber-Physical-Human Applications (UCSAs).

## A.1 Galapagos Metamodel

**Table A.1:** Galapagos Metamodel Specification

**Entity**

| **Attribute** | **Type** | **Description** |
|---|---|---|

***All Entities***

| | | |
|---|---|---|
| name [unique] | String | Identifier of the element. It is unique for the model instance. |
| description [required] | String | Long name given to the element. Used in the notation form as the name. |
| locked [required] | Boolean | Defines if the element is locked for modification. (Default: `false`) |

***Situation***

| | | |
|---|---|---|
| validFrom | Date | Defines the starting time of the `Situation`. |
| validTo | Date | Defines the starting time of the `Situation`. |

***Plan Item***

**Entity**

| **Attribute** | **Type** | **Description** |
|---|---|---|
| orderNo | Int | Defines the number in the sequence of items to be executed in the plan. |

***Condition***

| | | |
|---|---|---|
| predicate | String | Describes the condition as a predicate string. |

***Satisfaction Property***

| | | |
|---|---|---|
| threshold | String | Describes a threshold for the `Satisfaction Property`. |

***Activity***

| | | |
|---|---|---|
| expression | String | Describes the set of instructions to describe activities |

***Data***

| | | |
|---|---|---|
| typeValue | String | Determines the data type for the `Data` |

## A.2 Galapagos Model

**Table A.2:** Galapagos Model Specification

**Element**

| **Attribute** | **Type** | **Description** |
|---|---|---|

***All Elements***

| | | |
|---|---|---|
| Name [unique] | String | Identifier of the element. It is unique for the model instance. |
| Description [required] | String | Long name given to the element. Used in the notation form as the name. |
| Locked [required] | Boolean | Defines if the element is locked for modification. (Default: `false`) |

**Element**

| **Attribute** | **Type** | **Description** |
|---|---|---|

**Goal**

| | | |
|---|---|---|
| Subgoals | Array | List of `Goal`. Refers to a decomposition relation. |
| Task Sequences [required] | Array | List of root tasks that denote the sequence (or sequences). |
| Measurable Outcomes [required] | Array | Each position of the array contains the following information: `Information Resource`, `validation` rules, and `threshold` of acceptance |
| Execution Conditions [required] | Array | Defines policies for the execution of the task sequence. |

**Task**

| | | |
|---|---|---|
| Sequence Number | Int | Number in the sequence of execution |
| Preconditions | Array | Set of rules that need to be valid all the time. |
| Postconditions | Array | Set of activities that need to be executed after the `Task` changes. |
| Inputs | Array | Set of information required as input for the execution of the task. |
| Outputs | Array | Set of `Information Resource` resulting after the execution of the task. |
| Children | Array | List of `Task`. Refers to a decomposition relation. |
| Dependences | Array | List of `Task`, `Goal`, or `Information Resource` that the `Task` depends on. |
| Execution Date | String, Object | Defines the time dimension for the execution of the `Task`. Values: `withGoal, auto, specific(Date), withSequence`. |
| Execution Policies | Array | Defines specific policies for execution of the task. |

**Element**

| Attribute | Type | Description |
|---|---|---|
| Execution Constrains | Array | Defines specific constrains for execution of the task. |
| Operation Type | String, Object | Defines how the execution starts. Values: `withGoal`, `auto`, `specific(Date)`, `withSequence`. |
| Termination | String, Object | Defines how the `Task` ends. Values: `withGoal`, `auto`, `specific(Date)`, `withSequence`, `timeout(milliseconds)`. |
| Activities | Array | |

**Information Resource**

| | | |
|---|---|---|
| Type | String | Defines the type of Resource. Values: `webservice`, `app`, `file`, `database`, `event`, `message`, `unspecified`. |
| Location | String | Location of the resource (URI, path). |
| Resource name | String | Name of the resource in location. |
| Authorization request | Boolean | Defines whether the resource requires authorization to access. (Default: `false`) |
| Credential | Object | Defines an object that contains the credentials for authentication to access the resource. |

**Actor**

| | | |
|---|---|---|
| Location | String | Defines the locations of the `Actor` |
| Authorization level | String | Defines the level of authority regarding its responsibility |
| Situation policies | Array | Defines conditions and policies associated with situations of the user and relevant goals |

# Appendix B

# SUSGroceries Case Scenario Implementation

SUSGroceries is our User-Centric Smart Cyber-Physical-Human Application client to manage the personal tasking of the user that achieves an online grocery shopping goal. Figure B.1 depicts the overview of SUSGroceries and Primor. As the client application, SUSGroceries interacts with the supporting infrastructure, which is a service in the cloud, to request access and adaptations to the MARTs.

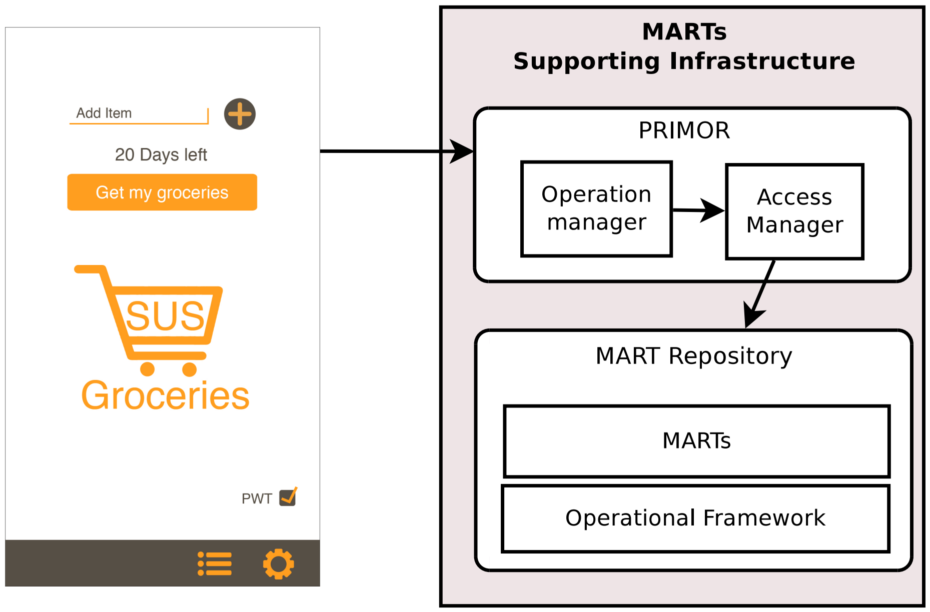
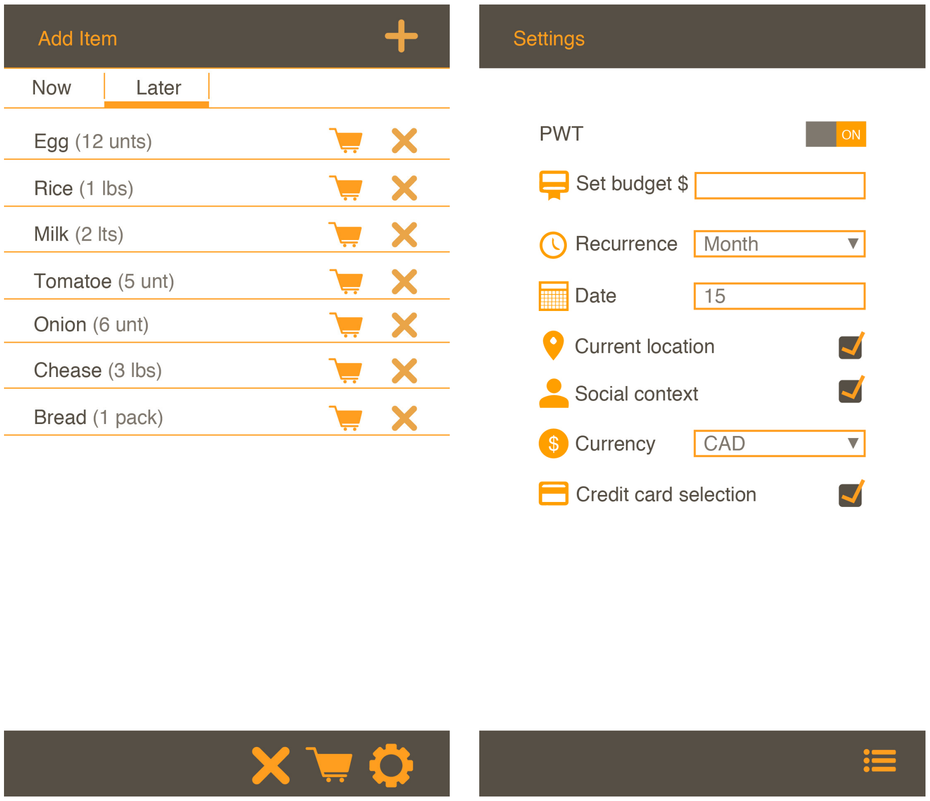


**Figure B.1:** SUSGroceries request services to Primor

SUSGroceries is a mobile application that manages the grocery list as well as general settings of the user. We use SUSGroceries to simulate context information such as location and user preferences. Figure B.2 shows two screen shots of the mobile application. The left of the figure shows the grocery list, which is a set of items that have been added. In our implementation we simulated sensors and actuators of an instrumented home that populates the list. The right of the figure shows the application settings, which allows to specify some information about the personal tasking including budget, recurrence of the task and enable/disable location.



| | |
|---|---|
| Egg (12 unts) | |
| Rice (1 lbs) | |
| Milk (2 lts) | |
| Tomatoe (5 unt) | |
| Onion (6 unt) | |
| Chease (3 lbs) | |
| Bread (1 pack) | |

| | |
|---|---|
| PWT | ON |
| Set budget $ | |
| Recurrence | Month |
| Date | 15 |
| Current location | ✓ |
| Social context | ✓ |
| Currency | CAD |
| Credit card selection | ✓ |

**Figure B.2:** SUSGroceries interface to manage the grocery shopping and user settings

## B.1 Object-Oriented Implementation of MARTs

We follow the object-oriented paradigm for our implementation to provide the advantages of modularity and reusability. Figure B.3 depicts the classes for our implementation of MARTs as graphs. The class `GoalsModel` implements the interface `MART`. a `MART` contains an artefact, which in the case of our model corresponds to a graph. Therefore, the class `UCSAGoalGraph` implements the interface `Artefact` and contains an object `Graph`. Since MARTs define a set of operations, the class `AbstractOperation` defines the set of supported operations of the MART listed in an enumerator object named `GoalsModelOp`. For this implementation we defined a selection of operations supported by our model, including `ADD_GOAL, DELETE_TASK`, and `UPDATE_GOAL`.

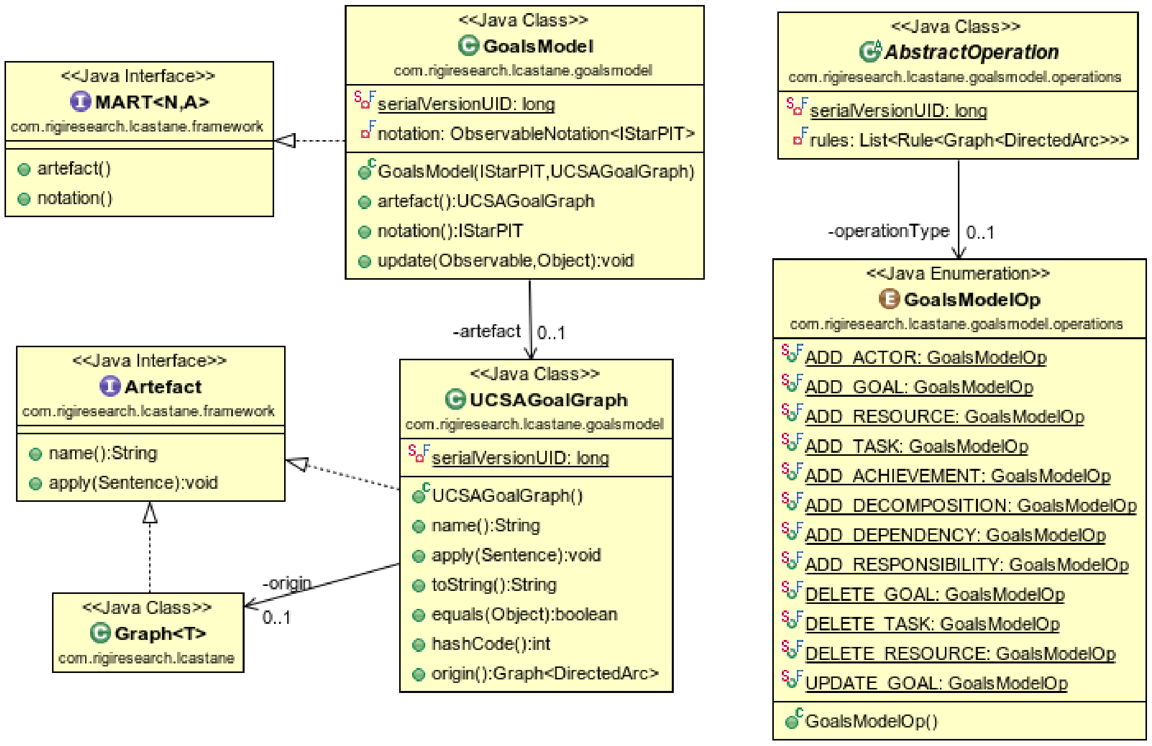

<<Java Interface>>
MART<N,A>
com.rigiresearch.lcastane.framework
artefact()
notation()
<<Java Class>>
GoalsModel
com.rigiresearch.lcastane.goalsmodel
serialVersionUID: long
notation: ObservableNotation<IStarPIT>
GoalsModel(IStarPIT,UCSAGoalGraph)
artefact():UCSAGoalGraph
notation():IStarPIT
update(Observable,Object):void
<<Java Class>>
AbstractOperation
com.rigiresearch.lcastane.goalsmodel.operations
serialVersionUID: long
rules: List<Rule<Graph<DirectedArc>>>
-operationType
0..1
<<Java Enumeration>>
GoalsModelOp
com.rigiresearch.lcastane.goalsmodel.operations
ADD_ACTOR: GoalsModelOp
ADD_GOAL: GoalsModelOp
ADD_RESOURCE: GoalsModelOp
ADD_TASK: GoalsModelOp
ADD_ACHIEVEMENT: GoalsModelOp
ADD_DECOMPOSITION: GoalsModelOp
ADD_DEPENDENCY: GoalsModelOp
ADD_RESPONSIBILITY: GoalsModelOp
DELETE_GOAL: GoalsModelOp
DELETE_TASK: GoalsModelOp
DELETE_RESOURCE: GoalsModelOp
UPDATE_GOAL: GoalsModelOp
GoalsModelOp()
-artefact
0..1
<<Java Interface>>
Artefact
com.rigiresearch.lcastane.framework
name():String
apply(Sentence):void
<<Java Class>>
UCSAGoalGraph
com.rigiresearch.lcastane.goalsmodel
serialVersionUID: long
UCSAGoalGraph()
name():String
apply(Sentence):void
toString():String
equals(Object):boolean
hashCode():int
origin():Graph<DirectedArc>
-origin
0..1
<<Java Class>>
Graph<T>
com.rigiresearch.lcastane

**Figure B.3:** Java classes of our GALAPAGOS MODEL

We based our implementation in the example presented in Chapter 5 (cf. Figures 5.13),. Source Code B.1 depicts a simplified view of the code that creates the graph based on the model specified with our notation. For our implementation, we simulated the translation of a notation into Java classes. In Line 3 the object type `UCSAGoalGraph` is initialise, which will contain nodes and arcs based on the MART specification. Lines 5 to 17 present examples of creating `Node` objects, such as `Goal, Actor, Task`, and `Resource`. Lines 20 to 31 present examples of adding the nodes to the graph and creating the arcs based on the relations. It is important to notice that the creation of the graph is made using the operational framework using sentences. For instance, Line 20 adds a goal to the graph by applying the operation `ADD_GOAL` defined in the operational framework through the enumeration `GoalsModelOp`. Line 23 creates the connection of responsibility between SUSGROCERIES which is an `Actor` and the main goal. Lines 26 and 27 add the first `Task` of the model and the relation between the `Task` and the `Goal` is an achievement connection. Finally, Lines 31 and 31 create a relation between two elements `Task` through a relation of decomposition.

**Source Code B.1:** Simplified view of the MART instance of the GALAPAGOS MODEL depicted in Figure 6.8

```
 1 public static UCSAGoalGraph artefact() throws
       ValidationException {
 2
 3   final UCSAGoalGraph graph = new UCSAGoalGraph();
 4
 5   Goal goal = new Goal("G_grocery_shopping", "Online grocery
        shopping");
 6   Actor SUSGroceries = new Actor("A_SUSGroceries",
        "SUSGroceries");
 7   Task shopGroceryTask = new Task("T_shop_grocery_task", "Shop
        grocery task");
 8   Resource listItemsResource = null;
 9   try {
10     listItemsResource = new Resource(
11       "R_list_items",
12       "WebService",
13       new URL("http://primor/list"),
14       "list",
15       true
16     );
17   } catch (MalformedURLException e) {}
18
19   // Online Grocery Shopping
20   graph.apply(new Sentence(GoalsModelOp.ADD_GOAL, goal));
21
22   // SUSGroceries, SUSGroceries --r--> Online Grocery Shopping
23   graph.apply(new Sentence(GoalsModelOp.ADD_ACTOR, SUSGroceries,
        goal));
24
25   // Shop Grocery Task, Shop Grocery Task --a--> Online Grocery
        Shopping
26   graph.apply(new Sentence(GoalsModelOp.ADD_TASK,
        shopGroceryTask, goal));
27   graph.apply(new Sentence(GoalsModelOp.ADD_ACHIEVEMENT,
        shopGroceryTask, goal));
28
```

```
    // Get grocery items list, Shop grocery task --dc--> Get
       grocery items list
    graph.apply(new Sentence(GoalsModelOp.ADD_TASK,
       groceryListTask, shopGroceryTask));
    graph.apply(new Sentence(GoalsModelOp.ADD_DECOMPOSITION,
       shopGroceryTask, groceryListTask));

    /*...*/

    return graph;
}
```

## B.2 SmarterContext Ontology Extension and Vocabulary

We use the Context Ontology by Villegas [Vil13] with the following extension in the Shopping Ontology. We added the class `Grocery` and its subclasses depicted in Figure B.4.

We use OWL (Web Ontology Language) to represent the ontology for the context model. Specifically, we use the RDF (Resource Description Framework) Schema (RDFS) triples denote subject-predicate-object expressions. The subject defines a resource, the predicate denotes aspects of the resource as well as the relationship between the subject and the object.

**Table B.1:** Vocabulary for the grocery shopping scenario

| **Class** | **Vocabulary** |
|---|---|
| Vegetable Fruits | Tomato, Orange, Strawberry, Celery, Lettuce, Potato |
| Beverages | Coffee, Tea, Juice, Soda, |
| Bread Bakery | Tortilla, Bagel, Bread |
| Canned Jarred | Ketchup, Spaghetti Sauce, Beans |
| Dairy | Eggs, Milk, Cheeses, Butter |
| Frozen | Waffles, Ice Cream, Meals, Vegetables |
| Meat | Lunch Meat, Poultry, Beef, Pork, Fish |
| Cleaners | All Purpose, Laundry Soap, Dishwasher, Hands soap |

| | |
|---|---|
| Paper | Paper Towels, Toilet Paper, Sandwich Bags, Aluminum Foil |
| Personal | Shampoo, Soap, Shaving Cream, Hand Lotion |
| Baby | Baby Food, Diapers, Powder, Wet Wipes |

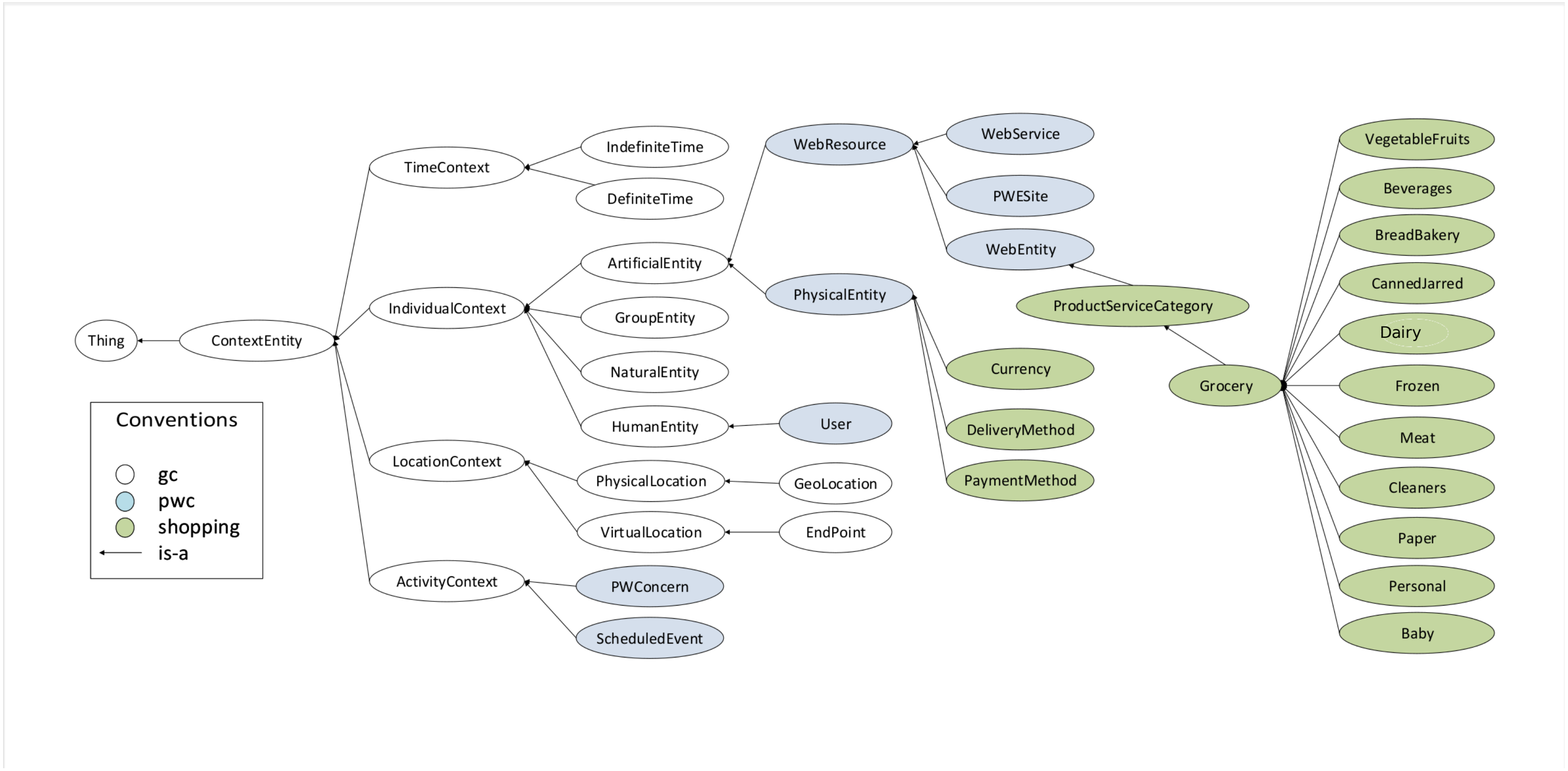


**Figure B.4:** Subset of the SmarteContext ontology extension for the online grocery shopping scenario.

## B.3 Personal Context Sphere Instance

This section presents a a simplified PCS for user Edel in our online grocery shopping case study following the PCS definition by Villegas [Vil13]. Each triple in Table B.2 below represents an RDF statement, contextual facts in SmarterContext, defined by a subject, a predicate, and an object.

**Table B.2:** RDF triples that define the personal context sphere for user Edel

| # | Subject | Predicate | Object |
|---|---|---|---|
| 1 | geo:Victoria | rdf:type | gc:GeoLocation |
| 2 | http://www.walmart.ca/en | rdf:type | pwc:PWESite |
| 3 | https://shop.saveonfoods.com | rdf:type | pwc:PWESite |
| 4 | https://www.thriftyfoods.com | rdf:type | pwc:PWESite |
| 5 | Activia Yogurt | rdf:type | shopping:Dairy Category |
| 6 | Homogenized Milk | rdf:type | shopping:dairy Category |
| 7 | Roma Tomatoes | rdf:type | shopping:VegetablesFruits Category |
| 8 | Salmon | rdf:type | shopping:Meat Category |
| 9 | CAD | rdf:type | shopping:Currency Category |
| 10 | Week night | rdf:type | shopping:DeliveryMethod Category |
| 11 | VISA**1100 | rdf:type | shopping:PaymentMethod Category |
| 12 | edel.rdf#edel | pwc:isInterestedIn | deals:Produce_&_Dairy |
| 13 | edel.rdf#edel | pwc:hasIntegrated | http://www.walmart.ca/en |

| | | | |
|---|---|---|---|
| 14 | edel.rdf#edel | pwc:hasIntegrated | https://shop.saveonfoods.co |
| 15 | edel.rdf#edel | pwc:hasIntegrated | https://www.thriftyfoods.com |
| 16 | http://www.walmart.ca/ena | gc:locatedIn | geo:Victoria |
| 17 | https://shop.saveonfoods.co | gc:locatedIn | geo:Victoria |
| 18 | https://www.thriftyfoods.com | gc:locatedIn | geo:Victoria |
| 19 | edel.rdf#edel | shopping:toBuy | shopping:Grocery Category |
| 20 | edel.rdf#edel | gc:locatedIn | geo:Victoria |
| 21 | edel.rdf#edel | rdf:type | pwc:User |
| 22 | edel.rdf#edel | pwc:likes | https://www.thriftyfoods.com |

I wanted to push the boundaries of knowledge.
Here is my push, might not be the last.