OULU 2014
C 499

ACTA UNIVERSITATIS OULUENSIS

Lauri Tuovinen

FROM MACHINE LEARNING TO LEARNING WITH MACHINES

REMODELING THE KNOWLEDGE DISCOVERY PROCESS

UNIVERSITY OF OULU GRADUATE SCHOOL;
UNIVERSITY OF OULU,
FACULTY OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING,
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING;
UNIVERSITY OF OULU, INFOTECH OULU

ACTA UNIVERSITATIS OULUENSIS
C Technica 499

LAURI TUOVINEN

FROM MACHINE LEARNING TO LEARNING WITH MACHINES

Remodeling the knowledge discovery process

Academic dissertation to be presented with the assent of the Doctoral Training Committee of Technology and Natural Sciences of the University of Oulu for public defence in Auditorium TS101, Linnanmaa, on 29 August 2014, at 12 noon

UNIVERSITY OF OULU, OULU 2014

Copyright © 2014
Acta Univ. Oul. C 499, 2014

Supervised by
Professor Juha Röning

Reviewed by
Professor Christopher Clifton
Professor Vassilios Verykios

Opponent
Professor Alan Smeaton

ISBN 978-952-62-0523-6 (Paperback)
ISBN 978-952-62-0524-3 (PDF)

ISSN 0355-3213 (Printed)
ISSN 1796-2226 (Online)

Cover Design
Raimo Ahonen

JUVENES PRINT
TAMPERE 2014

Tuovinen, Lauri, From machine learning to learning with machines. Remodeling the knowledge discovery process

University of Oulu Graduate School; University of Oulu, Faculty of Information Technology and Electrical Engineering, Department of Computer Science and Engineering; University of Oulu, Infotech Oulu

Acta Univ. Oul. C 499, 2014

University of Oulu, P.O. Box 8000, FI-90014 University of Oulu, Finland

Abstract

Knowledge discovery (KD) technology is used to extract knowledge from large quantities of digital data in an automated fashion. The established process model represents the KD process in a linear and technology-centered manner, as a sequence of transformations that refine raw data into more and more abstract and distilled representations. Any actual KD process, however, has aspects that are not adequately covered by this model. In particular, some of the most important actors in the process are not technological but human, and the operations associated with these actors are interactive rather than sequential in nature. This thesis proposes an augmentation of the established model that addresses this neglected dimension of the KD process.

The proposed process model is composed of three sub-models: a data model, a workflow model, and an architectural model. Each sub-model views the KD process from a different angle: the data model examines the process from the perspective of different states of data and transformations that convert data from one state to another, the workflow model describes the actors of the process and the interactions between them, and the architectural model guides the design of software for the execution of the process. For each of the sub-models, the thesis first defines a set of requirements, then presents the solution designed to satisfy the requirements, and finally, re-examines the requirements to show how they are accounted for by the solution.

The principal contribution of the thesis is a broader perspective on the KD process than what is currently the mainstream view. The augmented KD process model proposed by the thesis makes use of the established model, but expands it by gathering data management and knowledge representation, KD workflow and software architecture under a single unified model. Furthermore, the proposed model considers issues that are usually either overlooked or treated as separate from the KD process, such as the philosophical aspect of KD. The thesis also discusses a number of technical solutions to individual sub-problems of the KD process, including two software frameworks and four case-study applications that serve as concrete implementations and illustrations of several key features of the proposed process model.

Keywords: data management, knowledge discovery, knowledge representation, process model, software architecture, workflow management

Tuovinen, Lauri, Koneoppimisesta koneilla oppimiseen. Uudenlainen malli tiedonlouhintaprosessille

Oulun yliopiston tutkijakoulu; Oulun yliopisto, Tieto- ja sähkötekniikan tiedekunta, Tietotekniikan osasto; Oulun yliopisto, Infotech Oulu

Acta Univ. Oul. C 499, 2014

Oulun yliopisto, PL 8000, 90014 Oulun yliopisto

Tiivistelmä

Tiedonlouhintateknologialla etsitään automoidusti tietoa suurista määristä digitaalista dataa. Vakiintunut prosessimalli kuvaa tiedonlouhintaprosessia lineaarisesti ja teknologiakeskeisesti sarjana muunnoksia, jotka jalostavat raakadataa yhä abstraktimpiin ja tiivistetympiin esitysmuotoihin. Todellisissa tiedonlouhintaprosesseissa on kuitenkin aina osa-alueita, joita tällainen malli ei kata riittävän hyvin. Erityisesti on huomattava, että eräät prosessin tärkeimmistä toimijoista ovat ihmisiä, eivät teknologiaa, ja että heidän toimintansa prosessissa on luonteeltaan vuorovaikutteista eikä sarjallista. Tässä väitöskirjassa ehdotetaan vakiintuneen mallin täydentämistä siten, että tämä tiedonlouhintaprosessin laiminlyöty ulottuvuus otetaan huomioon.

Ehdotettu prosessimalli koostuu kolmesta osamallista, jotka ovat tietomalli, työnkulkumalli ja arkkitehtuurimalli. Kukin osamalli tarkastelee tiedonlouhintaprosessia eri näkökulmasta: tietomallin näkökulma käsittää tiedon eri olomuodot sekä muunnokset olomuotojen välillä, työnkulkumalli kuvaa prosessin toimijat sekä niiden väliset vuorovaikutukset, ja arkkitehtuurimalli ohjaa prosessin suorittamista tukevien ohjelmistojen suunnittelua. Väitöskirjassa määritellään aluksi kullekin osamallille joukko vaatimuksia, minkä jälkeen esitetään vaatimusten täyttämiseksi suunniteltu ratkaisu. Lopuksi palataan tarkastelemaan vaatimuksia ja osoitetaan, kuinka ne on otettu ratkaisussa huomioon.

Väitöskirjan pääasiallinen kontribuutio on se, että se avaa tiedonlouhintaprosessiin valtavirran käsityksiä laajemman tarkastelukulman. Väitöskirjan sisältämä täydennetty prosessimalli hyödyntää vakiintunutta mallia, mutta laajentaa sitä kokoamalla tiedonhallinnan ja tietämyksen esittämisen, tiedon louhinnan työnkulun sekä ohjelmistoarkkitehtuurin osatekijöiksi yhdistettyyn malliin. Lisäksi malli kattaa aiheita, joita tavallisesti ei oteta huomioon tai joiden ei katsota kuuluvan osaksi tiedonlouhintaprosessia; tällaisia ovat esimerkiksi tiedon louhintaan liittyvät filosofiset kysymykset. Väitöskirjassa käsitellään myös kahta ohjelmistokehystä ja neljää tapaustutkimuksena esiteltävää sovellusta, jotka edustavat teknisiä ratkaisuja eräisiin yksittäisiin tiedonlouhintaprosessin osaongelmiin. Kehykset ja sovellukset toteuttavat ja havainnollistavat useita ehdotetun prosessimallin merkittävimpiä ominaisuuksia.

Asiasanat: ohjelmistoarkkitehtuuri, prosessimalli, tiedon louhinta, tiedonhallinta, tietämyksen esittäminen, työnkulun hallinta

In memory of
Tapani Tuovinen

Preface

Conventional wisdom has it that one should be able to complete a doctoral dissertation in four years; depending on where you start counting, it took me eight or nine to reach my destination. I cannot name a single reason to account for this, but perhaps the chief one is that I took what one might call the scenic route. A straight line may be the shortest path between two points, but it is not always the most interesting one. I might have finished sooner by sticking to that path, but then I would have missed a golden opportunity to mingle with a strange mix of computer scientists, engineers and philosophers. So I strayed, and shared ideas (as well as several bottles of wine) with each of those worlds, and took home something from each of them, and slowly pieced together my thesis from choice bits of this creative chaos.

I will not pretend that the path I took never got me lost or frustrated, but it got me there eventually, and I would like to thank my supervisor, Professor Juha Röning, for bearing with me and letting me find my own voice. My pre-examiners, Professor Christopher Clifton and Professor Vassilios Verykios, also have my gratitude; they did not let me off the hook easily, but I feel that the final result of my work is much better than it would have been without their backside-kicking efforts. Dr. Pertti Väyrynen contributed by checking my spelling, punctuation and grammar.

Doing science is teamwork, and I wish to recognize the contribution of all the good people of Intelligent Systems Group, past and present, who have co-authored papers with me. Everyone with whom I collaborated in the SIOUX, SAMURAI and XPRESS projects also deserve my thanks. Special thanks are due to Dr. Perttu Laurinen, who advised me during the first half of my winding journey, before I cut my teeth as an independent researcher. Speaking of teeth, I want to thank everyone at work with whom I have sat down for lunch, or a cup of coffee, or a pint — you have made the workplace a complete hoot on the best of days and tolerable even on the worst. You know who you are.

Several organizations have provided financial support for the completion of my thesis. I am particularly grateful to those that have awarded personal scholarships and grants: the Graduate School in Electronics, Telecommunications and Automation (GETA), the Tauno Tönning Foundation, and the Nokia Foundation. The projects in which I have created contributions included in the thesis received funding from the

European Commission, Tekes, Rautaruukki, and Polar Electro. The support of Infotech Oulu to the Intelligent Systems Group is also gratefully acknowledged.

In the software engineering community it is said, perhaps only half-jokingly, that in any given project, the first ninety percent of the work takes ninety percent of total development time, and the last ten percent takes the other ninety. I feel like something similar has happened in my case, and I want to thank all my family and friends for their patience in listening to my laughably optimistic estimates of when I would defend. To my parents, I am deeply grateful for fostering my love of learning (and computers) when I was growing up, not to mention all the material support they have given me during my studies. The final word goes out to my special someone Elisa, to whom I owe countless thanks for what she has done to my life since we met — I try my best every day to return the favor.

Oulu, March 2014 Lauri Tuovinen

Abbreviations

ACM	*Association for Computing Machinery*
AI	*Artificial Intelligence*
ANN	*Artificial Neural Network*
API	*Application Programming Interface*
CL	*Common Logic*
CRISP-DM	*Cross Industry Standard Process for Data Mining*
DBMS	*Database Management System*
DBSA	*Device-Based Software Architecture*
DLL	*Dynamic Link Library*
DM	*Data Mining*
EER	*Enhanced Entity-Relationship*
ER	*Entity-Relationship*
GUI	*Graphical User Interface*
HCI	*Human-Computer Interaction*
HTTP	*Hypertext Transfer Protocol*
ICT	*Information and Communications Technology*
IEEE	*Institute of Electrical and Electronics Engineers*
ISO	*International Organization for Standardization*
JSON	*JavaScript Object Notation*
JVM	*Java Virtual Machine*
KD	*Knowledge Discovery*
KDD	*Knowledge Discovery in Databases*
KE	*Knowledge Engineering*
KIF	*Knowledge Interchange Format*
KR	*Knowledge Representation*
ODBC	*Open Database Connectivity*
OODB	*Object-Oriented Database*
OQL	*Object Query Language*
OS	*Operating System*
OWL	*Web Ontology Language*
PC	*Personal Computer*

RAM	*Random Access Memory*
RDB	*Relational Database*
RDF	*Resource Description Framework*
REST	*Representational State Transfer*
RVM	*Relevance Vector Machine*
SA	*Smart Archive*
SE	*Software Engineering*
SIGKDD	*Special Interest Group on Knowledge Discovery in Databases*
SQL	*Structured Query Language*
SVM	*Support Vector Machine*
T2MT	*Task-to-Method Transformation*
UI	*User Interface*
W3C	*World Wide Web Consortium*
XML	*Extensible Markup Language*

Contents

1 Introduction

The word “computer” originally referred to a person, someone whose job is to perform scientific computations (Grier 2005). Before the advent of the electronic computer, this was a task involving a great deal of manual labor. A computer room, in those days, was simply a room where computers worked. Since then, however, the original usage of the word has been totally superseded by the modern one, so that in most contexts saying “electronic computer” or “digital computer” seems redundant.

Modern computers are used for such a wide array of tasks that the English-language legacy term hardly does them justice any more. Interestingly, in the Finnish language, the opposite has happened: the corresponding term has become more and more apt as the technology has progressed. The Finnish word for computer is “tietokone”, for which a literal translation would be “knowledge machine” — certainly an exaggeration when we consider the earliest computers, but quite accurate when we consider the ones we have now. The computers of yesterday computed, but the knowledge machines of today process knowledge.

This thesis is concerned with knowledge discovery, the process by which computers generate and accumulate new knowledge. There is an established process model of knowledge discovery, in which the process is viewed as a mostly linear sequence of transformations, with raw data as the original input and knowledge as the final output. In between, the data is cleaned, condensed, analyzed and converted from one representation to another until what remains is a crystallized expression of the meaning of the data. Experts can then use it, for example, to improve some aspect of an industrial or business process.

The thesis argues that while the established view is a good descriptive model of the knowledge discovery process, it represents a limited perspective. The principal hypothesis is that knowledge discovery projects would be better supported by a less linear process model with the following properties:

– The model is founded upon a *data model* that binds all the relevant states and representations of data within a single unified framework.
– The model embodies a *workflow model* that recognizes the importance of human actors in the knowledge discovery process and identifies the interactions on which the success of the process depends.

– The model is supported by an *architectural model* that enables the construction of knowledge discovery software tools by reflecting the data model and the workflow model.

By focusing on actors and interactions, the thesis emphasizes the collaborative nature of knowledge discovery. The collaboration takes on many different forms: sometimes, it takes place between humans, at other times between technological artifacts, and on certain occasions between humans and technology. The diversity of the interactions will necessarily be reflected in the variety of activities and tools designed to support them. Since it is humans who drive the process and since many of the interactions involve humans, supporting all aspects of the process requires the application of not just hardware and software but psychology and even ethics. The thesis therefore includes a significant cross-disciplinary component, applying methods and concepts adopted from philosophy of computing alongside those of computer science.

The remainder of this chapter explains the approach by which the thesis aims to validate its principal hypothesis. Section 1.1 gives definitions for certain key concepts and specifies the scientific objectives and scope of the thesis with respect to those concepts. Section 1.2 presents a summary of the contributions made by the thesis to the state of the art in the field of knowledge discovery. Section 1.3 enumerates the datasets and other materials used and the research methods applied to reach the conclusions of the thesis. Section 1.4 outlines the order in which the discussion proceeds from Chapter 2 onward.

1.1 Objectives and scope

Before it is possible to unambiguously formulate the objectives and scope of the thesis, it is necessary to define the terminology to be used. There are a number of relevant computer-science concepts bound together by the use of the word “knowledge”. These are defined here as follows:

– *Knowledge Discovery (KD)* is the process of extracting verified knowledge from digitally recorded data with the help of computer software. The main phases of the process, as commonly defined, are data acquisition and selection, pre-processing, feature extraction, modeling, and interpretation.
– *Knowledge Representation (KR)* is the digital encoding of knowledge according to a symbolic, machine-readable scheme suitable for automated interpretation and

inference. A *Knowledge Base (KB)* is a body of digitally stored knowledge that adheres to some KR scheme.
– *Knowledge Engineering (KE)* is the incorporation of expert knowledge in computer-based systems with the aim of enabling the systems to apply the knowledge for independent problem-solving.

To be entirely precise, the concept of knowledge discovery as used in this thesis should, in fact, more properly be called knowledge discovery in databases (KDD). However, for the sake of brevity, the less verbose form will be used.

Another widely used term, data mining (DM), is essentially a different name for a specific subset of the steps of the KD process; it is defined here as the process of applying computational algorithms to build a generalized model of a dataset, useful for predicting or describing the properties or behavior of the phenomenon represented by the dataset. It should be noted that while the thesis devotes considerable attention to data mining as an instrument of knowledge discovery, it rejects the implicit assumption that KD necessarily involves DM. For instance, in image classification, it may be a viable option to employ human volunteers instead of a classification algorithm because of the natural aptitude of humans for such tasks (Raddick *et al.* 2010). Making such choices is one of the aspects of the KD process emphasized by the model proposed in the thesis.

Given the above definition of knowledge discovery, the objectives of the thesis can be expressed as follows:

1. To identify aspects of the KD process that the established process model does not adequately reflect.
2. To define requirements for an augmented model that responds to the identified shortcomings.
3. To design augmentations to the established model based on the defined requirements.
4. To validate the augmented model against the requirements.

Although the objectives are all formulated with reference to KD only, the actual scope of the thesis is somewhat broader, incorporating elements of KR and KE through the inclusion of the data model in the augmented process model. The goal of the data model is to support not only the task of generating knowledge, but also those of storing and using it, which are all considered here to be part of the same higher-level process. KD, KR and KE are, in this thesis, viewed as different expressions of the same idea, namely that of a computer as an entity capable of creating and possessing knowledge.

The philosophical implications of such an idea are not negligible, and they shall be discussed at some length over the course of the thesis.

1.2 Scientific contributions

In terms of the three models listed where the principal hypothesis was introduced, the scientific contributions of the thesis can be summarized as follows:

- *Data model:* a reference architecture for databases that support the KD process; a KR scheme for method storage and retrieval based on task characteristics; implementations of the database design principles and the KR scheme as solutions to real-world manufacturing problems.
- *Workflow model:* identification of KD process actors and interactions neglected by the established process model; description of the significance of the actors and interactions; proposal for incorporating the actors and interactions in an augmented process model.
- *Architectural model:* a software architecture designed for KD applications; an implementation of the architecture as a software framework; real-world KD solutions implemented using the framework.

The thesis is largely based on results published previously in a number of international conference papers. These results are reproduced in the body text and figures as appropriate and necessary to support the conclusions of the thesis. More specifically, the conclusions rest on the following results:

- Development of a database for welding quality assurance (Tuovinen *et al.* 2007) and a knowledge system for task-to-method transformations in reconfigurable manufacturing systems (Tuovinen *et al.* 2010).
- Discussions of the ethical aspect of knowledge discovery (Tuovinen & Röning 2005) and the significance of human interaction in the KD process (Tuovinen & Röning 2009).
- Development and application of the Smart Archive framework (Laurinen *et al.* 2005, Tuovinen *et al.* 2008, 2009a,b) and its successor, the Device-Based Software Architecture (Kätevä *et al.* 2010).

Of the two papers published at computer ethics conferences, (Tuovinen & Röning 2005) and (Tuovinen & Röning 2009), the author of the thesis is the principal author in

both. Both papers are based on the author's own ideas and work. The co-author of these papers, Professor Juha Röning, provided comments and advice in the manuscript preparation phase.

Of the papers related to Smart Archive, the author of the thesis is the principal author in (Tuovinen *et al.* 2008), (Tuovinen *et al.* 2009a) and (Tuovinen *et al.* 2009b), and the lead designer and implementer of the work reported therein. Co-authors, Prof. Röning and Dr. Perttu Laurinen, had advisory roles. Additionally, Dr. Laurinen and Dr. Ilmari Juutilainen are the principal developers of the two Smart Archive-based applications that form part of the work reported in (Tuovinen *et al.* 2008).

Dr. Laurinen is the principal author of (Laurinen *et al.* 2005), a paper describing an earlier version of Smart Archive, which he developed. The author of the thesis assisted in formulating the point of view of the paper and editing the manuscript. The author had a similar role in the writing of (Kätevä *et al.* 2010), the paper on the Device-Based Software Architecture, the principal author of which is M.Sc. Janne Kätevä. Additionally, the author participated in the early stages of the specification and design of the framework.

The author of the thesis is the principal author of the two papers on databases and knowledge bases, (Tuovinen *et al.* 2007) and (Tuovinen *et al.* 2010), advised again by Prof. Röning and Dr. Laurinen. The author also designed and implemented the database described in (Tuovinen *et al.* 2007); co-authors Dr. Heli Koskimäki and M.Sc. Eija Haapalainen assisted in the testing of the solution. The knowledge system reported in (Tuovinen *et al.* 2010) was designed and implemented by the author and co-authors Tuomas Talus and Esa Koponen, using the work of Janne Kätevä.

1.3 Materials and methods

Chapters 3–6 represent the main body of this thesis. Of these four chapters, the first two are concerned with elaborating the principal hypothesis, whereas the other two are dedicated to testing the hypothesis. The methods employed to accomplish these two tasks are reviewed here.

For hypothesis elaboration, the first step was to establish a set of requirements that a successful solution should satisfy. Literature reviews and conceptual analysis were used to build a balanced view of the current state of KD research, from which the requirements were derived in a manner imitating the software engineering practice of requirements solicitation. When studying the literature, three questions were deemed of

particular interest, corresponding again to the three component models of the proposed process model:

– What is the relationship between knowledge discovery and epistemology, i.e., the philosophical study of knowledge?
– Who are the stakeholders in the KD process — including those who do not actively participate in it — and what are their contributions and expectations?
– How is the KD process reflected in the architecture of software tools created to support the process?

The dominant approach followed in the development of the solution is constructive research. In the case of the data model and the architectural model, the artifacts constructed are software applications, whereas in the case of the workflow model, the results are more abstract. Besides constructive research, some elements of action research were included in the approach, as it was sometimes the case that the artifact under construction was being developed within an organization as part of an effort to improve an organizational function.

For hypothesis testing, the first approach employed was requirements tracing, again mirroring the software engineering practice. The goal was to demonstrate through observation and reasoning how each requirement is satisfied by a specific component of the proposed solution. To support the requirements trace with additional evidence, a number of case studies on the constructed software artifacts were used. For these studies, several datasets were provided by project partners:

– The welding database was populated with data from welding experiments carried out by Harms & Wende, Stanzbiegetechnik and Technax Industrie.
– For Smart Archive, Rautaruukki provided data and specifications for an application for maintenance of predictive models in the steel industry; Polar Electro did the same for an application for analyzing personal fitness data.
– Test cases for the manufacturing knowledge system were defined by KUKA Systems and Karlsruhe University of Applied Sciences.

In addition to real-world data and applications, synthetically generated datasets and simulation tests were used in some cases to evaluate the performance of the artifacts studied.

1.4 Organization of the thesis

Before the treatment of the main subject matter of the thesis begins, Chapter 2 presents a review of related work in four major areas of research. These are data mining, software architecture, databases and knowledge bases, and philosophy of computing. The chapter provides both a state-of-the-art survey of relevant issues and a historical perspective on the technologies discussed.

Chapters 3, 4 and 5 each follow a similar structure, with sections corresponding to the data model, the workflow model, and the architectural model. Chapter 3 establishes the requirements for each model, Chapter 4 presents the proposed models, and Chapter 5 performs a requirements trace for each model, as described in the previous section.

The case studies, four of them altogether, are presented in Chapter 6. The significance of each case study is evaluated in terms of what it tells about the ability of the proposed solution to respond to the established requirements. This is followed by an evaluation of the contributions of the thesis in Chapter 7, which discusses the strengths and weaknesses of the presented results, assesses the limits of their applicability, and highlights important questions to which the thesis offers no answers. The thesis is then concluded by Chapter 8.

2 Background

The technological development leading up to the current state of the art in knowledge discovery spans a number of computing disciplines. Given the focus of this thesis, the most important categories of prior research and development are data mining, software architecture, database management and philosophy of computing. This chapter presents a review of pertinent literature on these four major topics.

The relevance of data mining research is obvious, since DM technology is one of the principal instruments of knowledge discovery. Literature on DM methods and applications is therefore reviewed, even though the thesis does not purport to demonstrate new results in the field of DM per se. This review, consisting of a concise history of DM and a survey of major techniques and applications, is presented in Section 2.1.

A significant portion of the scientific contribution of the thesis is in the area of software architectures for KD applications, which is why it is important that literature on software architectures be reviewed here. The review, in Section 2.2, introduces the theoretical concepts necessary for the full appreciation of the research results discussed in Section 4.3. The section also surveys KD software architectures that will serve as points of reference for the solution proposed in this thesis.

Management of data and knowledge is another area of KD technology where the thesis makes a contribution. In this area, the thesis focuses on two major topics: designing databases to support the knowledge discovery process, and representing knowledge in digital form. Section 2.3 presents a review of literature on these topics, along with a discussion of the concepts of data and knowledge.

The final section of this chapter reviews literature on the philosophy of computing in general and on the philosophy of KD in particular. This is done because the thesis uses philosophical theories and methods to study the concept of knowledge in KD and the social and ethical implications of the KD process. To provide additional context for these discussions, the review in Section 2.4 paints a broad picture of philosophy of computing as an academic discipline.

2.1 Data mining

This first section of the literature review serves two purposes. One is to provide an overview of the state of the art in the field of data mining. This overview is further

split into two parts: one reviews the techniques of DM, while the other discusses the applications and practical impact of DM. These areas are covered in Subsection 2.1.2 and Subsection 2.1.3, respectively.

Before the state-of-the-art review, however, the history of data mining is discussed at some length. This discussion is also split between two perspectives: first the history of the enabling technologies of DM is recounted, after which the history of the adoption of DM as a new technology of practical value is briefly reviewed. Both perspectives are presented next, in Subsection 2.1.1.

2.1.1 A historical perspective

Going back in time, the history of electronic computing merges seamlessly into the history of mathematics, so any discussion on the history of a given field of computing could, with some justification, be extended to antiquity. However, a moderately recent starting point is available in Thomas Bayes's posthumously published essay on probabilities (Bayes 1763). Starting with the Bayesian theory of probability is still a somewhat arbitrary choice, but not entirely, since Bayesian probability computations underlie many DM techniques, starting with certain formulations of linear regression. Regression analysis as such predates all-purpose electronic computers by nearly a century and a half and thus is not at all exclusively a DM technique, but naturally the rise of the computer has allowed it to be used to tackle much more complex problems than could ever be solved manually. The earliest form of linear regression, the least squares method, was invented independently by Adrien-Marie Legendre (1806) and Carl Friedrich Gauss (1809).

Linear regression represents an early example of a mathematical model that learns a sample dataset and outputs a function that predicts the overall distribution of data coming from the same source as the original sample. The digital computer, pioneered in the mid-20th century by Claude Shannon (1940), Alan Turing (1937), Konrad Zuse (Giloi 1997) and John von Neumann (1945), among others, has radically transformed the field of possibilities of such models. The ability to learn from data combined with the ability to represent the learning results in the form of a stored program led to the birth of a new computing discipline, machine learning.

Being the combination of these two main components, machine learning can be roughly defined as experience-based self-modification of the behavior of a computer program, where the experience is represented by a set of individual observations, the

training dataset. For DM, in turn, a rough definition can be composed by combining machine learning with the ability to store, organize and search large quantities of data efficiently in databases. In practice, it was the relational model of data that allowed sufficiently powerful database systems to be designed and built.

The relational model has its roots in classical set theory and predicate logic, so a thorough exploration of this topic would again lead us to plunge rather deep into the history of mathematics. However, the use of relations to represent database structure and content was first proposed by Edgar F. Codd (1970). Relational database management systems began to appear as commercial products roughly a decade later, and in the 1980s, they became the predominant type of database. Although other database models have been proposed and implemented since then, relational databases remain a popular data management solution.

From the rise of relational database systems, it is again approximately a decade to the origin of the term "data mining". The details of its coinage are unclear, and it is safe to assume that various organizations were doing data mining before the term was invented, calling it "data analysis" or perhaps "business intelligence". In any event, based on its occurrence in research literature, it appears that "data mining" has been a standard part of computing terminology since the early 1990s.

Adopting a business perspective instead of a technical one, the beginning of DM may be placed at the late 1980s, a time when many companies were counting their losses from having invested radically in artificial intelligence (AI) technology that was not ready to deliver what it promised to. When investments in AI did succeed, it was the result of moderate expectations and a solid understanding of the real capabilities of the technology. These became the foundation of DM as a commercially viable industry. Among the early adopters of DM were organizations operating in finance, retail, insurance and telecommunications. (Collier *et al.* 1998)

The first companies specializing in DM software, such as ANGOSS Software Corporation and Pilot Software, entered the market in the mid-1980s. The size of the industry remained steady until the end of the decade, but in 1990 began a major growth period; between 1990 and 1998 the total number of companies multiplied by a factor of 7. A survey in 1999 (Goebel & Gruenwald 1999) identified over 40 available software packages for DM and assessed most of them to be of commercial strength. The first companies offering DM services to businesses were founded in the middle of the decade, and at the same time, traditional ICT companies began to incorporate DM into their

information systems expertise. IBM, for example, launched its Intelligent Miner in 1996. (Collier *et al.* 1998)

Integration in various forms has been a characterizing aspect in the development of DM at and after the turn of the century. It is now common for major database vendors to offer DM packages as optional features of their main products; IBM does this with its DB2 Data Warehouse Edition, as does Microsoft with SQL Server Analysis Services and Oracle with Oracle Data Mining. An example of industrywide integration is provided by the creation of DM standards, which include consortium-defined standards for application programming interfaces, model specification and the DM process, as well as an extension to the ISO SQL standard (Clifton & Thuraisingham 2001, Grossman *et al.* 2002).

2.1.2 Data mining techniques

From linear regression, it is not a long conceptual leap to linear classification. The objective in classification is to predict discrete class labels rather than continuous variable values, but many of the principles of statistical regression analysis still apply; for example, the least squares method can be adapted to construct a simple linear classifier. Other early classification techniques include Fisher's linear discriminant (Fisher 1936) and the perceptron (Rosenblatt 1958). The latter is particularly interesting in that it represents an early example of artificial neural network (ANN) modeling. Later developments such as the multilevel feed-forward ANN proposed by Rumelhart *et al.* (1986) have made it possible to develop nonlinear models. Even more recently, nonlinear methods based on the kernel concept (Aizerman *et al.* 1964) have been developed; these include support vector machines (SVM) (Boser *et al.* 1992) and relevance vector machines (RVM) (Tipping 2000). ANNs, SVMs and RVMs are applicable to both regression and classification problems.

Another important group of DM methods is used for grouping data points into clusters, defined informally as subsets of points that are close to one another and far from points in other subsets. Unlike classification, clustering algorithms do not require category labels to be included in training data; instead, cluster membership is inferred directly from the data. Two closely related clustering techniques represent the seminal work in this area: the EM algorithm (Dempster *et al.* 1977) and the K-means algorithm (Lloyd 1982). The two algorithms yield similar results but differ in that EM is probabilistic, whereas K-means is not. These suffer from the drawback that the number

of clusters must be specified in advance; this is avoided by a more recent method, the QT algorithm (Heyer *et al.* 1999).

Agrawal *et al.* (1993) and Agrawal & Srikant (1994) represent the seminal work on association rule discovery. An association rule specifies that given a large number of sets of items, certain items occur together in the itemsets with a certain likelihood and confidence. The standard textbook example of association rule discovery is market basket analysis, which seeks to identify products that are frequently purchased together. The classic Apriori algorithm given in (Agrawal & Srikant 1994) is comparably inefficient but has served as a basis for improved algorithms such as (Mannila *et al.* 1994) and (Park *et al.* 1995).

Mining temporal or spatial data requires modeling methods capable of capturing the dependencies that typically exist between elements that lie close to one another in such datasets. The earliest reports of temporal pattern discovery were published in the 1980s, e.g. (Dietterich & Michalski 1985), but broad interest in temporal data mining did not arise until the mid-1990s, at which time research on spatial data mining also began to be published (Roddick & Spiliopoulou 1999). Among the methods developed for temporal and spatial DM are algorithms for classification, association rule discovery, trend analysis, and clustering (Roddick & Spiliopoulou 1999, 2002).

Towards the end of the 1990s and on into the 21st century, there has been a growing interest in mining increasingly complex and unstructured types of data. Automatic classification of text documents using a Bayesian probabilistic approach was experimented with already in the early 1960s (Maron 1961), but as computers began to be employed for processing images and then audio and video signals, new challenges arose for DM. These challenges are addressed in (Zaïane *et al.* 1998) and (Thuraisingham *et al.* 2001), among many others. Another branch of DM research that has emerged relatively recently deals with page content, link structures and usage patterns on the World Wide Web (Cooley *et al.* 1997, Jicheng *et al.* 1999).

2.1.3 Practical significance

A 2006 special issue of the Explorations newsletter of the ACM Special Interest Group on Knowledge Discovery in Databases (SIGKDD) consists of ten case studies on real-world applications of data mining (Melli *et al.* 2006). The case studies reflect a broad spectrum of application domains for DM, ranging from healthcare to web marketing. To illustrate this diversity, four general categories of applications of DM

shall be reviewed here: science, industry, business and administration. The categories are not necessarily disjoint and they are not meant to be exhaustive, but they form a solid enough foundation for an assessment of the variety and scale of the benefits yielded by the appropriate use of DM technology.

In the science category, medical applications of DM are prominent. These include analysis of medical images (Dimitriadou *et al.* 2004), general diagnostics (Tan *et al.* 2003), detection of particular illnesses such as cancer (Li *et al.* 2004), arthritis (Wyns *et al.* 2004) and dementia (Zaffalon *et al.* 2003), and classification of tumors (Antal *et al.* 2003, Lukas *et al.* 2004). Data mining techniques are useful for analyzing genomic data, which makes DM a valuable tool for the life sciences also in a pure-research context, as exemplified by King *et al.* (2000). Another example of a pure science that can be aided by DM is astronomy (Grossman *et al.* 2001).

In industrial production, there are numerous examples of successfully using DM to improve the quality of products or processes. Traditional industries that have benefited from the application of DM methods include welding (Zhang *et al.* 2003a), steelmaking (Cser *et al.* 1999), oil refining (Chen *et al.* 2004), and electronics manufacturing (Kusiak & Kurasek 2001). A 2006 report regarding the use of DM in software maintenance (Kanellopoulos *et al.* 2006) shows that the technology can also help industries that work with nonphysical artifacts.

The business applications of DM are, to a considerable extent, focused on improving the marketing of products. A typical textbook example is market basket analysis; see, e.g., (Singh *et al.* 2006, Setiabudi *et al.* 2011) for case studies. Data mining can also be used for more precise targeting of marketing campaigns and products (Chou *et al.* 2000, Gersten *et al.* 2000). Businesses that may suffer losses due to fraudulent or otherwise improper use of the services they provide can use DM to help minimize these losses (Brause *et al.* 1999, Viaene *et al.* 2004, Pinheiro *et al.* 2006). Another non-marketing application is customer service quality improvement, as described by Ha & Park (2006).

Fraud detection by DM works equally well for government organizations carrying out such functions as taxation (Bonchi *et al.* 1999, DeBarr & Eyler-Walker 2006) and customs (Shao *et al.* 2002). Law enforcement in general is an important administrative application domain of DM: the investigation of many types of crime can be supported by computational pattern analysis (Brown & Oxford 2001, Underson 2002, Zhang *et al.* 2003b, Adderley & Musgrove 2001). Security-related applications of DM, especially ones pertaining to terrorism prevention, have raised privacy concerns, which in turn

have spawned a new branch of DM research known as privacy preserving data mining (Jensen *et al.* 2003, Wahlstrom & Roddick 2000, Verykios *et al.* 2004).

2.2 Software architecture

The role of software architecture in software engineering (SE) is to guide the development of a software system by providing a set of abstract, mutually complementary views of the composition of the system. Each view depicts a distinct aspect of the system in terms of parts, their responsibilities and their relationships with other parts. Furthermore, fundamental constraints, such as the decision to use a particular implementation platform, are also included in the architectural specification of a system.

The more complex the system to be developed and the larger the team working on it, the more important it is to have a comprehensive and thoroughly documented software architecture. This is because the architecture consists of those decisions regarding the design of the system with which all further design and implementation decisions must comply. Each developer must therefore be aware of the agreed-upon architecture in order to avoid making decisions that are incompatible with those of other developers. If the work of an individual developer is in conflict with the architecture, either the developer must find a different way to achieve the same outcome or the architecture must be changed and the change propagated to all parts of the system.

A software architecture generally exhibits one or more distinct architectural styles. Different styles have different strengths and are suitable for different tasks, so the choice of styles depends on the requirements of the software. Architectural styles in general are discussed in the first subsection, followed by a discussion of software frameworks, which are reusable embodiments of software architectures, in the second. The last subsection reviews software frameworks intended specifically for the development of KD software.

2.2.1 Architectural styles

Defining the concept of an architectural style without resorting to something trivial such as “a style of software architecture” is not an entirely easy task. Since the word “style” is intuitively understandable, this is not necessarily a big problem. However, it would still be good to have a slightly more formal definition to work with before we proceed.

One way to define architectural style is by analogy with the design pattern concept introduced in the seminal work by Gamma *et al.* (1994), or the Gang of Four as they are colloquially known. Design patterns are abstract designs described using the concepts of object-oriented programming — classes, class members and relationships between classes. Each pattern represents a tried-and-true solution to a particular category of problems, so when developers encounter a problem belonging to a category covered by an existing design pattern, they know they can create a good design by following the pattern. Therefore, whereas a design is a concrete solution to a problem, a design pattern is a model that shows how to solve every problem in the category addressed by the pattern.

Similarly, when we consider a software architecture as a solution to a particular problem, we can define an architectural style as an abstract model that shows how to create architectures applicable to a group of problems. Just like a software design may employ multiple design patterns, a software architecture may exhibit multiple architectural styles. The main difference is that instead of classes, architectural styles are described in terms of larger entities such as components or subsystems. Bass *et al.* (2003) use the term "architectural pattern" to refer to the same concept, further emphasizing the analogy to design patterns.

The architecture of a software system is a composite of multiple complementary structures, some depicting the static arrangement of architectural elements, others the run-time dynamics of the system. From the run-time perspective, one of the most important defining features of software architecture is communication: what communicational links exist between subsystems and what roles the subsystems assume as members of those links. We can therefore roughly categorize architectural styles based on the communicational patterns they exhibit.

The simplest style of software architecture, from this perspective, is monolithic architecture. In a pure monolithic architecture, there is no division into subsystems at all, so there is also no communication among subsystems. This is a perfectly viable option if the application being built is very simple, although it is questionable whether any actual architectural design takes place in such cases. In the majority of software engineering projects, it will be necessary to partition the application, for example, by its major functions in order to create natural units of responsibility to be distributed among the members of the development team. (Bass *et al.* 2003)

In architectures where there are communicating subsystems, the topology of subsystems and communication links is often centered around a hub that plays a

dominant role in the system. An especially common hub-based architectural style is client-server architecture, where the hub (server) provides services that the other subsystems (clients) can use by sending requests to the hub. Most web applications — information retrieval services, social networking sites, browser-based games, et cetera — are examples of client-server-style software architecture. Alternatively, the hub can be a shared data structure accessed and manipulated by the surrounding subsystems; this is how software architectures following the blackboard style are organized (Engelmore & Morgan 1988).

Although hub-based architectures are popular, a hub is not necessary to coordinate communication among system components. An alternative way to organize a system is to have the components organize themselves through local interactions with other components. In such architectures, no component dominates over others: each component depends upon some components while being depended upon by others, and there is no globally controlling entity to determine the flow of execution in the system.

Data flow architecture is an architectural style that represents software as a collection of data streams manipulated by the system components through which they pass. The components and the data streams connecting them form a graph structure where each node carries an equal weight in determining the outcome of execution. There may be a separate controlling component that alters the structure of the graph by rerouting the streams, but it is not a hub from which the other components request services. Instead, each component in the graph implements services for or requests services from its neighbors, i.e., those components from which it receives or to which it sends data streams. Another name for this style of architecture is pipes and filters (Buschmann *et al.* 1996).

In data flow architecture, the links between components are externally defined and relatively stable, but there are also architectural styles where the links are made by the components themselves and broken as soon as the relevant data has been exchanged. The components in such architectures are independent units that seek out the services of other components, each according to its own agenda. Well-known examples are found among web applications that are not based on client-server architecture; peer-to-peer architecture (Schollmeier 2001), for instance, is popular in file sharing applications such as BitTorrent. Another notable style, agent-based architecture (Woolridge & Jennings 1995), relies on autonomous interactive components known as agents and has frequently been proposed as an architectural solution for adaptive systems, e.g., in the field of intelligent manufacturing (Shen *et al.* 2006).

The choice of architectural style or styles for a system depends on the purpose and requirements that the system is expected to fulfill. Although there are often multiple alternative approaches that all lead to the same outcome, it is generally the case that some of them are better suited than others to solving the problem at hand. Architectural styles that are particularly suitable for KD software are discussed in Subsection 2.2.3.

2.2.2 Software frameworks

Reusability is an issue constantly on the table in software engineering. In most software development projects, the fact is that something similar has already been done, quite probably by the same team or at least the same organization, so it would make sense to build on that earlier work rather than start from nothing. The goal of reusability research is to help developers design and code with reusability in mind in order to make it more likely that their solutions can be used again in future projects.

The earliest and most elementary form of reuse in software engineering is copying a passage from one code file and pasting it into another one. Occasionally, this is still justified, but in most cases, more sophisticated and elegant ways of reusing work are favored. Code is commonly shared and reused via function and class libraries; designs and architectures can be made reusable by codifying them as design patterns and architectural styles. Software reuse is thus a much broader concept than mere code reuse, referring instead to the reuse of software engineering artifacts on every level of abstraction (Jacobson *et al.* 1997).

At the intersection of these diverse ways of building reusable SE artifacts lies an interesting special case: software frameworks. A software framework combines a reusable design with a body of reusable code, making frameworks distinct from other types of reusables. Compared with architectural styles and design patterns, the difference is obvious: frameworks contain code, which styles and patterns do not. Compared with code libraries, on the other hand, the distinction is slightly more subtle.

When a developer writes a program using a code library, the developer writes the main routine of the program and calls library functions from there. When a developer uses a software framework, this relationship is reversed: the framework provides the main routine, calling code written by the developer at specific points. The architecture of the application is thus defined by the framework, whereas the solution-specific functionality is included in the parts created by the developer. (Gamma *et al.* 1994)

One could say that a software framework is the concrete manifestation of a software architecture, considered in isolation of any specific problem to be solved. Before details of functionality are fixed, an architecture defines not just one solution, but a class of solutions, and the same is true of a software framework that implements the architecture. Adopting a framework therefore means that the application will have all the qualities of the generic architecture embodied in the code of the framework. (Gamma *et al.* 1994)

Choosing the right framework for a software engineering project is not a trivial task: the framework should be reliable, suitable for the application domain, powerful enough for the application, and understandable to the developers. Provided that these requirements are satisfied, however, a framework brings considerable benefits to the development process. Since only the application-specific parts need to be designed, coded and tested, developing with an appropriate framework is faster than without one, and the architectural similarity of applications based on the same framework improves maintainability across the product line (Gamma *et al.* 1994).

In practice, almost all graphical applications developed for modern desktop platforms are programmed using an event-based software framework. These frameworks make it simpler for developers to implement windowed applications by providing an intuitive interface to various user interface elements and the events they generate when activated. Currently popular frameworks include .NET for Microsoft Windows, Swing and SWT for Java, and the cross-platform framework Qt. These are fairly generic frameworks, not focused on any specific application domain, but domain-specific frameworks have also been developed for various purposes. The next subsection gives examples of domain-specific software frameworks for KD.

2.2.3 Software architectures for knowledge discovery

The nature of the knowledge discovery process is such that intuition immediately suggests certain architectural styles for KD applications. If one takes the essential elements of the KD process and maps them to components in a software architecture, there are not many fundamentally different architectures at which one can arrive. As a consequence, the same basic patterns tend to be found in architectures designed for KD software, although there is still room for considerable variation when one examines these architectures in more detail.

The KD process has not been properly discussed yet in this thesis, apart from the short definition of KD given in Section 1.1. However, at this point, it is sufficient to

describe the KD process as a progression of data transformations for refining a set of observed data points into a predictive or descriptive model. A more comprehensive treatment of this topic is given later, in Section 3.2.

The model of knowledge discovery as a process of stepwise refinement of data translates naturally into a software architecture where the transformation steps are represented by system components. Through the components, the data proceeds in the manner of a stream. In other words, the result is a pipes-and-filters architecture, and unsurprisingly, this is usually the primary architectural style of software frameworks intended for the development of KD applications. Some notable examples are listed below:

– RapidMiner, formerly known as YALE (Mierswa *et al.* 2006), is an open-source KD environment originally developed at the University of Dortmund. Filters in RapidMiner are referred to as operators and arranged into an operator tree, a structure designed as a trade-off between the restrictive simplicity of an operator chain and the arbitrary complexity of an operator graph. The software emphasizes rapid prototyping of KD solutions via a graphical user interface (GUI), but it can also be operated on the command line or through an application programming interface (API).
– KNIME (Konstanz Information Miner) (Berthold *et al.* 2006) is another open-source platform for KD applications, originally developed at the University of Konstanz. KNIME filters are known as nodes, but apart from terminology, its application architecture is a standard pipes-and-filters one. The software is based on the Eclipse platform (des Rivières & Wiegand 2004) and is primarily a GUI application, although it also offers an API for developing extensions to the basic KNIME package.
– D2K (Data2Knowledge) (Data2Knowledge 2005) is a proprietary KD tool developed by the eponymous Data2Knowledge Corporation. The company markets data analysis as a service, but also offers developers the option to license the D2K analysis engine for inclusion in their own applications. Details on the architecture of D2K are not available to the public, but the white paper for prospective developers indicates that it follows the standard pipes-and-filters pattern, although it has again its own terminology, referring to filters as data transformation components.
– Chipster (Kallio *et al.* 2011) is an open-source analysis platform for high-throughput data developed at CSC, the Finnish IT center for science. The particular focus of the software is on bioinformatics, which is reflected in its selection of analysis tools and default workflows. Another distinguishing feature of Chipster is its client-server

architecture: workflows (processing pipelines) are created using a client application running on the user's desktop, but the actual processing takes place on a remote server that has the required analysis packages installed.

– Weka (Hall *et al.* 2009) is a library of KD algorithms developed at the University of Waikato. The algorithms can be called from the user's own code, but the library also comes with a suite of text-based and graphical interfaces for applying the algorithms and arranging them into pipelines. Weka has become a popular resource among developers of KD software; the complete library has been integrated to RapidMiner and KNIME.

The increasing computational difficulty of KD problems, especially in certain fields of science, has made it necessary to develop KD solutions that harness the power of distributed computing resources. Grid computing is the umbrella term under which these are gathered. Grid computing platforms come in two basic flavors, distinguished more by terminology than by functionality: some are branded as knowledge discovery frameworks, while others use the term scientific workflow management instead, emphasizing the scientific applications of KD. Within the latter category, there are some deviations from the architectural archetype. Examples of grid computing platforms are given below:

– Discovery Net (AlSairafi *et al.* 2003) and Knowledge Grid (Cannataro *et al.* 2004) are grid computing frameworks designed for knowledge discovery. Both are based on visual composition of KD process graphs to be executed on the grid.
– Taverna (Oinn *et al.* 2004) is a grid computing framework for bioinformatics applications; Pegasus (Deelman *et al.* 2004), GridNexus (Brown *et al.* 2005) and Kepler (Ludäscher *et al.* 2006) are generic scientific workflow management frameworks. MagentA (Walton & Barker 2004) is similar in purpose but uses an agent-based architecture.
– Triana (Taylor *et al.* 2003) is based on a grid composed of resources owned by individuals rather than organizations. The participation of non-expert individuals in the KD process as providers of computing resources is an important theme in the thesis, discussed at length in Section 4.2.
– MapReduce (Dean & Ghemawat 2008, 2010) is a programming model inspired by functional languages that is suitable for solving various data analysis problems. The Hadoop platform provides a popular open-source implementation, with applications, e.g., in the field of bioinformatics (Taylor 2010).

Since databases play a crucial role in the KD process, database-centered architecture is another appropriate architectural style for KD software. This style is characterized by the presence of a central data repository in the role of a hub via which other subsystems exchange information. The style is not explicitly addressed by KD frameworks, but they do enable it to be used in conjunction with the pipes-and-filters style. The distributed computing application described in Subsection 6.2.2 gives an example of using multiple instances of a pipes-and-filters framework to build a database-centered KD system.

Besides suggesting particular ways of composing the software architecture of KD applications, the KD process imposes some additional quality requirements that the architecture must be able to satisfy. Processing data in very large quantities or in real time requires considerable performance and constrains, for instance, the choice of implementation technologies. Efficient and comprehensive interfaces to data sources and stores are also essential, so it is important for the architecture to make appropriate provisions for them.

Another use for external interfaces in a KD application is making it interoperable with other applications. This makes it easier to build a KD system composed of multiple specialized tools, each dedicated to a particular step in the KD process. On the other hand, it may be desirable to be able to extend the application with new processing modules after initial deployment, in which case an internal interface must be specified for this purpose. It is thus beneficial for the architecture of KD software to be open, both externally and internally.

2.3 Databases and knowledge bases

One of the most basic functions of the digital computer is the ability to keep large quantities of data in permanent storage. What is considered a large quantity depends heavily on context; the high-end hard drive units sold to consumers today are capable of storing one terabyte or more. A large organization with highly data-intensive functions, such as a major research institute, may require thousands of terabytes of storage capacity. For example, the Large Hadron Collider at CERN produces 15 petabytes of data during one year of operation (CERN 2008).

The ability to store data is useless unless paired with the ability to find and retrieve desired data with sufficient ease and speed, a problem that grows ever more difficult as the body of searchable data increases in size and complexity. Database management

systems (DBMS) provide the solution to this problem by keeping track of each data item and providing an interface that helps users identify and locate the items they need.

The evolution of database models and database management systems has made it reasonable to ask whether it would be possible to use computers to store not just data but knowledge. The first step toward an answer is clearing up what exactly is the difference between data and knowledge. This question is examined in the first subsection. The second subsection reviews various types of DBMSs, and the third one discusses the idea of representing knowledge as data in a database.

2.3.1 The data-knowledge distinction

Data, etymologically speaking, is the plural form of the Latin word *datum*, meaning "something given". Incidentally, this is not that bad as an informal definition of data as the concept is understood in knowledge discovery: data is the starting point, accepted as it is, and everything else is derived from there. From this, it follows that if the quality of the data is poor, any knowledge extracted from it is suspect at best.

Implicit in the above is that for any given body of data, there is a way to determine the quality of the data. The data was recorded for a purpose, so the quality of the data is determined by how well the data serves that purpose. Another way to put this is that the data carries a meaning, and its quality is determined by how accurately and comprehensively it conveys that meaning. This is what distinguishes data from mere noise, which by definition contains no information and therefore means nothing.

It should be noted that the situation is not always as straightforward as indicated in the previous paragraph. A dataset collected for a specific purpose may later be adopted for a considerably different purpose, or the purpose of the data may be vague to begin with and clarified later. The meaning and quality of a dataset are therefore not entirely intrinsic but depend on the context in which the dataset is placed.

The notion of data carrying a meaning brings us to the concept of knowledge, specifically digitally stored knowledge. The philosophical concept of knowledge will be covered later, in Subsections 3.1.1 and 4.1.1; here, the focus is on a much narrower issue, namely the distinction between digital knowledge and digital data. Clearly, the two are related, but we need to make explicit the exact nature of the relationship.

As an initial step, we will adopt the following working definition: a *digital representation of knowledge* is a body of data that has the meaning of the data encoded as part of the data. This definition rests on the following premises:

– Everything that is stored digitally, including representations of knowledge, is composed of data. Therefore, the relationship of data to knowledge is that of a set to its subset. In other words, all knowledge is data, but not all data is knowledge. The task of distinguishing knowledge from data is thus reduced to finding the defining attributes of this particular subset.
– Any useful body of data has a meaning, but the meaning is contingent on the context within which the data is interpreted. If the context has not been determined, neither can the meaning be.
– A knowledge representation should be self-contained rather than dependent upon externally imposed context. A representation of the intended context should therefore be included, along with the data itself.

The question of how to digitally encode the meaning of data is examined in other parts of the thesis, most importantly, Subsection 4.1.2 and Subsection 6.1.2.

2.3.2 Database systems

The earliest database management systems emerged in the 1960s. The dominant database modeling approaches at the time were the network model and the hierarchical model. In both models, a database consists of records, which in turn consist of data fields and pointers to other records. The hierarchical model is more strict in how it allows records to be linked, but with both hierarchical and network databases, it is up to the programmer to navigate the pointers to find the desired record. These types of databases, collectively known as navigational databases, are considered the first generation of database management systems. (Connolly & Begg 2005)

The invention of the relational model and the introduction of DBMSs based on it transformed the scene quite radically. The great impact of relational databases (RDBs) was due not only to the relational model itself, but also, perhaps even more importantly, to the relational query languages developed for manipulating these databases. One such language, Alpha (Codd 1971), was proposed by E.F. Codd himself, but the SQL language, or SEQUEL as it was originally abbreviated (Chamberlin & Boyce 1974), eventually supplanted this and all other RDB languages, becoming the undisputed standard in all major implementations of the relational model, which constitute the second generation of DBMSs (Connolly & Begg 2005).

SQL transformed database programming in two important ways. First, unlike the procedural interfaces that were used to access navigational databases, SQL queries are declarative, meaning that the programmer only needs to declare the desired result, not to specify in detail how the result should be arrived at. This is done by describing the result set in terms of relational and logical operations applied to the relational tables that make up the database. It is then the task of the query optimizer, a component of the DBMS, to determine the optimal way to build the result set.

The second thing that makes SQL attractive from a programmer's point of view is its ubiquity, which enables programmers, in theory at least, to make their software interoperable with all relational DBMSs without having to code separate query statements for each DBMS they wish to support. In practice, the situation is not quite so straightforward, because every SQL implementation has its own small idiosyncrasies and deviations from the ISO standard. The problem of accounting for multiple DBMS vendors, each with their own database APIs and SQL dialects, is alleviated by higher-level wrapper APIs such as Microsoft's Open Database Connectivity (ODBC).

With several good options entering the market in the late 1970s to mid-1980s, relational DBMSs soon became the database management solution of choice for most purposes. Most likely they are still the most commonly used type of database in the world, given that widely recognized brands such as Oracle, Microsoft SQL Server and MySQL are all relational (albeit not necessarily purely relational) systems. However, the development of new database models and technologies has not ceased there. In the 1990s, software engineering underwent a paradigm shift where object-oriented programming supplanted procedural programming as the dominant programming paradigm. This, in turn, spurred interest in a new type of DBMS: object-oriented databases (OODB).

An object in an object-oriented programming language is an entity that encapsulates data (the attributes of the object) with functions that operate on the data (the methods of the object). Object-oriented databases are based on the notion of making objects persistent, i.e., storing them in such a manner that their state can be restored even after they have been erased from the RAM of the original host computer. A persistent object in an OODB is equivalent to a record or table row in earlier types of databases.

An interesting feature of OODBs is that persistent objects may contain references to other persistent objects. For instance, a collection object such as a list contains references to the members of the collection, and the member objects can be accessed by following references from the collection object. Therefore, OODBs are, in a sense, navigational databases, although object-oriented DBMSs typically also provide a search

function in the form of a query language such as OQL (Cattell *et al.* 2000). Moreover, since objects contain functions as well as data, objects containing other objects can implement search functions of their own to help programmers find specific member objects.

Although pure object-oriented DBMSs have not gained widespread popularity comparable to that of relational DBMSs, the object-oriented paradigm has nevertheless had a significant impact on database management. Recognizing the benefits of the object-oriented approach, several major DBMS vendors have created best-of-both-worlds type hybrid solutions by including object-oriented features in their RDB products. There are also object-to-relational mappers that provide an OODB-style interface to applications while using a relational DBMS to store the state of persistent objects.

Object-oriented thinking has also affected the way relational databases are modeled. The standard relational modeling method is to draw entity-relationship (ER) diagrams (Chen 1976), where the data domain of a database is represented by entities (domain concepts), relationships that connect entities, and attributes that characterize entities and relationships. There is a set of rules for deriving a database structure directly from an ER model, and several RDB modeling tools are capable of doing this automatically.

One of the shortcomings of the ER method is that it does not acknowledge inheritance, a very important concept in object-oriented SE. With the introduction of inheritance, the monolithic entities of pure ER models are replaced by nested hierarchies of supertypes and subtypes. This is a useful concept even when using a non-object-oriented DBMS, but ER notation provides no means for representing it. The enhanced entity-relationship (EER) modeling method addresses this problem by including all ER concepts as a subset and adding new ones for expressing inheritance relationships and other object-oriented concepts such as aggregation (Connolly & Begg 2005). Additional rules have been devised for translating supertype-subtype hierarchies into relational table structures.

Document-oriented databases are the latest type of database to gain significant popularity. XML databases such as Sedna (Fomichev *et al.* 2006) use the XML document format to store data and provide query languages such as XPath (Berglund *et al.* 2010) or XQuery (Boag *et al.* 2010) for searching and manipulating document collections. XPath and XQuery, like the XML format itself, are standards developed by the World Wide Web Consortium (W3C). Even more recently, database systems employing less syntax-heavy document formats such as JSON have increased in popularity, e.g., as database backends for web-based applications. Couchdb (Apache Software Foundation 2012), for example, stores and manages data in the form of JSON documents and

provides an API based on the REST paradigm that allows the database to be queried via HTTP requests.

2.3.3 Knowledge representation

Above, in the discussion concerning the data-knowledge distinction, it was stated that digital knowledge is basically a special category of digital data. After that, in the subsection on database systems, we saw that there are several powerful solutions available for the management of digital data. These two facts put together would seem to suggest that the management of digital knowledge should not pose a great difficulty.

In a certain sense, this is true: given the multitude of available options, it is more than likely that a satisfactory storage solution will be found for any body of data. However, it is not the actual storage that makes knowledge representation difficult. Translating knowledge into a digital dataset is the real challenge here.

The problem of knowledge representation is akin to the classic beginner programmer's problem: one knows what one wants the computer to do, but the computer does not understand. The only way to make it understand is to break down the solution into primitives, operations so simple that their meaning can be precisely defined on the execution platform without relying on other concepts. In practice, this means expressing the solution in terms of operands, operators, conditions, loops and standard library function calls when using a modern imperative programming language.

Similarly, it is relatively easy to teach another person something one knows, but much more difficult to teach it to a computer. This is because natural language is a highly effective way of communicating knowledge, thanks to its high information density — a single word is enough to communicate a complicated abstract concept in such a way that the listener immediately knows what the speaker is talking about. It takes many years for a human individual to build up the necessary cognitive skills and background knowledge, but once fully developed, this system is far beyond our current ability to teach knowledge to computers.

Incidentally, one usage of the term "knowledge base" refers to systems that are essentially databases containing passages of text in natural language, possibly enhanced with multimedia. Such systems can indeed be said to contain knowledge in the same sense that a book contains knowledge. In other words, the content has meaning, but only to a human interpreter with the required background knowledge. While content search and presentation in these systems are interesting issues themselves, the central problem

of digitizing knowledge is circumvented, so this is a relatively uninteresting category in the context of this thesis.

The problem with knowledge representation by means of natural language, from a computing point of view, is that as a result of being very well suited for conveying information to a human, it is very poorly suited for conveying it to a machine. We are therefore primarily interested in knowledge encodings designed to be machine-readable. A popular approach is to employ an ontology of some kind.

In philosophy, ontology is the metaphysical study of being, dealing with such fundamental questions as what it means for an entity to exist. In computer science, however, the term has been adopted to refer to a systematic description of a set of concepts and relationships between them (Chandrasekaran *et al.* 1999). An ontology is expressed in a formal notation that allows automated or computer-assisted reasoning about the domain represented by the ontology. A medical ontology, for instance, might represent knowledge related to various diseases and could be used as a diagnostic aid (Rodríguez *et al.* 2009).

To construct an ontology, one typically uses an ontology language; early examples include F-Logic (Kifer & Lausen 1989), CycL (Lenat *et al.* 1985, Lenat & Guha 1991) and KIF (Genereseth & Fikes 1992). These are usually declarative, as opposed to imperative, which means that the languages are only used to describe the ontology, not to specify computing operations on the ontology. They are thus analogous to SQL, which in its original form is a pure declarative language, although there are extensions (both standard and proprietary) for procedural programming that follow the imperative paradigm instead.

As one would expect for a field much younger than relational database management, there is no single established standard language for knowledge representation, although there have been efforts towards standardization. KIF, short for Knowledge Interchange Format, failed to reach this stage, but a later contender, the Common Logic (CL) framework, was adopted by ISO in 2007 (ISO 2007). Particularly interesting in this respect are technologies such as RDF (Klyne & Carroll 2004) and OWL (W3C OWL Working Group 2009), which are being promoted by the W3C as KR languages for the Semantic Web.

The Semantic Web is a proposed machine-readable extension to the currently existing human-readable World Wide Web. The intent of the proposal is to make it possible to represent knowledge on the Web such that software agents can access and use it to provide more sophisticated services to human Web users than current technology

allows. This would be accomplished by using semantic notation to create descriptions of Web content and attaching them to human-readable pages as machine-readable metadata.

The intriguing thing about the Semantic Web is how it would popularize knowledge representation. The idea of KR-based services being available to Internet users as universally as the regular Web is today seems almost revolutionary in comparison with the current state of the art, in which KR applications are specialized and relevant only to a relatively small number of people working in knowledge-intensive professions. However, although several technologies for the Semantic Web exist already, it remains to be seen when the vision will be fully realized.

Another ambitious effort in this area is the development of so-called commonsense knowledge bases. These aim to be comprehensive machine-readable collections of standard everyday knowledge, giving computers access to the body of knowledge possessed by a typical human being. A computer system armed with such a knowledge base could then, theoretically, reason about any matter at the level of a human being. (Davis 1990)

This idea, taken to its logical conclusion, brings us very close to the concept of universal AI, a notorious tangle of technical and philosophical difficulties. We shall take a closer look at AI in the next subsection, but at this point, it is worth noting that while a truly intelligent and knowledgeable computer is still an unrealized dream, there are some remarkably good approximations already in existence. In a recent example, the Watson system (Ferrucci *et al.* 2010) created by IBM defeated two human champions at the game show *Jeopardy!*.

IBM had previously won fame when its Deep Blue chess computer beat the Russian grandmaster Garry Kasparov in 1997, but the nature of *Jeopardy!* makes it a much more difficult game for a machine to play competently. The strict formality of chess is in stark contrast with the quiz show format of *Jeopardy!*, which requires the computer to employ natural language processing and probabilistic reasoning to evaluate the clues. Additionally, *Jeopardy!* adds an extra twist to the conventional format by giving the players answers to which they must give the right questions, not the other way around. Despite these difficulties, which would seem to put the machine at a disadvantage, the two human players lost the game to Watson by a wide margin.

The knowledge base used by Watson in the match against its human opponents consisted of a vast collection of natural-language texts, including, e.g., a complete copy of Wikipedia. Knowledge representation, in the sense discussed here, was therefore not involved: the software used probabilistic analysis of human-readable data to determine

the most likely correct result. From this, it follows that while Watson's knowledge look-up algorithm is certainly impressive, the results mean nothing to Watson itself. Nevertheless, it is interesting to contemplate the idea put forward by IBM of using software based on the Watson query engine to find answers from professional literature to questions posed by experts in fields such as medicine.

Since the overall topic of the thesis is knowledge discovery, it makes sense to conclude this section by reviewing representation schemes designed specifically for knowledge generated through KD. Some systems, such as the chemical reaction KB of Wang *et al.* (2001) and the cerebrovascular diseases KB of Včelák *et al.* (2010), use ontology-type structures to represent the discovered knowledge. Others favor a rules-based approach: Fan *et al.* (2001) propose using decision logic to represent the rules, and the PYTHIA-II system (Houstis *et al.* 2000) supports several types of rules, including logic rules, if-then rules and decision trees. The web mining KB of Velásquez & Palade (2007) is composed of two repositories, one for patterns discovered from web data and another for inference rules that specify how to utilize the patterns.

Of the examples cited above, the KR scheme put forward in this thesis is closest to the last one in that it also uses a separate repository of programs that perform operations on the content of the main repository. The most obvious difference is that the programs in the KB system presented in the thesis are considerably more complex than the if-then rules of the web mining system. The thesis argues that this style of dynamic knowledge representation is more natural for KD than static KR based on ontologies, since knowledge acquired using KD is typically procedural rather than descriptive, representing a method for achieving a specific objective in a specific situation. Furthermore, it is suggested that the question of whether a computer-based system knows something can be considered equivalent to the question of whether the system is able to apply the knowledge independently. Subsections 4.1.2 and 6.1.2 discuss a KR scheme and KB system based on this idea.

2.4 Philosophy of computing

To understand what philosophy of computing is and why it matters, let us consider the academic study of philosophy and the diversity inherent therein. There is philosophy as a standalone discipline, but there are also infinitely many flavors of "philosophy of X", where X is some human endeavor. These philosophies of X concern themselves with such questions as "What is X?", "Why do we do X?" and "How should we do X?".

Philosophy of computing, then, explores these questions in the context of the digital computer, its possibilities and limitations. What is and what is not a computer? What can and what cannot be computed? How do computers and computing relate to other aspects of humanity? These are all questions that can and should be studied primarily outside the confines of any physical manifestation of the concept of computer.

As the study of computing has progressed, certain areas of it have attracted more attention from philosophers than others. The study of AI, for instance, has been and is the source of much philosophical debate, owing to the controversial nature of the notion of a machine possessing intelligence, perhaps even consciousness. KD has close ties with AI, and the philosophical questions pertaining to KD bear a strong resemblance to those surrounding AI research.

Parts of the thesis discuss KD from a philosophical perspective. To provide background for these discussions, literature on philosophy of computing is reviewed in this section. Subsection 2.4.1 establishes an overview of philosophy of computing as an academic discipline by examining its history and branches on a general level. Subsection 2.4.2 focuses on computer ethics, arguably the most important branch from a practical point of view, and also an important factor in KD as the thesis will show. Subsection 2.4.3 takes a closer look at the ethical aspect of KD, and also gives a brief overview of other branches of philosophy that are relevant to KD.

2.4.1 Origins and branches

Although it may not be immediately obvious, the two disciplines discussed here — philosophy and computing — have something important in common: logic. Logic, the study of argumentation and reasoning, is both a major branch of philosophy and a major component of the theoretical foundation of the computer, and the founders of computer science as an independent discipline were logicians. Some of the questions studied by computing pioneers such as Alan Turing were, in fact, deeply philosophical in nature, so arguably philosophy deserves to be counted among the parent disciplines of computer science as much as mathematics does.

Turing, among other early computer scientists, took a keen interest in the idea of a thinking machine. In his seminal paper on the subject (Turing 1950), he proposed a test based on an imitation game played by a computer against a human player. Another human assumes the role of an interrogator who communicates with both players via typed messages and tries to tell them apart based on their communications only. If the

interrogator fails, the computer wins the game and passes the test, which has come to be known as the Turing test.

Turing motivated his paper by pointing out that it is impossible to give a simple answer to the question "Can machines think?" because the terms "machine" and "think" are ambiguous and hard to define in a manner that would satisfy everyone. His reaction to the intractability of the original question was to replace it with a less ambiguous one, namely "Is it possible to build a computer that consistently passes for a human in the imitation test?" If the answer is yes, then the computer is capable of demonstrating, if not true intelligence, at least something that for some practical purposes is equivalent to intelligence.

Although Turing specifically sought to eliminate the inherent philosophical complexity of the question of whether thinking machines are possible, there is no denying that even his proposed replacement question has profound philosophical implications. Generalizing from the basic set-up of the Turing test, one might well ask whether there is any real difference between intelligence and the ability to imitate intelligence so well that no outside observer can tell the difference. The actual computers Turing worked with were extremely primitive by modern standards, but he and his contemporaries had the foresight to realize that given enough processing power and memory capacity, this thought experiment could be turned into a real working machine.

We shall return to the notion of intelligent machines and its philosophical connotations in the next subsection, which discusses computer ethics. First, however, we shall briefly review other major lines of investigation within philosophy of computing. These can be roughly divided into two categories: some focus on what is philosophically interesting about computers and computing as such, whereas others are more concerned with the philosophical aspect of the effects of computers on other things.

A prominent area of study in the first category is philosophy of computer science. Philosophy of science in general is a major sub-discipline of philosophy, studying fundamental questions concerning the nature of the scientific method and scientific knowledge. Philosophy of computer science studies these questions in the context of computer science specifically, starting with the question of whether computer science is a science at all or just a branch of engineering (Hartmanis 1993, Loui 1995, Denning 2005).

Another aspect of computing that interests philosophers because of what it is rather than what it causes is the relationship between the physical and the virtual. Virtual objects in computer memory have dual modes of existence: they have a manifestation in

the physical world — a configuration of electromagnetic bit states, fairly unremarkable in itself — but at the same time, they exist in the virtual world in a way that mirrors the relationship between physical objects and the physical world. The tension between these two modes spawns questions such as whether computer programs that mimic life forms should be considered life if they satisfy the criteria commonly used to define biological life (Bedau 2008). Aesthetical issues, such as the debate on whether programmatically generated artworks should be considered proper art, can also be included in this category (Lopes 2008).

In the second category, a topic that has received considerable attention from researchers is the role of computers in philosophy education and research (Bynum & Moor 1998). In contrast to the other topics reviewed here, the focus of this work is not on what is philosophically interesting about computers specifically, but rather on how computers can advance the study of philosophy in general. This emphasizes the nature of philosophy of computing as an intersection discipline that covers a relatively wide range of topics, defined mainly by the property that they are somehow relevant to both philosophy and computing.

The effect of information and communication technologies on philosophy education and research is a special case of the broader phenomenon of ICTs affecting the acquisition and dissemination of information in general. The increasing capabilities of computers are a powerful driver of change in society and therefore a rich source of phenomena that call for philosophical analysis. Being able to predict and control the ill effects of this change is especially important, so the triumph of the digital computer has given rise to a new branch of applied philosophy: computer ethics.

2.4.2 Computer ethics

Aristotle famously defined ethics as practical philosophy, the art and science of living the good life (Broadie 1993). Since we all surely want to have a good life, then by this definition, a good knowledge of ethics is of great practical significance to everyone. On the other hand, the musings of academic ethicists may sometimes seem somewhat removed from real life, which would appear to contradict their alleged practical importance.

This apparent contrast can be explained when we recall that it is in the nature of philosophy to anticipate future issues. Such concepts as representative democracy, civil liberties and free-market economy — all of them hallmarks of modern Western

societies — were originally thought up by philosophers, some of them centuries before they became a reality. Similarly, even though some of the ideas being discussed by today's philosophers may not seem very relevant to us now, the time may yet come when they are a crucial part of what is widely considered to be good life.

To illustrate this point, let us consider the ethics of AI. At the moment, we do not know whether we will ever be able to create machines with true intelligence, let alone consciousness, and there are many who are quite convinced that the answer is no. Many therefore probably consider it a waste of resources to think and talk about what rights and responsibilities we should assign to such machines. However, if it ever does turn out that the answer is yes after all, it is certainly better that somebody has already given such questions due consideration than to start creating policies on AI without any research whatsoever to rely on.

The AI example shows why it is necessary to study computer ethics. The processing power and other capabilities of computers continue to increase at an astonishing pace, and with the capabilities expands the space of potential applications. The application space contains many ethically unsound possibilities, which computer ethics research can help predict and restrain, but perhaps even more importantly, there is a gray area between acceptable and unacceptable where our everyday ethical instinct fails us. Computer ethics has an important role in determining what stance we should adopt on these applications of computers. The remainder of this subsection shall explore some of the most prominent themes in the study of computer ethics.

Malicious behavior facilitated by computers is an obvious research topic for computer ethics. Actions that are clearly wrong such as electronic theft of money or wilful destruction of data are one aspect of this, but from a philosophical perspective, this category is not particularly interesting. However, with some forms of behavior, it is harder to define why they should be considered morally questionable. A classic example is breaking into an information system without touching anything — hackers who do this argue that they are not doing any harm, or even that they are doing a valuable service by exposing weaknesses in system security (Hollinger 1991, Spafford 1992). Another source of controversy is the use of cyber-attacks as a means of pursuing political goals, a phenomenon known as hacktivism (Manion & Goodrum 2000).

Another interesting aspect of malicious activity is that while computer networks enable new types of attacks, they also enable ways for the targeted people and organizations to respond to them. There are various examples of creative self defence and even vigilantism on the Internet, presumably stemming from the insufficient ability of the

proper authorities to deal with the cybercrime problem. For instance, the activity known as scam baiting, which has evolved as a countermeasure to email scams, shares many properties with traditional vigilantism and is ethically problematic for the same reasons (Tuovinen & Röning 2007).

The ability of malicious individuals to execute new kinds of attacks — as well as the ability of technically competent net users to defend themselves — is an example of the more general pattern that information and communication technologies tend to change the rules we expect to be in effect in our dealings with other people. In the case of cybercrime, ICTs enable the perpetrators to evade traditional means of control. At the same time, however, the same technologies enable new forms of control, leading to ethical problems related to privacy and surveillance.

In the physical world, surveillance by technology is typically done for security purposes using equipment such as closed-circuit television cameras and motion sensors. This is generally not considered objectionable; a department store, for instance, is thought to be justified in using such means to protect its employees and property from crime. Meanwhile, the same store may be using tracking software to monitor the browsing behavior of its online customers, not for protection but to increase its profits, e.g., through more precisely targeted special offers (Schafer *et al.* 2001). Even this is usually not considered particularly invasive and may even be praised by the customers as improved service, but there are other, less benign ways to earn money with personal information.

The most talked-about example of this at the moment is probably Facebook. The popular social networking service is provided free of charge, so the company relies almost exclusively on advertising for revenue. Advertisers are willing to pay for the ability to reach specific focus groups, making the users' personal information valuable merchandise. It is therefore in the interest of Facebook that the users reveal as much information about themselves as possible. Consequently, the company has been criticized for failing to appreciate the privacy concerns of the users of the service. On the other hand, there are also indications that many users are not particularly concerned about their privacy and place themselves at risk through their own lax attitude rather than the unethical behavior of Facebook or a third party. (Jones & Soltren 2005, Debatin *et al.* 2009, Boyd & Hargittai 2010)

A major factor in why ICTs and privacy are such a difficult combination is that traditional conceptions of privacy, derived from concrete and intuitive notions such as freedom from observation, fail when transferred to the abstract world of information.

Introna (1997) offers a comprehensive discussion of the significance of privacy in the information society; Clifton *et al.* (2002) do the same in the context of knowledge discovery specifically. Privacy issues associated with KD are discussed further in the next subsection.

The failure of conventional ways of thinking to capture complications introduced by new technology is a recurring theme in computer ethics. In the context of intellectual property, the complications arise from the fact that an indefinite number of copies can be made of a digital object quickly, inexpensively and without loss of quality. Moreover, the copies can also be distributed worldwide quickly and inexpensively via the Internet. This appears to be a challenge that established copyright principles, formulated under the assumption that creative works are distributed on physical media that are relatively costly to copy, are unable to cope with. Hettinger's paper on the subject (Hettinger 1989), while not recent, captures the difficulty of defining and justifying intellectual property well.

Given the significant financial interests associated with intellectual property, it is hardly surprising that the topic has been explored intensely, also by philosophers. However, not all research on the subject start with the premise that authors want to earn profit with their creative works. Work on topics such as open source software and free licensing models (e.g., Creative Commons) emphasizes sharing instead, either of the creative process or of the results. Privacy is also an important theme here, as demonstrated by the outcry that followed the discovery of a secret rootkit in Sony's digital rights management software (Mulligan & Perzanowski 2007).

Social networking was mentioned above in the context of privacy issues, but the broader theme of life online spans a number of other topics. For example, trust in online interactions is an important issue that is also relevant to certain forms of knowledge discovery, as discussed in Subsection 4.2.2. The digital divide and the ability of individuals to participate in society's decision-making processes via ICTs is another major topic (Bélanger & Carter 2009, Helbig *et al.* 2009). Somewhat more marginal, but nevertheless interesting, are issues such as antisocial behavior in online multiplayer video games (Foo & Koivisto 2004, Ku & Gupta 2008).

Most branches of computer ethics deal, in one way or another, with the responsible application of information and communication technologies. However, it is also recognized that creators of ICTs, like all engineers, are professionals with obligations to their employers, to their clients, and to society. One branch of computer ethics is therefore devoted to clarifying the responsibilities of computing professionals and

creating ethical guidelines for them to follow. The concrete outcome of this work is exemplified by the software engineering ethics code adopted jointly by the ACM and the IEEE Computer Society (Gotterbarn *et al.* 1999).

2.4.3 Philosophy of knowledge discovery

"Knowledge is power", goes a famous saying; "Power corrupts", goes another. While the latter saying is perhaps too pessimistic as an unconditional statement, the former is certainly true given the classical definition of power as the ability to achieve one's goals, either through one's personal actions or through influence on the actions of others. In either case, possessing the right knowledge can be of great instrumental value.

Regrettably, it is also true that those with power, from knowledge or otherwise, sometimes use it unethically. Even more sadly, it is sometimes possible to get away with this because the world is changing so fast that our ethical and social norms are unable to keep pace with the change. A major aspect of this change, from an ICT point of view, is the increasing wealth of digital information available in various databases and the similarly increasing variety of potential ways of using that information. Particularly problematic from the perspective of ethics is information representing properties or behavior of human individuals.

In a modern technological society, it can be very difficult, if not impossible, for individuals to keep track of, let alone control, what information is being recorded about them, who is recording it and for what purpose it is used. Fortunately, governments have picked up on the privacy concerns arising from this bewilderment and implemented legislation to alleviate them. In Finland, for example, the Personal Data Act of 1999 regulates the collection, maintenance and use of records containing personal data and prohibits the inclusion of sensitive data in such records (Finnish Ministry of Justice 1999). A special independent authority, the Data Protection Ombudsman, is appointed to protect citizens' rights concerning their personal data.

The protection of sensitive data is made more difficult by the fact that even data that is entirely non-sensitive, when each data item is considered in isolation, may yield sensitive information when processed using methods capable of discovering hidden connections and patterns (Clifton & Marks 1996, Tavani 1999). Knowledge discovery methods do just that, prompting concerns for their potentially detrimental effect on privacy. This has not escaped the attention of researchers, and privacy issues are now a

central topic in the study of the ethical aspect of KD and the anchor point for most lines of investigation.

The field of privacy-preserving data mining was already touched upon in Subsection 2.1.3. Here, the objective is to develop techniques for mining personal data that are able to discover interesting patterns pertaining to a population while not exposing sensitive information traceable to individual members of the population. A variety of methods are available for accomplishing this: data can be protected through hiding or modification, and the hiding or modification can be implemented in the mining algorithm or in the data itself (Verykios *et al.* 2004, Bertino *et al.* 2005, Zhang & Zhao 2007). The purpose, however, is always to achieve a satisfactory trade-off between the level of privacy protection and the quality of mining results.

Security-related applications of KD form an interesting subclass here. These, in many cases, must invade the privacy of the data subjects in order to be successful, but on the other hand, it is argued that the invasion is morally justified by the security concern addressed by the application. This is a classic case of having to weigh one value against another, often without accurate knowledge of how much one will be boosted or how severely the other will be breached.

Terrorism prevention applications such as airline passenger profiling by DM (Thuraisingham 2002, Solove 2008) have been the target of particular media attention, leading to a somewhat imbalanced public perception of what DM is. A group of executives of the ACM SIGKDD eventually thought it necessary to write an open letter to correct this perception (Kim *et al.* 2003). The letter points out that person profiling is but one of many diverse applications of DM, so it is completely unreasonable to treat the term as if it were synonymous to privacy-threatening government projects.

Indeed, it is good to keep in mind that ethics is about what is good and right just as much as it is about what is bad and wrong. Discussion of the ethics of KD should therefore consider not just the ethically questionable applications of KD, but also the ethically commendable ones. It is good to think about how to reduce the potential negative effects of KD, but we should also be proactively thinking about how to use KD to advance positive values.

(Tuovinen & Röning 2005) looks at KD from both perspectives, using the terms "negative ethics" and "positive ethics" to refer to the harmful and beneficial effects of KD, respectively. Several application examples of both the negative and the positive ethical aspect of KD are cited in the paper. Medical DM forms a notable class of applications on the positive side; DM techniques can be used, for instance, for early

diagnosis of various diseases, leading to significant improvements in prognosis and quality of life for the patients.

Other topics related to the philosophical aspect of KD are not as popular as ethics. This does not mean, however, that there are no other philosophically interesting questions related to KD. Knowledge is a highly significant concept in philosophy and a branch of philosophy, epistemology, is dedicated to studying it. In this thesis the epistemology of knowledge discovery is an important theme, discussed at length in Sections 3.1 and 4.1.

There are yet some other branches of philosophy that intersect with KD in ways that merit further attention. In philosophy of science, the implications of using KD as an instrument of scientific knowledge acquisition could be investigated. Using text mining to analyze philosophical literature and thus applying KD to philosophy itself is another topic that might be worth studying. Finally, recalling that logic and computing are closely related to begin with, the intersection of logic and KD might prove a fertile research area. All these questions are, however, out of scope for this thesis.

3 Analysis of requirements

The objective of this chapter is to define a set of requirements that the solution proposed in the next chapter should satisfy. The requirements are derived from an analysis of the current state of research and practice in the field of knowledge discovery and related disciplines. Based on the analysis, aspects of the KD process that have been neglected in the past are identified and translated into formal requirements. Each individual requirement is given a shorthand designation to be used when referring to the requirement in body text. In Chapter 5, once the proposed solution has been presented, the requirements will be used as a test to show whether and to what degree the solution achieves the goals set for it.

The augmented process model was defined in Chapter 1 as being composed of three sub-models: the data model, the workflow model and the architectural model. The remainder of this chapter is structured to reflect this division. Section 3.1 defines requirements for the data model, which specifies how data and knowledge are stored and accessed in the KD process. Section 3.2 does the same for the workflow model, which is concerned with the actors of the KD process and their roles and responsibilities. Finally, Section 3.3 presents requirements for the architectural model, which deals with the design and implementation of KD software.

3.1 Data requirements

At the heart of the KD process is the repeated transformation of data from one state to another, with each transformation taking the data to a higher level of refinement. The objective of the data model is to capture these different states, or categories, of data, and to guide the design and implementation of storage solutions and access interfaces. The model is intended to be decoupled from any particular type of data repository, although the concrete implementations discussed in the thesis exhibit a fairly strong preference for relational databases.

One aspect of the data model is the management of data over the course of the refinement process. Storing the data is, in itself, largely a trivial issue, but the transformations add a metalevel that makes the matter more complicated. Through modification of transformation parameters, any number of transformation sequences can result from the

same original data. Tracking the data along the path of transformations is one of the major problems that the data model should address.

As discussed in Subsection 2.3.1, this thesis adopts the position that knowledge is a special case of data, an extreme point along a continuum where untreated data represents the opposite extreme. It is therefore argued that a single unified data model can and should cover knowledge representation in addition to knowledge generation. The model should clarify the concept of knowledge beyond what was said on the topic in the background chapter; following that, we can proceed to study the problem of digitally representing knowledge as the concept is understood in the context of KD.

The requirements for the data model are collected in Table 1. In the subsections that follow, each requirement is given a more detailed description and justification. Requirements pertaining to data management for knowledge generation are described in Subsection 3.1.1, and those pertaining to knowledge representation in Subsection 3.1.2.

Table 1. Data requirements for the augmented KD process model.

Req no.	Description
D-01	The data model should provide a conceptual framework that identifies and characterizes each state that data passes through during the KD process.
D-02	The conceptual framework should identify and characterize each transformation that causes data to shift from one state to another.
D-03	The data model should provide technology-independent guidance for the creation of storage structures that cover all the identified states of data.
D-04	The guidance for the creation of data stores should address the design of access methods for storing, searching for and retrieving data.
D-05	The data model should establish criteria for validating the knowledge generated by the KD process.
D-06	The validation criteria should be based on a theoretical understanding of the concept of knowledge.
D-07	The data model should provide a KR scheme for the creation of a knowledge base.
D-08	The KR scheme should share a theoretical basis with the other components of the data model.
D-09	The KR scheme should integrate with and satisfy the same requirements as the data management scheme specified by D-03 and D-04.

3.1.1 Data management

D-01: The data model should provide a conceptual framework that identifies and characterizes each state that data passes through during the KD process.
D-02: The conceptual framework should identify and characterize each transformation that causes data to shift from one state to another.

These requirements address the theoretical foundation of data management in the KD process. The rationale for them is that in order to gain a clear and comprehensive view of the data management needs of the process, the data model should begin with a formal description of the process from a data perspective. The description will serve as a fixed point of reference that connects the data model to the overall KD process model.

Data in the KD process has a static aspect and a dynamic aspect. The different states that data goes through over the course of the process represent the static aspect. The transformations that shift data from one state to another constitute the dynamic aspect. Data management involves both aspects: the data needs to be stored in its different states, and it needs to be tracked along its progression through the states. Therefore, both the states and the transformations should be identified and formally described by the data model.

D-03: The data model should provide technology-independent guidance for the creation of storage structures that cover all the identified states of data.
D-04: The guidance for the creation of data stores should address the design of access methods for storing, searching for and retrieving data.

These requirements address the basic data management functions necessary for the application of KD algorithms to data. For the data analysis to begin, the original raw data needs to be stored in a database. Furthermore, it is desirable to have the ability to store intermediate results, because it enables the analysis process to be restarted with modifications from an arbitrary point along the sequence of transformations (Laurinen *et al.* 2004b). The data model should therefore assist in the design of a data repository capable of storing data in all of the states identified by the model.

A data repository consists of persistent data structures and interfaces for accessing and manipulating the structures. The data model should support the creation of both the data structures and the interfaces, but in an abstract manner such that there is no preference for a particular vendor or a particular type of database. Instead, the model

should provide a generic design template that can be applied regardless of the underlying DBMS. Technology-specific implementations can be included in the architectural model, however.

> *D-05: The data model should establish criteria for validating the knowledge generated by the KD process.*
> *D-06: The validation criteria should be based on a theoretical understanding of the concept of knowledge.*

These requirements address the target of the KD process, knowledge. The reason for their inclusion is that in order to conclude the KD process, there needs to be a test to determine whether the target has been achieved. Since the data model treats knowledge as the ultimate state of refinement for data, it should propose such a test: a set of criteria that can be used to judge whether a given body of data may legitimately be called knowledge.

In order to establish such criteria, it is first necessary to give a proper definition to the concept of knowledge. This is a heavily analyzed concept in philosophy, a fact that the data model should make use of: the model should first establish a theoretical basis by reviewing existing definitions of knowledge as a general concept. It should then study those definitions in the context of KD in order to discover how philosophical criteria for knowledge translate into criteria that can be applied in the KD process.

Epistemology, or theory of knowledge, is the branch of philosophy that studies knowledge. In classical epistemology, knowledge is defined as justified true belief. This definition is often attributed to Plato, but it is doubtful whether Plato himself originated it, nor is it known for certain whether he endorsed it. The definition has also been attacked by later epistemologists as incomplete, but even though it is not universally accepted by philosophers, it does give us three useful conditions:

– Truth: false knowledge is considered a contradiction in terms.
– Belief: the one in possession of a piece of knowledge must believe it to be true.
– Justification: the belief must be supported by a convincing rational argument.

In a human context, these concepts are all intuitively understandable. In the context of KD, however, it is much less clear how they should be defined to make them useful as evaluation criteria. The data model should therefore examine each concept more closely to make explicit the implicit epistemology of KD.

3.1.2 Knowledge representation

D-07: The data model should provide a KR scheme for the creation of a knowledge base.
D-08: The KR scheme should share a theoretical basis with the other components of the data model.
D-09: The KR scheme should integrate with and satisfy the same requirements as the data management scheme specified by D-03 and D-04.

These requirements address the creation of a digital repository of knowledge generated by KD. To facilitate this, the data model should include a KR scheme suitable for such a repository. Keeping in mind the premise that knowledge is a special case of data and should be managed by the same unified data model, the KR scheme should be rooted in the same definition of knowledge as the other components of the data model. Studying the epistemology of KD, as required by D-06, is therefore a prerequisite to the KR scheme.

For the KR scheme to integrate smoothly with the data management scheme, the two should be roughly equivalent in terms of both content and presentation. In other words:

- Like the data management scheme, the KR scheme should address the data structures in which the knowledge is stored and the interfaces by which the knowledge is accessed and manipulated.
- Like the data management scheme, the KR scheme should be presented in an appropriately abstract manner such that the presentation does not bind the implementation of the scheme to any particular vendor or technology.

Ideally, the data management and knowledge representation aspects of the data model should be implementable as a single integrated repository using whichever database technology is the most convenient one for the organization carrying out the KD process. The organization would thus not need to migrate to a new database platform or deploy one alongside the one it is already using.

3.2 Workflow requirements

A process model, in order to be useful, should be neither too imprecise nor too precise: it has to be precise enough to eliminate uncertainty on how to proceed in a given situation, but not so precise that it becomes impossible to grasp the process as a whole. Any useful

process model is therefore necessarily imperfect in the sense that it cannot accurately capture every single detail of the process it represents. Given this, one could well ask whether the process model of KD is not good enough already, since perfection is a meaningless goal here and thus the idea of improving the model indefinitely does not make sense. This thesis aims to show that there are aspects of the process model that could indeed be improved, and to suggest how an improvement can be attained.

The established KD process model is basically a workflow model, a set of steps that starts with a quantity of relatively cheap and crude material, which it attempts to reduce to its valuable essence or core. Before we can begin to discuss requirements for an improved workflow model, it is first necessary to present a more detailed description of the established model than the brief and informal one given in Chapter 1. We examine first a more exclusive version of the model, which only covers the series of transformations applied to data in order to refine it into knowledge, and then a more inclusive version, which covers the entire development process of a KD solution, starting with a description of what the current situation is in the application domain and how it should be improved.

When considered in the more exclusive sense, the KD process begins with the preprocessing of raw data. The preprocessing phase typically involves operations such as restructuring the data and fixing or removing data items with missing or erroneous values. The objective of preprocessing is to normalize the form of the data so that its content can be analyzed; the operations performed in this phase do not extract any information from the data but only prepare it for further transformations.

The next step in the KD process is extracting features from the preprocessed data. Feature extraction is essentially a form of lossy compression: the objective is to lose some of the original data in a controlled manner such that the resulting feature set represents the important characteristics of the original dataset. If the KD effort is to succeed, the feature set has to be comprehensive enough to capture all the relevant patterns in the distribution of the original data, yet compact enough for it to be computationally feasible to build a model of the data based on the features.

In the third step, a modeling algorithm is applied to the data items of the feature set. The purpose of the algorithm may be, for instance, to divide the data into classes or clusters such that data items that are close to each other in feature space are grouped together. The modeling algorithm thus yields a view of the original dataset that rises above the level of individual data items, showing substructures defined by groups of data items.

The final step of the process is to visualize and interpret the substructures exposed by the modeling algorithm. This enables domain experts to assess the significance of the results of the process. If they find that the results answer the original questions they set out to answer, then it is concluded that the KD effort was a success and valuable new knowledge was generated. If the results are inconclusive, the experts may choose, e.g., to gather additional data for analysis.

Defining the questions to be answered is the first step of the KD process in the more inclusive sense. This is done by experts who must be aware of both what is already known in the application domain and what is achievable using current technology. With the aid of this knowledge, they can define a goal for the KD effort such that the effort is feasible to carry out and advances the state of the art sufficiently to justify the investment.

Once the experts have defined what is to be achieved, they must next determine how it can be achieved. This involves determining what datasets are required, where they can be acquired and how they should be analyzed in order to arrive at the desired result. Once the step is complete, the experts have a plan for preparing and executing the KD process (in the more exclusive sense).

Carrying out the plan begins with procuring the required datasets, which may come from a number of different sources. Some of the required data may already be in the possession of the organization executing the KD process, and some may be acquired from other organizations such as project partners or strategic collaborators. It may also be necessary to generate some new datasets, e.g., by performing controlled experiments designed specifically to support the chosen goals of the KD effort.

When all the required datasets have been created and brought together, analyzing the data can begin. It is at this point that the KD process in the more exclusive sense is executed, with one more step to follow once it has been completed: integrating the results with existing knowledge. As suggested later in the thesis, in Subsection 4.1.1, this is the final test of the proposed new knowledge and may yet lead to unexpected conclusions if the results do not seem to fit in with what is already known about the subject of the KD effort.

The CRISP-DM (CRoss Industry Standard Process for Data Mining) process model (Shearer 2000) is a formalization of the knowledge discovery process developed by a consortium founded specifically for this purpose, the (now apparently defunct) CRISP-DM Special Interest Group. On the top level, the model consists of six phases: business understanding, data understanding, data preparation, modeling, evaluation and

deployment. The model proceeds through the phases sequentially, with a few feedback points where it is possible to return to an earlier phase.

Compared with the KD process model as described above, the CRISP-DM model is more business-oriented, as witnessed by the business understanding and deployment phases: the process is driven by the needs of a business and the goal is to get results that can be deployed by the business in a profitable manner. The model described above is more generic, or perhaps more idealistic, in that it simply takes it as a given that knowledge has value. Apart from this, the two models are largely equivalent, although they group the process tasks into phases somewhat differently.

An interesting feature of the CRISP-DM model is that at several points its specification puts emphasis on the roles and responsibilities of human actors in the process: what they contribute to the process and what they expect to receive from other actors. This human aspect of the KD process is a major source of the workflow requirements presented in this section. Another is the structure of the workflow itself, which, it is argued here, is not as linear as the established model implies.

Table 2 presents a summary of the workflow requirements identified. This is followed by more detailed descriptions and justifications of the requirements in the subsections. Subsection 3.2.1 examines requirements concerning process actors, "actor" being used here as a broad term referring to a person, group or artifact with a contribution to the KD process. Requirements concerning interactions among the actors are discussed in Subsection 3.2.2.

3.2.1 Actors

> *W-01: The workflow model should provide a conceptual framework that identifies the types of technological artifacts used in the KD process, and characterizes their roles and relationships.*

This requirement addresses the software tools used to carry out the KD process. Algorithms are the most elementary of KD tools: they perform transformations of data from one state to another. Data transformations thus represent a common ground with the data model, making algorithms a natural starting point for the workflow model.

Algorithms are generally not used as standalone entities. Instead, they are packaged as components and assembled into applications where multiple algorithms work together toward a common goal. Applications, in turn, may exchange data and instructions with

Table 2. Workflow requirements for the augmented KD process model.

Req no.	Description
W-01	The workflow model should provide a conceptual framework that identifies the types of technological artifacts used in the KD process, and characterizes their roles and relationships.
W-02	The conceptual framework should identify the human actors responsible for executing the KD process, and characterize their contributions and expectations.
W-03	The conceptual framework should identify the human actors who otherwise hold a stake in the KD process, and characterize their contributions and expectations.
W-04	The conceptual framework should identify the interactions that take place between human actors, and characterize their significance in the KD process.
W-05	The conceptual framework should identify the interactions that take place between human actors and technological artifacts, and characterize their significance in the KD process.
W-06	The conceptual framework should identify the interactions that take place between technological artifacts, and characterize their significance in the KD process.
W-07	The workflow model should provide guidance for mapping the interactions that make up the KD process onto a flowchart of process steps.
W-08	The workflow model should identify any potential conflicts of interest that may result from interactions between actors in the KD process.

one another, forming systems of multiple collaborating applications. The workflow model should provide a conceptual framework that formally describes this hierarchy of KD technology.

> *W-02: The conceptual framework should identify the human actors responsible for executing the KD process, and characterize their contributions and expectations.*

This requirement addresses the primary drivers of the KD process. Although computing technology is an indispensable instrument for KD, the tasks that have the most profound effect on the outcome of the KD process — setting the goals, deciding the methods, controlling the process, evaluating the results — are performed by human actors. These actors and tasks should therefore hold a central position in the process model.

In order to understand how to support the work of a given actor, it is necessary to understand what the contribution of the actors to the KD effort is and what the actor needs from other actors in order to make the contribution. The conceptual framework of the workflow model should therefore identify the contributions and expectations of each

actor. Based on this information, it is then possible to reason about the most effective way to secure the contribution of a given actor.

> *W-03: The conceptual framework should identify the human actors who otherwise hold a stake in the KD process, and characterize their contributions and expectations.*

This requirement addresses the contributions and expectations of actors who have no immediate interest in the outcome of the KD process and whose goals are therefore of a more personal nature. Here, the case may be that there is no direct dependency between an actor's contributions and expectations, but it is nevertheless important to respect the expectations, lest the primary drivers' relationship with the actor become exploitative. To avoid the ethical issues associated with such relationships, these actors should be identified by the workflow model.

One of the ethical issues that may be raised by KD, privacy, has already been discussed. The workflow model should explore the implications of the privacy issue to the KD process, but it should not focus on privacy alone. A more comprehensive survey of the potential ethical issues associated with KD should be carried out, and the results of the survey should be translated into constraints on the KD process and made part of the KD process model.

3.2.2 Interactions

> *W-04: The conceptual framework should identify the interactions that take place between human actors, and characterize their significance in the KD process.*
> *W-05: The conceptual framework should identify the interactions that take place between human actors and technological artifacts, and characterize their significance in the KD process.*
> *W-06: The conceptual framework should identify the interactions that take place between technological artifacts, and characterize their significance in the KD process.*

These requirements address the nonlinear nature of the KD process, which, the thesis argues, the established KD process model does not adequately capture. To justify this argument, let us first consider a parallel from the field of software engineering: the classical "waterfall" process model. The model has its roots in SE practices of the 1950s

and 1960s, but its modern formulation originated in the 1970s (Boehm 2006). There are several versions of the model, but the basic pattern they all follow is that the development of a software product follows a linear path from solicitation of requirements through specification, design, coding and testing to deployment. As in the KD process, the outputs of each process step become inputs for the next step in the sequence. Thus, the original inputs, consisting of information gathered from the customer, are eventually transformed into the final product.

The most straightforward version of the waterfall model is not a very realistic model to follow in an actual software engineering project, and it mainly serves as an instrument for understanding the variety and hierarchy of tasks that need to be successfully executed in order to create a software product that meets all expectations. To make it useful in practice, the naive, perfectly sequential model requires additions such as feedback loops between process steps to acknowledge the reality that quality software cannot be developed in a purely linear fashion. However, even in its refined form, the waterfall model is somewhat unwieldy, and one of the major themes of SE work in the 1990s and 2000s has been the search for more flexible software development methodologies, with emphasis on concurrency and agility (Boehm 2006).

The lesson here is that while a simplistic linear process model can be useful by virtue of being easy to grasp and follow, it is inevitably rather narrow in perspective, failing to account for those aspects of the process that are not readily reducible into a sequential list of instructions. It is this narrowness of scope that this thesis considers the main weakness of the established KD process model: like the waterfall model in SE, it is a reasonably good top-level description of the lifecycle of a solution, but at the cost of obscuring many important nuances of the actual solution development process.

To illustrate what the established KD process model does not show, we can begin by noting that there is a word for the process of acquiring knowledge: learning. From this perspective, KD can be informally defined as the process of learning from a digital dataset, which is not unlike learning from a textbook. The latter is an inherently interactive process where the learner must constantly try to place the new information in the context of what they already know about the subject, and the interactive aspect is emphasized even further when we consider KD as an act of learning. Knowledge in a book is explicitly stated, but the knowledge in a dataset is implicit in the hidden dependencies and regularities of the data items. The learner is therefore very likely to require the help of auxiliary sources, such as a software application to process the data and a technology expert to show how to use the application and interpret the results.

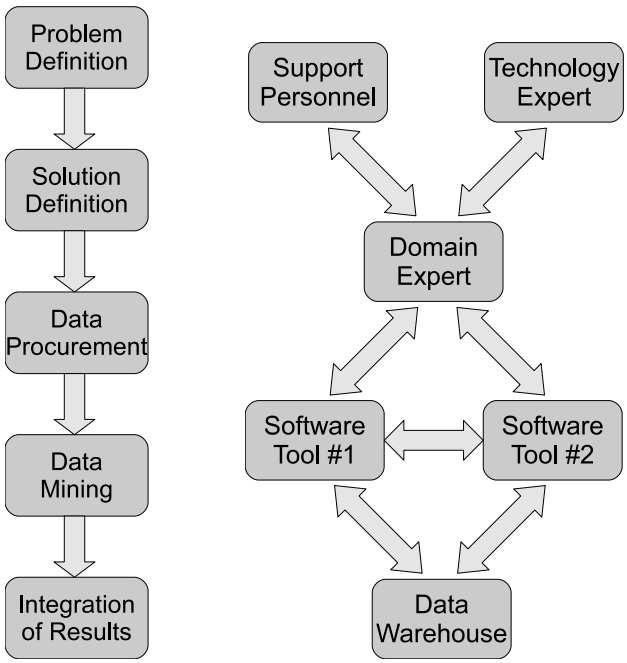
Even this rather simplified scenario shows that the KD process involves diverse interactions: between the learner and the technology expert, between the learner and the software, between the software and the data. In a real KD project, at least some of these entities are likely to be multiplied, so there may be multiple learners (domain experts), multiple technology experts, multiple data processing tools or multiple datasets, increasing the number of interactions. Additionally, the process may involve people or technology acting in roles other than these four.

From this perspective, the traditional path of sequential steps no longer seems an appropriate visual metaphor for the KD process. A graph would be a more apt representation, with process actors as the nodes and their relationships as the edges connecting them. An edge between two nodes would indicate that some kind of interaction, or dialogue, takes place between the actors represented by the nodes. Learning emerges as the overall effect of this potentially highly complex web of interactions. Figure 1 illustrates the difference between this and the sequential KD process model.

The conceptual framework of the workflow model, having identified and characterized the actors of the KD process, should thus then proceed to identify and characterize the interactions among the actors. Actors, by the definition adopted here, may be either humans or technological artifacts, so there are three major categories of interactions that we can discern: those involving only humans, those involving only technology, and those involving both. The current state of KD research and practice is such that interactions involving humans are much less thoroughly explored than interactions involving only technology, so the workflow model should focus special attention on providing a comprehensive description of the roles and tasks of human actors in the KD process.

> *W-07: The workflow model should provide guidance for mapping the interactions that make up the KD process onto a flowchart of process steps.*

This requirement addresses the ability of the augmented process model to guide the execution of the KD process. While the thesis criticizes the established process model for failing to represent important aspects of the process, the model has undeniable practical value by virtue of providing a reasonably unambiguous flowchart for KD practitioners to follow. Something comparable should be provided by the workflow model for the augmented process model to be useful as a reference for carrying out the KD process. Although it is not feasible to present one in this thesis, the workflow model



(a) Sequential model

(b) Graph model

Fig 1. The traditional, sequential model of the knowledge discovery process, emphasizing process steps, versus a graph model of the process, emphasizing process actors and interactions.

should make it possible to create a detailed user guide similar to the one included in the specification of the CRISP-DM model.

> *W-08: The workflow model should identify any potential conflicts of interest that may result from interactions between actors in the KD process.*

This requirement addresses interactions where the expectations of the actors involved are in conflict. When such situations arise, it is important for the actors to be prepared to resolve the conflict, which is why the workflow model should identify cases where conflicts of interest are possible. The requirement continues to emphasize the importance of ethical conduct, but it also has a more pragmatic dimension, which can be illustrated using the privacy example again.

In KD projects where potentially sensitive data is mined, there is always a certain tension between the data miners, who would like to have access to as much data as

possible, and the data subjects, who would prefer to retain their privacy. Assuming that the data needs to be voluntarily provided by the data subjects and that the subjects realize its sensitive nature — and arguably the data miners have a moral obligation to make sure that they do — the miners cannot acquire the data unless they can win the trust of the subjects. If the subjects suspect that their data may be used unethically, they will decline to disclose it to the miners and the project will fail. This is one instance where the success of a KD effort depends on successful conflict resolution.

For a less specific example, consider the case of a KD company bidding for a project. For the potential client, the project would mean, besides entrusting their data to another company, a substantial financial investment as well, so establishing trust is of great importance for several reasons here. A bid has a chance of succeeding only if the client is sufficiently convinced that the bidder will keep the data secure and has the necessary competencies to create value for the client.

Largely similar conditions apply when a research group seeks a grant for an academic KD project, with the funding source (e.g., a foundation) in the role of the client and the research group in the role of the bidder. Grant applications are also typically subject to scrutiny by an ethics review board, which has the authority to demand revisions to the project plan or even reject the entire project. The researchers are thus forced to review the ethical issues involved, even if it is just to reach the conclusion that there are none.

The various examples given above demonstrate that addressing ethical issues is an important KD task that, if done poorly, may even kill a project before it has a chance to start. The reputation of an organization is affected by the way it handles these issues, so the importance of ethical awareness is emphasized in the long run: the more credibility the organization loses as an ethical actor, the less likely it is to be given new opportunities to work on projects involving sensitive data or other ethical issues. To ensure that these issues are handled in a thorough and responsible manner, identifying and addressing them should be made an integral part of the KD process.

3.3 Architectural requirements

Although the thesis argues for a less technology-centric view of the KD process, it is undeniable that technology plays a crucial role in providing support for the process. The aim of the architectural model is to provide such support through a reusable software design that reflects key aspects of the proposed augmented process model. Ultimately,

the design should be expressed in the form of a software framework that can be used to develop and deploy KD solutions in any application domain.

As the survey in Subsection 2.2.3 shows, there are many KD software frameworks in existence already. The new perspective offered here is that the three sub-models of the proposed process model — data, workflow and architecture — should constitute a consistent whole. The role of the architectural model in this arrangement is to bring both the data model and the workflow model under a single software architecture.

The requirements for the workflow model focus on process actors and their interactions. Therefore, in order to accommodate the workflow model, the architectural model should be able to represent the actors and to support the interactions among them. Many of the interactions involve human actors only, and these are deliberately left outside the scope of the architectural model. However, there are various human-technology and technology-technology interactions that warrant more attention than they have received so far. These are taken into account by the requirements for the workflow architecture.

The requirements for the data model deal with data management and knowledge representation. The objective of the data architecture is to integrate these functions smoothly with the workflow architecture. This does not imply, however, that a software framework that implements the workflow model should also implement its own solution for database management. Instead, the requirements for the data architecture presume the existence of an established solution that can be adapted to implement the data model.

The architectural requirements for the augmented KD process model are gathered in Table 3. A more detailed treatment of the requirements is given in the subsections: Subsection 3.3.1 discusses requirements related to the data model, and Subsection 3.3.2 those related to the workflow model. Subsection 3.3.3 looks at general quality requirements that are not specific to either model.

Table 3. Architectural requirements for the augmented KD process model.

Req no.	Description
A-01	The architecture should provide access to data sources via a transparent abstract interface.
A-02	The architecture should assist in the generation of a data repository structure that covers the entire KD process.
A-03	The architecture should assist in the generation and management of metadata attached to the contents of the data repository.
A-04	The architecture should not be bound to any database technology or vendor.
A-05	The architecture should be able to process any type of data, including unstructured and multimodal data.
A-06	The architecture should support the construction of KD workflows in the form of pipe-and-filter graphs.
A-07	The architecture should support partial execution of workflows based on previous intermediate results.
A-08	The architecture should provide a library of reusable application elements implementing solutions to common problems.
A-09	The architecture should provide internal interfaces for the purpose of extending the architecture with new application elements.
A-10	The architecture should provide external interfaces for communication with other software, to support requirement W-06.
A-11	The architecture should provide external interfaces for communication with humans, to support requirement W-05.
A-12	The architecture should support development with and of reusable components.
A-13	The architecture should grant developers access to it on multiple levels of abstraction, enabling them to control low-level functions.
A-14	The architecture should be portable to multiple hardware and operating system platforms.
A-15	The architecture should be able to deliver satisfactory performance under realistic operating conditions.
A-16	The architecture should cover all the design and coding stages of the development lifecycle of KD software.

3.3.1 Data architecture

A-01: The architecture should provide access to data sources via a transparent abstract interface.
A-02: The architecture should assist in the generation of a data repository structure that covers the entire KD process.

A-03: The architecture should assist in the generation and management of metadata attached to the contents of the data repository.

These requirements address the integration of the data architecture with the workflow architecture. To achieve a good architectural integration, the data architecture should present an interface that abstracts away irrelevant details and automates or assists in the generation of the underlying data structures. Convenient shortcuts to the most commonly needed data management functions should be provided.

The architecture of the database used by a KD application mirrors the architecture of the application itself to a considerable extent. Given a KD algorithm, a data structure suitable for storing the data it generates can be automatically derived from a machine-readable description of its output variables. Similarly, a complete database that implements the data management scheme specified by D-03 can be generated when a machine-readable description of the entire application is available.

The use of metadata is required for knowledge representation, and also for certain advanced data management functions. Data structures for attaching metadata to data should be provided by the architecture. Furthermore, the generation of the metadata itself should be automated and hidden by the data architecture. Where automation is not possible, a generic interface for metadata definition should be provided.

A-04: The architecture should not be bound to any database technology or vendor.

This requirement addresses the technology-independence of the data model, which should extend to the manifestation of the data model in the architectural model. In a software framework that implements the architectural model, vendor-specific database connectors are a desirable feature, since the required development effort is reduced when a connector for the chosen database platform is available as part of the framework. However, users of the framework should not be restricted to using only such connectors as are provided by the developers of the framework.

A-05: The architecture should be able to process any type of data, including unstructured and multimodal data.

This requirement addresses the diversity of the application domains where KD may be applied. As illustrated by the survey in Subsection 2.1.2, the data processed by a KD application does not necessarily come in the form of sets of individual data points. Instead, the data may represent a more complex modality such as spatiotemporal data,

unstructured text, or multimedia. The architecture should account for this by providing an open-ended type system that does not unnecessarily limit the range of data types available to KD solution developers.

3.3.2 Workflow architecture

A-06: The architecture should support the construction of KD workflows in the form of pipe-and-filter graphs.

This requirement addresses the principal architectural style of KD software developed using the architectural model. As discussed in Subsection 2.2.3, the pipes-and-filters style is well suited to the task of building and executing KD workflows, and therefore, it is desirable for the architectural model to support the development of applications representing this style. The basic features that the model needs to have in order to accomplish this include representing KD algorithms as filters, coupling filters with pipes to form a graph, reading input data from data sources, and pushing the data through the graph to be processed by the filters.

A-07: The architecture should support partial execution of workflows based on previous intermediate results.

This requirement addresses the ability to restarting a workflow from an intermediate step instead of re-executing the workflow in its entirety. The usefulness of this ability was cited in Subsection 3.1.1 as motivation for the requirement that the data management scheme should cover every state of data. This requirement thus complements D-03, ensuring that intermediate data stored in a database that implements the data model can indeed be used to initiate a workflow in a software framework that implements the architectural model.

A-08: The architecture should provide a library of reusable application elements implementing solutions to common problems.
A-09: The architecture should provide internal interfaces for the purpose of extending the architecture with new application elements.

These requirements address the acquisition of filters, pipes and other basic architectural elements from which KD applications are built. Certain sub-problems in KD software development are universal or near-universal, and it would be wasteful to have every

developer create their own solutions to these problems. Therefore, a software framework that implements the architectural model should come with a library of reusable standard solutions, with top priority placed upon the most broadly applicable ones, such as database connectors interfacing with the products of major vendors and filters implementing frequently used DM algorithms.

Anticipating every contingency with the library included in the framework is not a feasible goal, so it is necessary to complement the library with a set of internal interfaces for the development of new application elements. Developers using the framework to create KD applications can then use these interfaces to code new filters and other application elements whenever a solution to a particular sub-problem is not already available. The interfaces also enable the development and distribution of third-party libraries, increasing the probability that when a problem is encountered, a solution can be acquired without writing a large quantity of new code for it.

> *A-10: The architecture should provide external interfaces for communication with other software, to support requirement W-06.*

This requirement addresses interactions between technological artifacts that involve multiple software applications coordinating their actions. To support such coordination, a framework that implements the architectural model needs to provide external interfaces that enable it to send data to other applications and receive data from them. One situation where this is required is when a developer wishes to use a standalone external tool, such as a scientific computing application, to perform some tasks in the KD workflow. Another is when a KD application developed using the framework is to be integrated with a legacy information system.

> *A-11: The architecture should provide external interfaces for communication with humans, to support requirement W-05.*

This requirement addresses interactions between technological artifacts and human actors. To support such interactions, user interface (UI) hooks should be provided so that developers using the framework can build control and monitoring interfaces for their applications. Furthermore, the framework should provide UIs of its own, for the purpose of configuring and operating the framework itself. A graphical UI that can be used to create applications from library elements without writing code would be highly desirable. Another possible use case for UIs is a reversal-of-roles situation where an application assigns a task, such as a visual pattern recognition problem, to a human actor.

3.3.3 Architectural quality

A-12: The architecture should support development with and of reusable components.

This requirement addresses the efficiency of developing KD applications based on the architectural model. High efficiency can be achieved through component-based development, in which applications are assembled from reusable components, with standardized interfaces that enable the components to be connected, configured and coordinated. This development paradigm is highly compatible with the pipes-and-filters architectural style, and it should be supported by the architecture. Furthermore, the architecture should encourage developers to design with component-based development in mind, maximizing the reusability of the new application elements they write.

A-13: The architecture should grant developers access to it on multiple levels of abstraction, enabling them to control low-level functions.

This requirement addresses the flexibility of development based on the architectural model. A software framework that implements the model necessarily has several layers of functionality, with each layer providing services for the ones above it. Using higher-level services reduces the effort required to develop an application, but this advantage comes at the cost of losing control over lower-level behavior. Developers should therefore be given access to all the layers, so that they may choose the trade-off that best suits their purposes and build their applications on top of the layer that best represents the chosen trade-off.

A-14: The architecture should be portable to multiple hardware and operating system (OS) platforms.

This requirement addresses technology-independence from the perspective of execution platforms. There are several viable hardware architectures and operating systems on which KD applications may be deployed, and it would be undesirable for the framework to be limited to just one or a small subset of all the available options. Ideally, the framework should be portable to all major hardware and OS platforms in server, desktop and mobile environments. The means that may be used to achieve high portability include coding practices and virtual machines.

A-15: The architecture should be able to deliver satisfactory performance under realistic operating conditions.

This requirement addresses the increasing performance requirements of KD software. Several factors contribute to this trend: one is increasing dataset sizes (big data), another is the use of resource-restricted devices such as smartphones and tablets as KD application platforms. Any software framework necessarily generates some amount of processing overhead, but it should not get out of control with increasing data quantities or diminishing computing resources, or the applicability of the framework will be severely limited. Furthermore, to avoid being limited to the processing power of a single device, parallel computing platforms such as clusters and grids should be a significant consideration in the design of the framework.

> *A-16: The architecture should cover all the design and coding stages of the development lifecycle of KD software.*

This requirement addresses the range of KD software development tasks for which the framework is suitable. The principal ones are implementation and deployment of a KD solution, but before them, there is frequently an experimental phase involving the construction of application prototypes and testing of different algorithms and parameters. The framework should support these tasks as well, and enable developers to make a gradual transition to the final release version by replacing prototype components with finalized ones, one by one, while the overall application architecture remains unchanged.

4 Proposed solution

This chapter introduces the proposed solution to the research problem outlined in Chapter 1 and defined in detail in Chapter 3. The solution is composed of the three models identified in Chapter 1, the data model, the workflow model and the architectural model. Collectively, these three models constitute the augmented KD process model that the thesis seeks to formulate and validate.

Although the three models are all part of the same overall model, they represent different stages of refinement in terms of how concrete the model artifacts are. The most abstract of the three is the workflow model, which is mainly a conceptual framework. The most concrete is the architectural model, which is implemented both as generic software frameworks and as specific applications based on the frameworks. The data model is a set of design guidelines that are expressed in several software artifacts but have not been implemented in an application-independent manner.

The organization of this chapter is similar to that of the previous one, so that the data model is presented in Section 4.1, the workflow model in Section 4.2, and the architectural model in Section 4.3. The applications will be discussed later, in Chapter 6, as case studies for testing the solution against the requirements.

4.1 Data model

The three sub-models of the proposed KD process model can be organized into a rough dependence hierarchy with the data model at the bottom, the workflow model on top of the data model, and the architectural model on top of both. In other words, the workflow model depends on the data model, and the architectural model on both the data model and the workflow model. It therefore makes sense to begin the presentation of the process model by discussing the data model.

Subsection 4.1.1 examines data from three perspectives. First, it looks at what happens to data during KD, depicting the process as a sequence of states and transformations. Second, it studies the creation and management of a data repository for KD. Third, it analyzes the concept of knowledge and posits a number of principles intended to bridge the gap between knowledge as the outcome of the KD process and knowledge as an epistemological concept.

Subsection 4.1.2 follows up on all three perspectives, attempting to derive from the data model a knowledge representation scheme that can be used to build a knowledge base for KD. It begins by revisiting the previously established definition of knowledge and translating it into a set of KR formalisms. It then proceeds to discuss how these formalisms might be incorporated into the previously presented data repository design, enabling it to be used as a repository for discovered knowledge.

4.1.1 Data management

One way to view the established process model of knowledge discovery is that it depicts the categories, or states, of data involved in the process, and the transformations that cause data to move from one state to another. We begin the presentation of the data model by reviewing these states and transformations. Six states of data are identified here: raw data, prepared data, condensed data, activated data, explained data, and knowledge (see Figure 2).

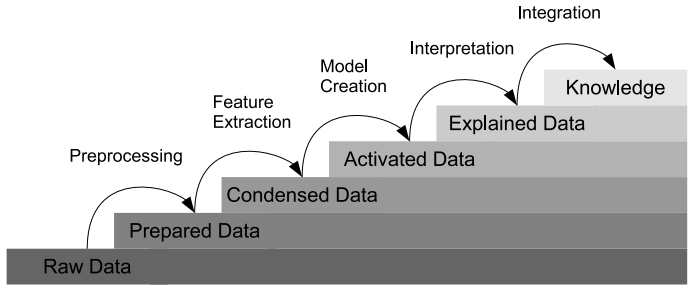


Fig 2. The knowledge discovery process, depicted in terms of data states and transformations.

Raw data is the starting point of the process, consisting of observations that have been recorded and digitized but not modified in any manner. The observations may come from a variety of sources, such as human observers or various sensor devices. If the phenomenon of interest occurs in the virtual world rather than the physical one, instruments such as logging software or online questionnaires can be used to gather data that is digital by nature and therefore ready for processing without digitization.

The first step of the KD process, preprocessing, transforms raw data into *prepared* data. This transformation does not yet move the data to a higher level of abstraction: the data still consists of observations. However, it may be necessary, e.g., to drop out some

of the data because it is believed to be erroneous or otherwise unsuitable. Compensating for missing values in data items is another frequently needed preprocessing operation.

The next process step takes the data to a higher level of abstraction and radically reduces its volume. The result, *condensed* data, consists of features, the purpose of which is to represent essential characteristics of the data using relatively few values. The goal of feature extraction and selection is to come up with a set of features small enough to make further processing feasible, yet informative enough to constitute an adequate stand-in for the full dataset.

The concrete meaning of the feature selection criteria depends on what kind of method is used to process the feature values in the next process step. In this step, the features are used to train a model that learns the relevant properties of the dataset and generates a function that, when a data item is plugged in, answers a question concerning the data item. Because the data, which thus far has been input for processing methods, has now been transformed into a processing method, this new type of data is given here the name *activated* data.

Assuming that the process has been successful so far, the activated data represents a previously unknown predictive or descriptive rule concerning the behavior of a real-world phenomenon. The next step is to consider the meaning of the rule with respect to the phenomenon in question. The objective of this is to understand why the rule works and under what circumstances it does not work. The result amounts to a new category of data, which is here called *explained* data.

The final step is to incorporate the explained data with the body of existing *knowledge*. A body of verified knowledge should not contain any incoherencies, so any incompatibilities between the existing knowledge and the new addition must be resolved, a task that may involve revisiting the KD process. Once a coherent view has been established, the transformation of the raw data into knowledge is complete.

The purpose of recapitulating and reformulating the established KD process model in this way is to shift the focus of the model slightly: instead of data transformations, we are now concentrating on the data itself. This enables us to use the established model as a foundational process dimension to which additional dimensions can be anchored. The data, workflow and architectural models are all connected through this conceptual model of data.

Figure 3 depicts a reference architecture for a data repository based on the conceptualization of the KD process given above. The reference architecture is composed of three distinct architectural layers, the bottommost of which is the entity labelled

"Data Store" in the figure. This is simply a database designed such that it can be used to store data in any of the states described above. Any type of database can be used to implement the data store, since only its content is specified by the reference architecture, not its storage format.

Above the data store is another database, labelled "Metadata Store" in the figure. The purpose of the metadata store is to store information that describes various aspects of the actual repository content stored in the data store. General dataset properties such as producer information and timestamps are useful for searching the repository for a specific dataset. Tracking information establishes lateral relationships between datasets representing different states of data, allowing a result to be traced back along the transformation chain. Semantic annotations are used to represent the meaning of data; their importance increases toward the end of the transformation chain, as the concepts embodied by the data grow increasingly complex.

The topmost layer of the reference architecture consists of two access interfaces, labelled "Import Interface" and "Export Interface" in the figure. The interfaces expose a set of operations on the data repository to users and hide the implementation of the data and metadata stores from the users. The import interface is used to write data into the repository; it stores the primary content in the data store and the descriptive metadata in the metadata store. The export interface accesses the metadata store to locate the requested dataset and retrieves it from the data store, returning it to the user.

Although knowledge is treated above as one state of data among others, it holds a special status owing to its relationship with the philosophical concept of knowledge. Recalling the classical definition of knowledge as justified true belief, we can examine the concepts of justification, truth and belief more closely in order to get a better idea of the properties that define knowledge and set it apart from less refined forms of data. From such properties, we can derive conditions that must be satisfied for the outcome of the KD process to qualify as knowledge in a broader sense than the one normally used within the KD domain.

Starting with the concept of justification, we can define it as a convincing rational argument supporting a proposition. Since KD starts with data, that is where the justification for discovered knowledge must ultimately be found. We can thus derive two principles from the definition of justification, corresponding to the two keywords in the definition, *rational* and *convincing*.

For the justification for a piece of knowledge to be rational, the knowledge should be *traceable to the original data via a logical sequence of transformations*. Discontinuities

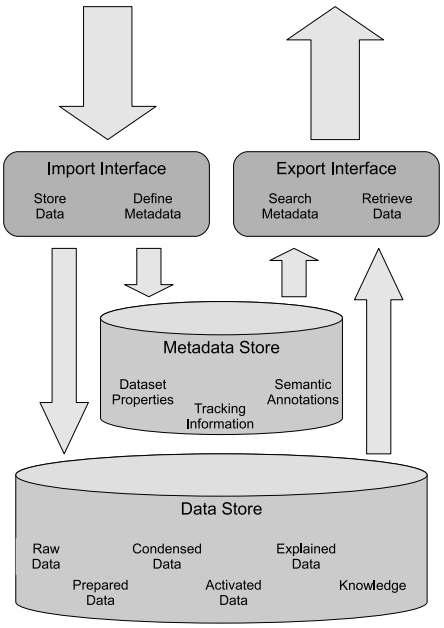


Fig 3. A generic design for a data repository intended to support the KD process.

in the traceability chain make it more difficult to understand the outcome of the discovery process: to grasp its significance with respect to the real world, to assess the scope of its applicability, to detect its weak points. Applying discovery results without properly understanding them invites problems. The discovery process should therefore not just produce an outcome that works, but also generate an explanation for why it works and suggest constraints on when it can be expected to work.

For the justification for a piece of knowledge to be convincing, the knowledge should be *backed up by enough data to dispel any reasonable doubt regarding its reliability*. This principle is already codified in several basic KD concepts, such as those of support and confidence in association rule mining, where a rule is only accepted as significant if its support (the fraction of the overall dataset that is relevant to the rule) and confidence (the fraction of the support set that follows the rule) exceed a predetermined threshold. In predictive modeling, the sufficiency of supporting data can be controlled by using validation datasets and by studying the data space to find regions where the available data is too sparse.

Moving on to the concept of truth, there are three major attempts to define it in classical epistemology: *correspondence* theory, *coherence* theory, and *pragmatic* theory.

Each theory has its own virtues and its own drawbacks, so that no single theory offers a perfect solution, but neither should any of them be rejected completely. The most sensible approach is to combine useful elements from all three theories into a hybrid view of the concept of truth, using each theory in turn to reveal a different aspect of the concept as a potential source of additional principles to be used as criteria for knowledge.

Correspondence theory states, simply put, that a proposition is true if and only if it corresponds to reality. To expand this definition a little, we can say that the truth of a proposition is determined by an examination of the real-world entities it invokes: if such an examination shows the statements made in the proposition to be accurate, then and only then is the proposition considered true. While this definition may seem attractive, as a test of truth it is rather problematic. There is no way to compare what a proposition says about reality to reality in itself, so one is left wondering what the practical method of determining truth should be.

A lesson that we can derive from correspondence theory is that the results of knowledge discovery represent objective statements concerning reality and should be treated as such. A KD expert should therefore *understand the results of a KD project not only in terms of their utility, but also in terms of what facts they reveal about the world*. As a test condition, this principle relies on the expert judgment of humans, emphasizing the role of human actors as the driving force of the KD process.

Coherence theory defines the truth of a proposition in terms of the system of propositions previously established as true. A true proposition is one that can be incorporated into the system without disturbing it: it corroborates and is corroborated by some of the established truths, and contradicts none. A proposition that fails to satisfy this condition is suspended as inadequately supported or rejected as demonstrably false. For instance, it is not uncommon for KD algorithms to learn properties of their training datasets that are merely random concentrations in the distribution of the data, a phenomenon known as overfitting. Independent validation of the discovered model is necessary in order to ensure that it is coherent with more than just the sample of observable reality represented by the training data.

However, there is always the possibility that a new, conflicting result represents a previously undiscovered truth, in which case there must be some kind of error in the body of existing knowledge. The weakness of coherence theory is that it is biased against accepting the new result also in this case. Although a dramatic upheaval of the status quo is unlikely to be caused by the outcome of a single KD effort, it is nevertheless

enlightening to look for the source of the conflict, since it may point out new lines of investigation, which may, in turn, eventually lead to major new discoveries. KD experts should therefore *make an effort to build a coherent model of the world that combines their new results with previous knowledge*, but if this fails, they should *make an effort to explain the apparent anomaly*.

Pragmatic theory differs from both correspondence theory and coherence theory in that it provides a practical way to test individual propositions in isolation, using only observable events as the criterion of truth. The theory states that a proposition is true if and only if assuming it to be true leads to a favorable outcome. To expand this definition, a pragmatic test of truth could be phrased as follows: "If I live my life and choose my actions assuming X to be true, are the results better than when I assume X to be false instead?" This, again, is problematic in a number of ways. A major practical problem is that making sufficient observations to come to an adequately conclusive decision may not be feasible because of the time or effort required. Another potential problem is the element of epistemological subjectivism introduced by the concept of favorable outcome.

Applying the pragmatic theory of truth in the context of KD brings forth the notion that the truth of a piece of knowledge is equivalent to its utility in a given application domain. Whether or not a KD result is accepted as true thus depends on how and within what constraints the experts expect to use it, so we must again rely on their judgment to establish a condition: KD experts should, when defining a KD problem, *define an objective test for deciding when to accept a result as (provisionally) true*. This may turn out to be not at all straightforward, owing to the exploratory nature of KD — frequently the objective is not to answer specific questions but to reveal any potentially interesting patterns in the available data without more than a vague prior expectation of what kind of knowledge the patterns might yield. Even so, however, at the end of the process, the results must be interpreted to find out what they imply, and evaluated to see if there is sufficient proof to accept the implications.

The third and final concept invoked by the classical definition of knowledge, belief, is usually defined in the KD context as a continuous value, the estimated probability of a proposition being true. In Bayesian probability theory, this value is calculated as prior belief modified by new (posterior) evidence, so by this definition, the concept of belief is closely related to coherence and justification: propositions that appear less likely in light of prior knowledge need to be backed up by a larger quantity of supporting data before they can be accepted as true.

In the more general sense, belief is a psychological phenomenon that occurs in the mind of the KD expert. The link to justification and coherence remains strong, but achieving belief is now a matter of how the justification is presented to and processed by the expert. First of all, the justification for each piece of knowledge should be *communicated in a form that allows a KD expert to grasp it*. Furthermore, the KD expert should *strive to understand the justification for each piece of knowledge, suspending belief when this fails*. Since it is in the interest of the expert to consolidate every piece of good knowledge discovered, it makes sense to make an effort to understand the justification, even if the way it is presented makes it difficult to grasp. However, in cases where the effort is inconclusive, it is better to err on the side of caution than to accept false information as true. Often, such cases can be resolved through further inquiry, e.g., by gathering more data or by examining the results in a new way.

4.1.2 Knowledge representation

Subsection 2.3.3 reviewed existing work on knowledge representation, with a particular focus on the use of machine-readable ontology descriptions and logic-based reasoning for KR. Although an alternative scheme is proposed in this subsection, the intent is not to imply that other KR approaches are flawed. The discussion that follows is exploratory rather than explanatory in nature, an attempt to extend the data model presented above such that it solves the KR problem in the context of KD specifically.

Keeping in mind the analysis of the concept of knowledge in the previous subsection, we can borrow from pragmatic epistemology the idea that knowledge is something that can be successfully applied to solve problems. We can then say that a computer knows X if and only if it is able to apply X independently in the appropriate context. This gives us an intuitive definition of KR technology as any technology that enables a computer-based system to possess knowledge in the pragmatic sense.

If we now examine again the view of data states and transformations given at the beginning of the previous subsection, we find that it is compatible with this definition of KR. In the knowledge-as-data reduction of the data model, knowledge is a predictive or descriptive function discovered from data, which has been interpreted as a model of reality and integrated with existing knowledge. The knowledge is thus by definition in a format that allows it to be executed by a computer to solve a problem — what is missing is a mechanism that enables the computer to identify the correct solution to a given problem, retrieve it from a knowledge base and use it to solve the problem.

We can now formalize the above by defining three conditions that must be satisfied. First, the system must possess the actual data for each individual piece of knowledge X. Second, it must have functions that enable it to apply X in practice. Third, it must be able to determine when it is appropriate to apply X and to locate X in such situations. Thus we can define a digital representation of a piece of knowledge X as the combination of three components:

– *Passive* component: the data that solves the problem to which X is related.
– *Active* component: functions that allow the data to be retrieved and applied.
– *Descriptive* component: metadata that allows the functions to decide whether X is the correct solution to the current problem.

The passive component of knowledge corresponds to the activated state of data in the data model. While this may seem contradictory, the term is justified in this context because the component is passive from the perspective of the KR system: a static object stored in the KB, the content of which is of no concern to the system. It is only after the data has been retrieved from the KB and returned to the entity that requested it that the data becomes active again.

The active component of knowledge is what allows the passive component to be produced from the KB for the requester. This is somewhat counterintuitive, since normally search and retrieval operations would be considered part of the access interface of the KB rather than part of its content. However, storage and retrieval of knowledge is a considerably more complex task than that of less refined forms of data, and the approach proposed here for dealing with that complexity is to introduce an additional level of processing between KB interface and KB content. The case study in Subsection 6.1.2 illustrates how this additional level might be implemented in a KR system.

With the descriptive component of knowledge we are already familiar from the previous subsection, where the use of metadata to enrich database content was discussed. To facilitate knowledge representation, the metadata needs to include information readable to the active component that describes the problem that can be solved using a given piece of knowledge and the circumstances under which the knowledge can be applied. This information is available as a result of the transformation from activated to explained data, so the problem is defining a suitable machine-readable encoding for incorporating the information in the KB. This is again illustrated by the case study in Subsection 6.1.2.

In terms of the reference architecture depicted in Figure 3, the passive component of knowledge would be stored in the data store and the descriptive component in the metadata store. Only the active component has no designated place in the architecture. We can thus extend the reference architecture to accommodate the KR scheme described above through the addition of one architectural layer. The extended architecture is shown in Figure 4.

The new layer is labelled "Function Store" in the figure and placed between the access interfaces and the data and metadata stores. As the arrows in the figure indicate, the access interfaces now communicate exclusively with the function store, which in turn communicates with the data and metadata stores. The function store thus assumes the role of an additional interface layer, which exposes its services to the import and export interfaces. The addition of a new abstraction layer makes the complexity of manipulating and searching a knowledge base, as opposed to a regular database, more manageable.

4.2 Workflow model

Although the presentation of the data model began with a recapitulation of the established KD process model, the closest counterpart of the established model in the augmented model is the workflow model. The most significant modification proposed by the augmented model is the greater emphasis upon process actors, especially human actors, and the nonlinear nature of the interactions between the actors that move the process forward. Furthermore, the augmented model brings into consideration actors not considered by the established model, and takes into account actors' expectations that are not aligned with the objectives of the KD process.

Subsection 4.2.1 discusses the actors of the KD process. It begins by constructing a hierarchy of the technological artifacts used in KD, which are considered actors in this context. It then proceeds to discuss the human actors, or stakeholders. The role of each class of human actors is described in terms of what the actor expects from other actors and what is expected from it by other actors.

Interactions between the actors are the topic of Subsection 4.2.2. Interactions between technological actors, those between technology and humans, and those involving only humans are examined each in turn. Special attention is devoted to interactions between actors whose expectations may be in conflict, since such conflicts are a possible source of ethical issues and should therefore be resolved in a manner that respects the rights of

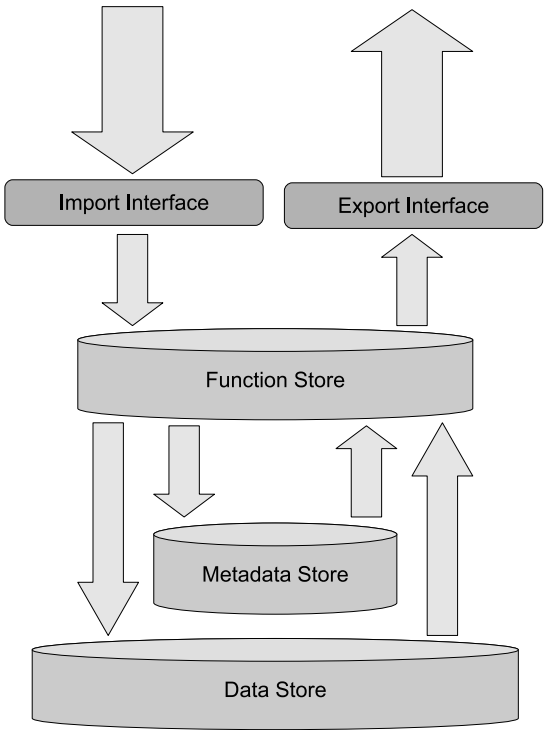


Fig 4. A generic design for a data repository intended to support the KD process, with extensions to facilitate knowledge representation.

the actors involved. The subsection is concluded by an attempt to reconcile the workflow model with the CRISP-DM model, the current standard model of the KD process.

4.2.1 Actors

Starting once again with the established model of the KD process, we can think about data as a horizontal chain-like structure, with the various categories or states of data placed next to each other and linked to their neighbors, as in Figure 2. In this view, the technology used in the process forms a vertical layered structure above the data. The structure consists of four layers, from bottom to top: data management, algorithms, applications, and systems (see Figure 5).

The *data management* layer consists of software for storing and accessing data and knowledge. File systems, databases and knowledge bases are the most important categories of data management technology. These three form a layered hierarchy of

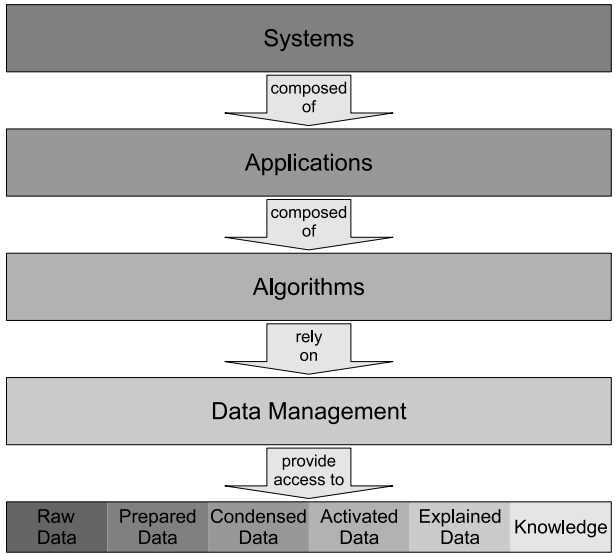


Fig 5. The hierarchy of knowledge discovery technology, including the hierarchy of data from Figure 2 as the bottommost layer.

their own, since a database system usually stores its data in files and a knowledge base system, in turn, generally relies on some kind of database engine. The higher one moves in this hierarchy, the more processing the data management software does on behalf of the client using its services.

The *algorithms* layer consists of small pieces of software that work on the data. Each algorithm represents a solution to a particular sub-problem encountered during execution of the KD process. For any given problem, there are usually several potentially acceptable solutions; the nature of the problem determines which one is the most appropriate. The solutions are implemented in program code and typically gathered in software libraries, where they can be conveniently invoked when required.

The *applications* layer consists of standalone software entities composed of algorithms and coordinating code. An application solves KD problems that are broader in scope than those solved by individual algorithms. This is done by breaking down the problem into sub-problems and applying algorithms in a coordinated fashion to solve them. Often, a software framework is employed, which makes it convenient to assemble a complete application from components representing algorithms.

The topmost layer consists of *systems* composed of multiple collaborating applications. Various forms of collaboration are possible: applications may communicate

with one another directly, or the communication may be mediated by human operators who transfer information between applications. In the latter case, the operators can be considered components of the system along with the applications they control.

The term "actor", as has already been remarked, is used here in a broad sense that covers entities belonging to the technology hierarchy, especially applications and systems, as well as humans. To distinguish human actors from technological ones, we now introduce the concept of a *stakeholder*. Being a stakeholder in the KD process means one or more of the following:

– having a *vested interest* in the results of the process;
– *participating* in the planning or execution of the process;
– being *affected by* the execution or results of the process.

Four groups of stakeholders are identified: technology experts, domain experts, volunteers and data subjects. Society is considered the fifth stakeholder, concerned with KD as a phenomenon rather than individual KD projects. Knowledge users are not counted as a distinct stakeholder group since they are represented in the process by domain experts and, to some extent, by society. Figure 6, adapted from (Tuovinen & Röning 2009), illustrates the relationships between the stakeholder groups.

Technology experts are people with the knowledge and skills required to apply KD technology. They understand the process and methods of KD, are familiar with KD tools and are able to use their knowledge to design and implement solutions to KD problems. Defining the problems is the job of the other expert group, domain experts.

Domain experts are people who are intimately familiar with a specific application domain of KD technology. They have a comprehensive view of what in the domain is already known and what is not, what knowledge would be desirable and what data can be acquired from which new knowledge could be discovered. Simply put, technology experts need domain experts to tell them what they should look for, whereas domain experts need technology experts to tell them how they can find it.

The two expert stakeholder groups are the primary drivers of the KD process. The objectives of the process are directly dependent on the business or research needs of the domain experts, and the means by which the process progresses toward the objectives are directly dependent on the analysis of the technology experts. Both groups are characterized by the fact that their individual expectations are, generally speaking, aligned with the goals of the process as a whole.

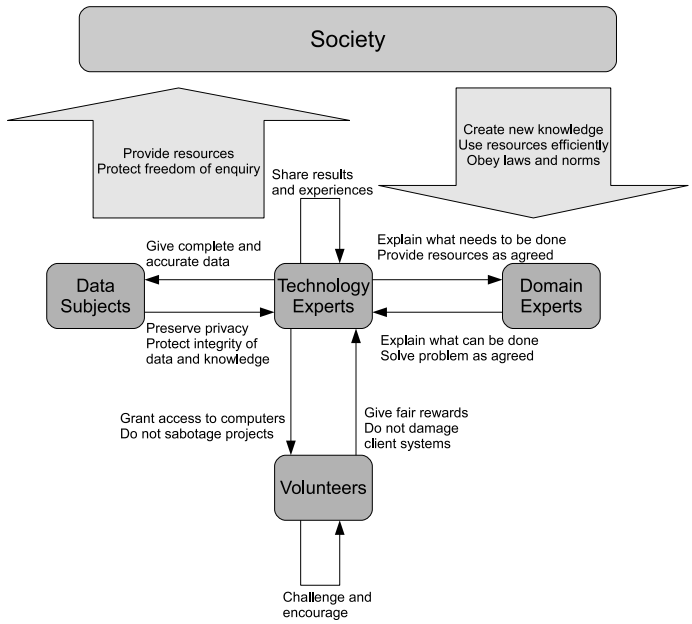


Fig 6. Stakeholders and their expectations. An arrow pointing from stakeholder A to stakeholder B indicates that A expects something from B; the arrow caption states the expectation verbally. An arrow may also point from a stakeholder to itself, indicating an expectation of community support. For the knowledge discovery process to proceed smoothly, all stakeholder expectations must be satisfied.

It is easy to see that since KD necessarily involves defining and solving a problem, the two expert groups must always be present, although they need not be distinct — it is possible for a person to play the role of both technology expert and domain expert. Conversely, the two non-expert stakeholder groups are not an essential component of the KD process and therefore are involved only in special cases. However, it is in these cases that some of the most interesting ethical issues of KD arise.

Volunteers are people who assist the experts in the execution of the KD process by performing tasks that do not require significant technology or domain expertise. Volunteer computing, or public-resource computing, is a distributed computing model that makes use of computing resources donated voluntarily by users of personal computers with Internet access. A volunteer computing server coordinates the computation, distributing tasks to volunteer machines. Each volunteer machine hosts a volunteer computing client that downloads a task from the server, processes it, uploads the result and requests another task, restarting the cycle. The administrator of a volunteer machine

can specify what resources the client application is allowed to consume and at what times. Figure 7 illustrates how the volunteer computing model works. (Anderson *et al.* 2002)

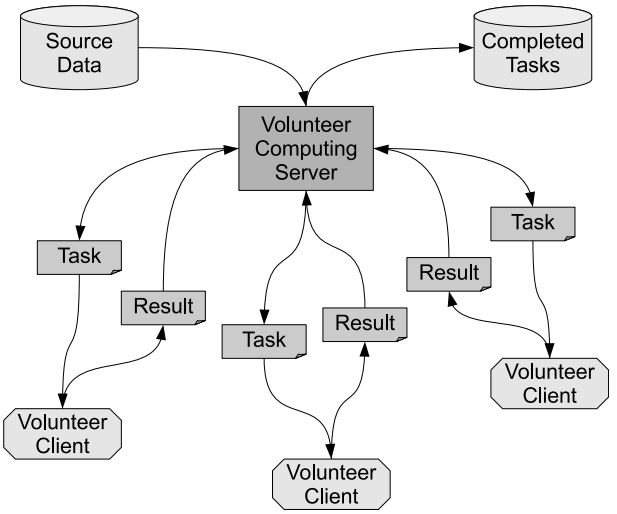


Fig 7. Volunteer computing illustrated. The server coordinates the work of the clients, distributing tasks and collecting results. The client machines work on their tasks as permitted by their administrators, and when finished, contact the server over the Internet to return the result and request a new task.

In the basic volunteer computing model, the task of the volunteers is to install and configure the volunteer computing client, but some projects engage the volunteers themselves rather than their idle computing resources by giving them reasoning or recognition tasks for which they can quickly train themselves using materials prepared by the experts (Raddick *et al.* 2010, Cooper *et al.* 2010). Volunteer computing can be a powerful way to attack a KD problem, provided that it can be broken down into relatively small sub-problems that can be solved independently. If, for instance, a research project requires heavy computation but lacks the funds to purchase the necessary hardware, recruiting volunteers to execute the computations on their desktop machines may be the solution. Probably the most famous volunteer computing project is U.C. Berkeley's SETI@home (Anderson *et al.* 2002), but many others also exist, studying such diverse topics as climate modeling (Christensen *et al.* 2005), protein folding (Beberg *et al.* 2009) and prime numbers (Ziegler 2004).

Data subjects are people who provide data regarding themselves as material for the knowledge discovery process. They do this by either recording the data by themselves (e.g., by filling in a questionnaire) or operating in an environment, physical or virtual, where their actions can be observed and recorded by an automated tracking or surveillance system. In the latter case, the collection of data does not depend on the subjects' input, knowledge or even consent, so being a data subject does not necessarily imply an active role in the KD process.

Of the two non-expert stakeholder groups, data subjects play the more critical role in the sense that whereas volunteers may sometimes provide the most cost-effective way to gain access to the computing resources required by a KD project, they are not strictly necessary for the project to succeed. If, however, the project is based on the premise that a body of data gathered from a specific group of subjects will be mined, then access to that data must be gained or the project cannot advance. Personal information is typically guarded by both authorities and the data subjects themselves, a fact which further emphasizes the importance of giving all due consideration to the expectations of this stakeholder group.

Society is a special case among the stakeholders in that it does not refer to a specific group of people, except in the sense that a society is composed of the individual human beings living in it. However, it makes sense to count society among the stakeholders because the boundary conditions faced by all KD projects are largely determined by its attitude towards knowledge and KD. Since KD technology is an instrument for generating new knowledge, arguments for the proliferation of knowledge in society are also arguments for the encouragement of KD by society. Factors such as the availability of research funding and affordable hardware can be boosted to increase the output of KD efforts. Funding decisions can also be used to steer KD projects towards types of knowledge considered particularly beneficial.

On the other hand, the interest of society in the generation of new knowledge is not unconditional; the process by which knowledge is created may also be socially relevant, in some cases more so than the outcome. For example, if the raw data for a KD project is gathered by potentially illegal means, society has an obvious interest in investigating the matter and, if necessary, imposing restrictions. Society is thus a stakeholder that plays multiple roles in the KD process: it creates a knowledge-friendly environment that KD practitioners can use to their benefit, but at the same time it sets boundary conditions that practitioners must observe in order to be successful. The social implications of KD should therefore be considered in the KD process model: since

society will always be present in the background, it is common sense to acknowledge its presence and make explicit its effects on the process.

4.2.2 Interactions

The interactions that take place during the KD process can be divided into three categories based on the involvement of humans. Interactions in the first category take place between different technological artifacts, so there is no human involvement in them. Interactions in the second category take place between humans and technology, and those in the third category are between humans with no technology involved.

It is notable, when one reviews KD literature with respect to its relevance to the three categories of interactions, that the quantity of published research grows lesser as the degree of human involvement in the research topics grows greater. There is a lot of research on exclusively technical KD topics, some research on certain special cases where humans interact with technology, and very little research on the effects of human-to-human interactions on the KD process.

One can speculate on whether the apparent imbalance in the research interests of KD researchers is a result of KD being relatively young as a research area. Although KD as an engineering discipline is already quite mature, it may be that this degree of maturity has not yet reached the less technically oriented corners of the KD domain. A background in engineering or mathematical sciences alone is not suitable for approaching problems related to human-machine interfaces or human-human relationships, and it seems plausible that there has not been enough time to achieve the level of interdisciplinary collaboration necessary to solve these problems.

Studying the backgrounds and interests of KD researchers beyond the level of speculation is out of scope for this thesis. The uneven distribution of published results across the three categories is an observable fact, and it is argued here that it is not simply reflective of the importance of the categories relative to one another. One of the objectives of the thesis is therefore to promote questions that are currently not receiving much attention from KD researchers.

In the most fundamental interaction of the KD process, an algorithm interacts with data to produce transformed data. Several algorithms interacting with one another make an application, and applications interacting with one another form a system of collaborating applications. This is the same hierarchy that was already described in the subsection on technology. It also contains the established data mining process,

which is essentially a series of interactions of algorithms with data and other algorithms. Together these interactions form our first category, consisting of interactions between technological artifacts.

The second category of interactions, those involving a human actor and a technological artifact, similarly covers every level of the technology hierarchy and every step of the DM process. At the beginning of the process, humans interact with data as they compile datasets for mining, and at the end of the process, they do it again as they explore and utilize the results of the process. Every intermediate step involves humans interacting with other levels in the technology hierarchy as they configure and operate algorithms, applications and systems to make them perform the required computations.

An interesting special case of humans interacting with data is when a human takes the place of an algorithm in the KD process, performing a pattern-recognition task for which humans have a natural aptitude, not easily matched by a computer program. Notable examples can be found among volunteer computing projects; the Galaxy Zoo project, for instance, asks volunteers to classify galaxies photographed by the Hubble Space Telescope (Raddick *et al.* 2010). The Foldit project adopts a more subtle approach, enticing volunteers to play a computer game that allows them to contribute to research on protein folding (Cooper *et al.* 2010).

The first two categories of interactions are similar in that they both relate to the configuration and execution of the components that make up the KD process. The third category of interactions, those of human actors with other human actors, is different: rather than steering the process, these interactions are concerned with creating an environment in which the process can take place. The most fundamental of these interactions is that of domain and technology experts working together to define the KD problem they wish to solve. Each of the other stakeholders must be negotiated with as well, to find out what their expectations are and how they can be balanced with the (potentially conflicting) expectations of other stakeholders. Finally, before the KD process can be launched, roles must be assigned to those who are to be responsible for carrying out the process. The distribution of roles and responsibilities is yet another type of human-human interaction.

Figure 8 illustrates the three categories of interactions described above. As already noted, the third category is the least thoroughly studied one, yet in a sense, it is the most important of the three: it is the human-human interactions that create the conditions necessary for a KD project to begin. There are several angles from which this area of

study could be approached, some of which are discussed in Section 7.3. Beyond that, however, the topic cannot be advanced in this work.

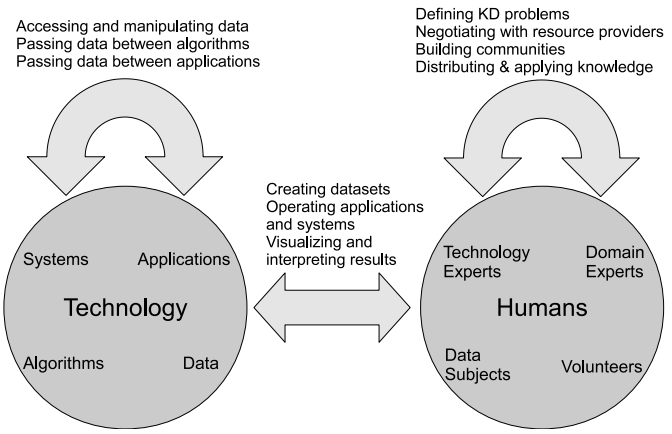


Fig 8. Actors and interactions. The two main categories of actors, technology and humans, correspond to Figure 5 and Figure 6, respectively. The arrows represent the three main categories of interactions: interactions within the technology category, those within the humans category, and those that bridge the two categories.

There are also differences between interactions in the human-human category. Interactions among experts are the more straightforward ones, since the primary concern of all experts is the successful completion of the KD effort. This is not to say that there can be no conflicts between experts, but if such conflicts do occur, they are more likely to be about the terms of their collaboration than the objectives. Resolving the conflict in this case is a simpler task than if the two parties were in disagreement over what is to be achieved.

Interactions involving non-expert stakeholders — data subjects or volunteers — are more difficult in the sense that the objectives of non-experts may not be aligned with those of experts, and may even be in direct conflict with them. Nevertheless, the non-experts are entitled to have their legitimate expectations fulfilled, even if it requires special provisions or trade-offs to be made by the experts. To illustrate this, let us first consider how the experts should handle interactions with volunteers.

The ethical aspect of volunteer computing is examined in (Tuovinen & Röning 2009). The paper analyzes the relationship of the coordinators of a volunteer computing project and the volunteers who participate in the project, identifying three dimensions of

this relationship that need to be considered from the perspective of ethics. These three are trust, fairness and persuasion.

It is easy to see why trust is an important issue in the volunteer computing model: The coordinators ask the volunteers to install a piece of software and let it operate independently on their computers. Given how much malicious software there is on the Internet, common sense dictates that one should only download software from sources one trusts, and it is up to the coordinators to convince potential volunteers of their trustworthiness. On the other hand, the coordinators also need to be able to trust the volunteers, because the success of the project depends on the volunteers' goodwill. A malicious volunteer might, for instance, try to sabotage the computation by tampering with the client application such that it returns false results to the server. This possibility has been recognized by developers of volunteer computing platforms, and technical means exist to counter it (Sarmenta 2002, Domingues *et al.* 2007).

The issue of fairness in volunteer computing is essentially a question of what is appropriate compensation for the volunteers for their participation. While the volunteer computing model depends on the volunteers not expecting to be paid for their contributions, the principle of fairness prescribes that they should be acknowledged in some manner. For example, if a volunteer computing project produces publishable scientific results, the authors of the publications should not omit to mention the contribution of the volunteers. The Foldit project (Cooper *et al.* 2010) sets an interesting precedent with respect to fairness: several papers published by the Foldit team give collective co-author credit to the players whose gameplay produced the results reported in the papers.

Persuasion in itself is not an ethical issue, but it becomes one when one considers how it is applied: people can be persuaded to do ethical or unethical things, and the motivation and means of persuasion can be ethical or unethical. In volunteer computing, the issue of persuasion is closely linked with the issue of trust, because creating trust is a major task in persuading potential volunteers to join a volunteer computing project. However, there are also persuasion issues not directly related to trust; for instance, the coordinators should avoid misleading potential volunteers when explaining the topic, scope and impact of their project. This ensures that volunteers have adequate information to choose the projects most compatible with their values.

Volunteer computing, of course, only represents a specific and relatively small category of KD, and the ethical issues involved in it are not relevant to KD projects that do not enlist volunteers. To find examples of ethical issues not restricted to volunteer

computing, let us examine again the concept of trust. It is one of the basic premises of KD that data is a valuable asset. From this, it follows that allowing someone else to handle your data is an act of trust, although this fact is easy to overlook if one only focuses on the methods and technology used to process the data. Trust is therefore frequently an issue in KD projects, even if it is not explicitly recognized.

The problematic relationship of KD, personal data and privacy was already discussed in Subsection 2.4.3, and again in Subsection 3.2.2. It was observed that personal data is potentially sensitive, and that applying KD methods to personal data may amplify this property, deriving sensitive conclusions even from an otherwise unsensitive dataset. Given the value and legal protection assigned to privacy, any data analysis operation that may invade someone's privacy is dependent on trust, whether or not the consent of the individuals concerned is required to gain access to the data.

In some cases, the data subjects are not simply providers of raw material but potential beneficiaries of the KD process. For example, if the provider of an online service applies KD to usage data for personalization purposes, then both the users of the service and the provider can benefit from the resulting improvement of user experience. In a more clear-cut example, medical KD aimed at improving the efficacy of a therapy can directly increase the quality of life of data subjects suffering from a chronic condition. In the latter case, there is a strong element of common good involved, making society a beneficiary as well.

Using KD for crime prevention provides another example of the role of society in interactions with data subjects. In this case society, in the form of authorities, assumes an active role by explicitly sanctioning the use of personal data. Here, again, there is the element of common good, but also the possibility of concrete harm to a data subject if a false positive causes one to be falsely implicated. In light of these examples, interactions involving data subjects are tri- rather than bilateral and emphasize the position of society as one of the actors in the workflow model.

Up to now, the workflow model has concentrated on identifying and describing the interactions that the actors of the KD process engage in. While these have a certain value in themselves, a list of activities does not yet constitute a process model; the activities must be placed within a structural framework that dictates the overall flow of the process. Although the thesis emphasizes the nonlinear aspect of the KD process, a high-level sequential workflow specification is still necessary for the augmented process model to be applicable in practice. We can adopt such a specification from the CRISP-DM process model.

The CRISP-DM model is composed of six major phases, shown in Table 4. Each row of the table gives the name of a CRISP-DM phase, a short description of the phase, and a list of corresponding interactions in the workflow model. The table thus places the activities specified by the workflow model into rough order of execution. It also serves as a checklist to find out if there are any major parts of the process that are conspicuously not addressed by the workflow model.

Table 4. The six major phases of the CRISP-DM process model and the corresponding interactions in the proposed workflow model.

Process Phase	Interactions
Business understanding: defining objectives; creating project plan	Human-human: defining KD problems Human-human: negotiating objectives and resources Human-human: recruiting volunteers
Data understanding: reviewing available datasets; evaluating dataset properties	Human-human: soliciting data Human-technology: creating initial datasets
Data preparation: selecting and preprocessing data for analysis	Human-technology: creating final raw datasets Human-technology: operating preprocessing software Technology-technology: transforming data
Modeling: analyzing the data with data mining algorithms	Human-technology: operating data mining software Technology-technology: transforming data Human-technology: analyzing/visualizing data
Evaluation: reviewing the modeling results against the objectives	Human-technology: interpreting results Human-human: discussing results
Deployment: preparing the results for application	Human-human: distributing knowledge Human-human: applying knowledge

It is interesting to note that there are interactions identified by the workflow model that do not readily fit into any individual phase of the CRISP-DM model. These are the ones that fall outside the immediate sphere of influence of the primary (expert) stakeholders: human-human interactions among non-experts, such as community-building activities of volunteers, and the influence of society. These are actually processes of their own that do not target any individual KD project, which makes it

difficult to address them from within the KD process in a meaningful way. In the long run, however, it pays off to cultivate a mutually benevolent relationship with non-expert stakeholders, for example, by fostering the creation of an established community of volunteers. Thus, one more recommendation can be derived from the workflow model: KD practitioners should look beyond the project at hand and plan for the long term to ensure a favorable landscape for future projects.

4.3 Architectural model

One of the major themes in KD research has been the development of software architectures for KD applications. This work is based on the principle that since KD applications share basic functionality, they also tend to share certain architectural foundations. Thus, it makes sense to try to identify the common architectural elements of KD applications and to assemble them into a reference architecture, making it faster and easier to build individual KD solutions for various purposes.

A good reference architecture is particularly effective when implemented as a software framework. A framework captures architectural objects and their relationships in a body of code upon which software developers can build their applications. Unlike a library, a framework takes charge of the flow of control and data in an application, relying on code created by the application developers to take care of application-specific functions only. The architectural model proposed in this thesis is presented in the form of two software frameworks developed or co-developed by the author: Smart Archive (SA) and Device-Based Software Architecture (DBSA).

Both SA and DBSA were developed to support the scientific work of the Intelligent Systems Group at the University of Oulu, Finland. Between the two frameworks, there is a lot of overlap in design and functionality, owing to their similar goals and also to the fact that several members of the research group were involved in the development of both. However, both frameworks were also used as test platforms for new ideas concerning the architecture of KD software, and as a result there are also several notable differences between the two, with each framework highlighting a different set of priorities and design innovations.

The SA framework (Tuovinen *et al.* 2008), as it appears in this thesis, was developed over a three-year period starting at the beginning of 2006. It was originally a Java-based framework for KD in desktop and server environments; later it was ported to C++ to explore its suitability for mobile platforms (Rautio *et al.* 2009). The overall goal of

this work was to investigate the requirements of KD frameworks, identify interesting problems and create novel solutions. The framework was also used to solve real-world industrial data mining problems.

The DBSA framework (Kätevä *et al.* 2010) is another data flow-oriented software framework, developed in part to be a successor to Smart Archive. Like SA, DBSA is aimed primarily at KD problems but suitable also for other types of data processing tasks. Being based to a significant degree on previous experiences with Smart Archive, DBSA adopts the innovative aspects of SA while improving those found to be problematic when working with SA. The framework was implemented in C++, principally by Janne Kätevä.

The thesis author's contribution to DBSA is limited to participation in the design of the software and database architecture of the framework. SA, in contrast, is entirely the author's original work, although some fundamental architectural concepts are inherited from an earlier framework with the same name. The following discussion therefore focuses on the SA framework as the principal software contribution of the thesis, whereas DBSA gets a more succinct treatment that mainly brings up those parts of it where it differs significantly from SA.

Currently, neither SA nor DBSA are being actively developed and distributed, and they serve here primarily as vehicles for the presentation of the architectural aspect of the augmented process model. Although the frameworks have been shown to be viable as implementation tools for KD software, they are more valuable for the novel architectural ideas they embody. They should therefore be viewed as demonstrations of how the proposed data and workflow models could be represented in software architecture, rather than as polished products intended to replace other KD frameworks.

In the remainder of this section, Subsection 4.3.1 describes the data architecture of the two frameworks, discussing two major themes in particular: database management and data type handling. Subsection 4.3.2 describes the workflow architecture, discussing basic application elements, application architecture, and interfaces. Subsection 4.3.3 outlines the process of developing a KD application with SA and illustrates it with a running case.

4.3.1 Data architecture

The basic design elements of Smart Archive, as well as its name, are inherited from an earlier incarnation of the framework, described in (Laurinen *et al.* 2005). This, in

turn, was based on a concept originally introduced by Väinämö *et al.* (2000). The basic elements are filters, which transform data, sinks, which store it, components, which combine filters and sinks into units, and pipes, which couple components together. The name of the framework refers to selective storage of data, which was one of the primary goals of the original SA.

The fundamental idea behind the SA data sink is that since data storage capacity is relatively inexpensive, there is often no need to make early decisions on what data to discard and what to keep for possible later use. Instead, it is possible to keep all data, including all intermediate results, just in case they prove useful. The data sink is designed to make sure that inconvenience is not an argument against doing so. In the older version of SA, each component in an application was required to have a data sink because the next component in the processing sequence would read its input from the sink. In the new SA, the focus of the framework has shifted toward a more generic KD framework, and data is passed directly from filter to filter, leaving application developers free to decide which components in their applications need sinks.

On the other hand, the archival point of view is still more strongly represented in Smart Archive than it is in other KD frameworks. Data sinks as such are not unique to SA — they could hardly be, since access to data stores is essential to any KD application and therefore to any KD framework. However, whereas in other frameworks writing data into a database is treated as a processing operation like any other, SA elevates it to a special status such that each component in an application can both execute a processing operation, using its filter, and write the results into a database, using its sink.

Another important feature of data sinks is that they are two-way: a sink object can both write data into a repository and read previously stored data. This means that a component can pull previously generated results out of the sink and present them as its output instead of using its filter to produce new results. Thus, it is possible to restart a previously executed KD task from any intermediate step, for instance, if the user of the framework wishes to examine the effect of modifying a model parameter on the final results.

Filters and sinks are inserted into an application by wrapping them in components; each component has one filter and one (optional) sink. The choice of filter and sink can be made either statically, during application programming, or dynamically, during execution. The wrapping component serves two major purposes. Internally, it connects its filter to its sink, allowing the filter to store its outputs. Externally, the component provides a single interface to other components that hides the filter and the sink from

their view. The implication of this is that to generate its outputs, a component may either use its filter to compute them, or use its sink to retrieve previously computed outputs from the application database.

The persistent data store used by a sink may be, e.g., a relational database or an XML file; the filter does not need to be aware of the storage format because the sink takes care of all format-specific details and hides them behind its external interface. One implementation of the sink interface can handle, for example, all RDB storage requests, so it is relatively rare for a developer to need to write a new sink. The RDB sink included in the framework automatically generates table definitions from filter output specifications, so a relational database capable of storing all the data processed by an application can be generated simply by attaching an RDB sink to each filter.

To cope with the vast range of different data types that a KD framework may be expected to handle, Smart Archive employs a data type engine that allows developers using the framework to use any conceivable data type in their applications. The type engine comes with a preprogrammed set of most commonly needed elementary and compound types, and developers can implement other types when they need them. Like other application elements, data types are reusable, so once a type has been implemented for one application, the same implementation can be used in any other application where the type is needed.

The purpose of the type engine is to take care of conversions of variable and parameter values from one format to another such that the conversion process is transparent to application developers. Format conversions are necessary because the framework has to be able to represent values in several different contexts: for each data type, there is an internal representation and a number of external representations. For instance, if a variable value is to be written into a relational database, the framework must convert the value into a string representation that can be appended to an SQL query. Similarly, if a parameter value is to be written into a configuration document, the framework must convert it into an XML representation. The type engine handles conversions from internal to external representation, and vice versa.

The type engine consists of a type dictionary and one or more type renderers. The type dictionary is an XML document that specifies the name of each SA type, the names of corresponding types in internal and external contexts, and the class name of the type renderer that handles the type. It is also possible to define type parameters, e.g., the type of collection elements in a collection type. The type renderers are objects that convert values between different representations. When an SA application is started, the

framework loads existing type renderers based on entries in the type dictionary. Then, when a conversion is required, SA looks up and invokes the appropriate type renderer, making it unnecessary for the application developer to implement the conversion.

In addition to the data type engine, SA includes features designed specifically to support KD from time series data. This class of data has certain special qualities that never become an issue when dealing with fixed-size datasets of independent observations. For example, the extent of a time series dataset is not necessarily limited, but with SA it is possible to buffer the input data and process it in small increments even while new observations are being generated. New data accumulates in the input buffers until the application is ready to process it.

Besides managing the input buffers and controlling the admission of new data for processing, SA can also extend incoming data items with a set of metadata variables that may be useful when mining time series data. These variables include timestamps and grouping information. Grouping variables can be used to indicate the beginning and end of interesting events in a time series, an event being, e.g., a manufacturing cycle or an athletic performance. Using these metadata variables is optional.

The name of the DBSA framework derives from its basic architectural unit, the device, which is equivalent to the component in Smart Archive. A DBSA device may represent either a physical device, such as a sensor, or a logical device, i.e., a data processing operation with no physical counterpart. Like components, devices produce output that other devices can read and process further. Connections between devices are defined in an XML configuration file that the framework loads on start-up.

Database integration in DBSA is similar to Smart Archive in that the framework automatically generates RDB tables based on the structure of the data generated by each individual device. However, instead of having separate data sink objects attached to the devices, the devices access the database interface of DBSA directly. A new database table is generated for each dataset processed by each device in a DBSA application.

In addition to the dynamically generated data tables, the DBSA framework provides a static set of metadata tables, which provide a way to locate and access the automatically generated data tables. The data model of the metadata schema reflects the architecture of the framework, enabling a DBSA application to find, for example, a dataset generated by a given device at a given time. The metadata schema and the associated data manipulation interface constitute what in the terminology of DBSA is called the knowledge base.

As with SA, DBSA application developers can extend the framework by implementing their own data types when a suitable type is not provided by the framework. DBSA,

however, does not have a special type management subsystem for this purpose. Instead, the framework requires the type objects themselves to implement certain operations such as serialization of data for insertion into a database. This seems awkward if one considers the representation, say, of a single double-precision number, but it is highly unusual for a DM application to use just one variable. Typically, the number of values passed between devices is greater, in which case wrapping the values in a data type object makes more sense.

4.3.2 Workflow architecture

Application frameworks can be classified into white-box and black-box frameworks based on how the functionality of the framework is exposed to developers so that they can extend it to create applications (Fayad & Schmidt 1997). White-box frameworks expose the actual class structure of the framework, allowing developers to extend it by inheriting their own classes from framework classes; black-box frameworks, in contrast, only expose specific interfaces where developer-created components are plugged in. More restricted access means simpler development but less control over the final product, so this property is an important factor in determining the type of development a given framework is most suitable for. Smart Archive gives developers the freedom to derive their own filter, sink and component classes from the base classes provided by the framework, so in this sense, it is a classic white-box framework.

On the other hand, SA has a layered architecture that adds a degree of subtlety to its otherwise straightforward white-box nature. Each layer forms an additional level of abstraction, using services provided by layers below it and providing more sophisticated functionality to layers above it. Developers are not forced to use all framework layers, so they can decide which parts of their application logic they wish SA to handle, include the appropriate layers in the application, and implement the remaining logic themselves. A developer can thus shift towards a more black-box-like development style by choosing to work at a certain level of abstraction and act as if the lower levels were hidden from the developer. The architectural layers of SA are shown in Figure 9.

Filters in Smart Archive, as in all pipes-and-filters style frameworks, represent data transformations that, when executed in correct sequence, implement the core functions of an application. In KD, the required transformations are derived from the KD process, so in KD applications, there will be filters for such tasks as feature extraction, classification, clustering, regression, visualization, etc. A filter may have any number of

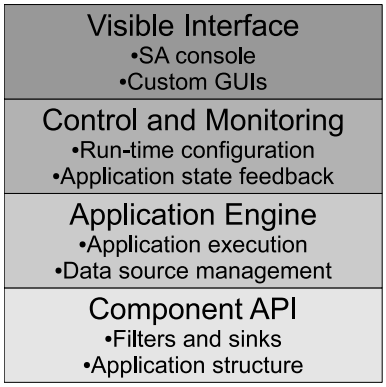


Fig 9. The architectural layers of the Smart Archive framework. A typical SA application uses at least the component API, the application engine, and some elements of the control and monitoring layer.

inputs, outputs and controlling parameters; a detailed description of these constitutes a definition of the external interface of the filter, allowing a developer to decide whether the filter is appropriate for their application.

The role of SA components, as explained in the previous subsection, is to encapsulate the functionality provided by filters and sinks. The component implementation included in Smart Archive handles this task in a highly generic fashion, so a developer will almost never need to write a new component class. The same holds for pipes, whose task is to transfer data from one application element to another. Pipes are used both to connect a component to other components and to connect the internal elements of the component to one another. Figure 10 illustrates this. Placing a pipe between two application elements requires that the names of the output variables provided by the origin agree with the names of the input variables expected by the destination; if this is not the case, variable name mappings can be defined for the pipe to correct the situation.

The components of an application, together with the pipes (links) connecting them, form a directed graph. This is known as the execution graph of the application. The execution graph has a single root node, the starting component, from which the graph may branch out arbitrarily. Cycles in the graph are not allowed. The execution graph class has the ability to run itself, i.e., to activate the filter of each component in turn and transfer its outputs to components that require them as their inputs. The class uses the structure of the graph to determine the correct running order: each component must be executed exactly once, and no component may be executed until all of its inputs have

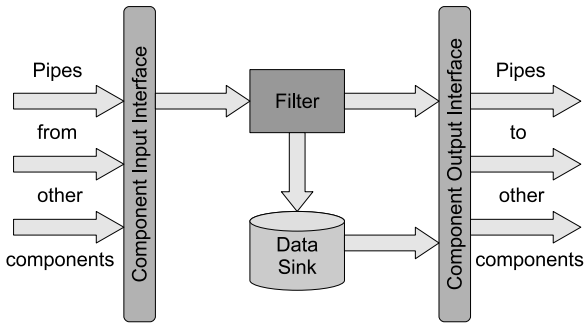


Fig 10. A Smart Archive component. Input data comes via pipes to the input interface and further on to the filter. The output data of the filter is similarly piped to the output interface and further on to other components. Alternatively, the outputs can be retrieved from the sink, which keeps data generated by the filter in persistent storage.

been supplied by previously executed components. The execution graph is illustrated in Figure 11.

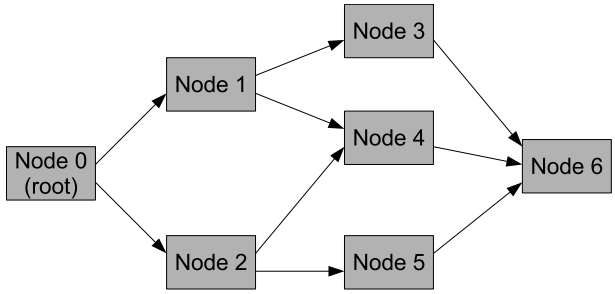


Fig 11. An example execution graph, illustrating how the graph may branch out from the root node. The numbering of the nodes reflects their execution order.

The execution graph is controlled by an object known as the mining kernel. The main task of the kernel is to feed input data to the execution graph and instruct the graph to process it, repeating this cycle until all the available data has been consumed or the process is otherwise terminated. To get the input data, the kernel interacts with one or more objects known as input receivers. Each input receiver reads data from one data source (a sensor device or a database, for instance) and writes it into one or more buffers, from which the kernel transfers the data to the execution graph.

In Figure 12 is a top-level view of the architecture of a Smart Archive application. In addition to the parts described so far, the figure includes a user interface, presented to

end users so that they can control the application while it is running. Through the kernel, the UI can start and stop the data processing cycle, change the controlling parameters of application elements, replace the application elements themselves, and alter the structure of the execution graph. The framework provides a console UI where some of the basic kernel functions (starting, stopping, displaying configuration information, etc.) can be controlled with text-based commands, but if this is not sufficient, developers are free to implement the UI however they wish, using hooks provided by the framework API.

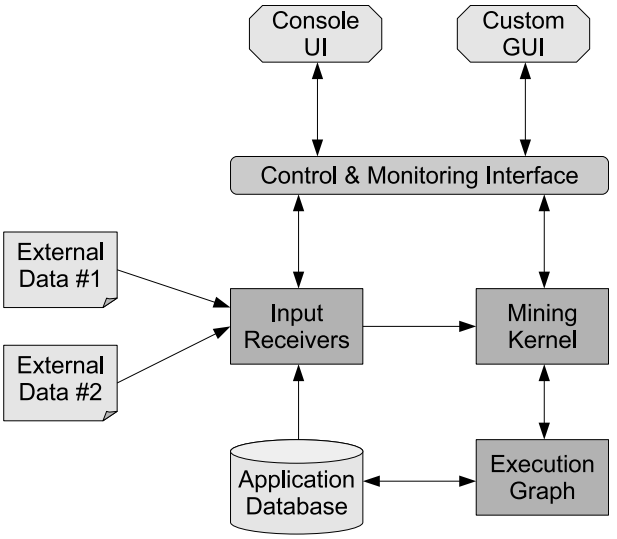


Fig 12. The major components and data flows of a Smart Archive application. The bulk of the application logic resides in the mining kernel and the input receivers, which the application developer controls via the control and monitoring interface. Note that the input receivers may access the internal database of the application instead of, or in addition to, external data sources.

The information required to set up a Smart Archive application is collectively known as an application configuration. A configuration consists of the following items:

- application name and description;
- for each component, filter and sink:
  - class name;
  - object name and description;
  - parameter values;

– for each inter-component link:
  – origin and destination;
  – variable name mappings;
– for each input receiver:
  – class name;
  – object name and description;
  – parameter values.

The framework allows the configuration to be exported as XML. The kernel accomplishes this by requesting each application element in turn to generate an XML description of itself and then compiling the descriptions of individual elements into a single XML document. Similarly, when a configuration is imported, each individual application element is responsible for setting itself up; the kernel only instantiates the corresponding class and sends to the resulting instance an XML object, extracted from the configuration document, specifying how the instance should set its properties. The framework provides abstract superclasses that implement the required XML export/import functions, so developers do not need to write them for their own classes; subclassing the appropriate SA class solves the problem.

During execution, a Smart Archive application has three main sections running in parallel: the input section, consisting of the input receivers, the processing section, consisting of the components and links of the execution graph, and the control section, consisting of the UI components. The mining kernel serves as a coordinator that enables the sections to communicate in a thread-safe manner. Most importantly, the kernel manages the input buffers, preventing unpredictable behavior that might occur as the result of poor synchronization as the input receivers and the execution graph try to access the buffers simultaneously.

The framework provides several synchronization-related properties that developers can adjust to control the flow of data through their applications. The rate of consumption and replenishment of buffer data can be regulated, and synchronization between input sources can be toggled when the application has more than one. These features complement the configuration-related control features of SA, enabling the framework to be used to develop both interactive applications and autonomous ones that stay in the background and process new data as it becomes available. Both operation modes are demonstrated in the application described in Subsection 6.2.1.

It is sometimes desirable to use something other than an SA filter for processing data in an SA application. Typically, this would occur during algorithm development, since there are specialized tools available that are more suitable for this type of work than generic programming languages such as Java. However, an algorithm must be implemented in Java as an SA filter if it is to be tested together with other application elements. To address this problem, Smart Archive introduces proxy filters.

A proxy filter in SA is a filter that acts as a placeholder for an algorithm implemented using an external tool. To the framework, a proxy filter appears no different from a regular filter, but instead of processing data, a proxy filter sends it to the external tool for processing. Once the external tool has executed the algorithm, it returns its results to the proxy filter, which presents them to the application as its output. To the application, this appears exactly as if the proxy filter had generated the results itself. The framework provides a ready-to-use proxy filter implementation, as well as an interface that developers can implement to create new ones.

Proxy filters are particularly useful in application prototyping. Using proxies, it is possible to compose a complete application structure at an early stage of development: previously implemented regular filters are used wherever possible, and proxies otherwise. When the proxies are coupled with suitable external tools, this arrangement amounts to a functional prototype of the application. The prototype can then be tested as a whole, using the external tools (e.g., scientific computing environments such as MATLAB) to make rapid modifications to those algorithms that have not been finalized yet. Once the final form of each algorithm is decided, it can be implemented as a regular filter, which then replaces the corresponding proxy in the application.

Replacing a proxy with a regular filter is a very simple operation, provided that the latter retains the input and output interfaces of the former. From this, it follows that transforming a prototype into a deployable application is also a relatively straightforward operation once all the required filters have been coded. Smart Archive thus makes it a viable option to build a working prototype early on, as soon as the application design has been finalized. Conceivably, proxy filters could also be used, e.g., to build distributed applications, in which case they would provide access to remotely hosted algorithms and would not be replaced in the final application.

The DBSA framework, like SA, is intended to be portable to multiple platforms. However, whereas SA is implemented in Java to ensure portability of the compiled binaries, DBSA is written in C++ and relies instead on the portability of its source code. This approach is commonplace in the open source community, where many projects are

distributed under the assumption that those who are interested have the required tools and knowledge to build the project sources on their platform of choice. On the other hand, many projects also offer pre-built binaries for the most popular platforms as a convenience.

To keep the source code portable, DBSA mostly stays within the bounds of the C++ standard, the main non-standard dependency being the Boost libraries. Boost itself is an open-source project and the libraries it provides are portable, so depending on it does not compromise the portability of the DBSA framework. Several Boost libraries have also been proposed for inclusion in the C++ standard (Auster 2005). DBSA relies on Boost for general-purpose utilities such as memory management and multithreading.

DBSA provides a few ready-to-use devices, e.g., a device for importing data from a wireless sensor box, and a code template for developers to use when writing their own devices. The template illustrates how to access data from other devices, process it and make the results available in outgoing data buffers. Unlike SA components, DBSA devices actively scan and process data from the devices they are listening to, whereas in SA, components wait for the framework to transfer the data and trigger the filter.

Unlike SA, DBSA does not have input receivers as a distinct class of objects. Instead, the task of reading input data into the framework is handled by devices. In the terminology associated with the framework, these are sometimes called physical devices, since in many cases, they represent actual physical sensors feeding data into the application. In contrast, devices with no physical counterpart are called logical devices. This is just a semantic distinction, however; from the perspective of the framework, there is no difference between physical and logical devices other than physical devices not accepting input from other devices.

DBSA has a console UI similar to the one provided by Smart Archive. However, while similar in functionality, the UI plays a more important role in DBSA than it does in SA. This is because with DBSA, the UI is the principal method of controlling the framework: unless the framework itself is modified, a DBSA application can only be executed by invoking the DBSA console, which configures the framework as instructed in the configuration file specified at start-up. With SA, both the console and the configuration file are optional; the primary method of configuring and controlling the framework is by code.

Integration of external tools to DBSA applications differs from the proxy filter approach of Smart Archive: a tool-specific interface must be used for each external entity that the developer wishes to integrate. The solution employed by SA is more

generic, but on the other hand, using non-generic interfaces is generally more efficient in terms of computing resources due to there being fewer abstraction layers creating overhead. Interfaces to MATLAB and Python are demonstrated in (Kätevä *et al.* 2010).

4.3.3 Developing applications

As an illustration of the process of developing a KD application using the SA framework, we will consider a running case that addresses all the steps required to design and implement an application. The application developed in the running case is a system for real-time assessment of the quality of resistance spot welding joints using data mining. Although the application as described is fictitious, it is based on actual research referenced in the discussion of the spot welding database in Subsection 6.1.1, and includes also elements of the predictive model maintenance system described in Subsection 6.2.1. The purpose of the running case is to show how the various features of SA support the work of KD software developers throughout the development process.

The first step in creating a Smart Archive application is designing the application. The design task is composed of the following activities:

1. Identify required processing steps.
2. Identify steps that can be implemented using existing component and filter classes.
3. Design required new component and filter classes.
4. Identify required data sources and repositories.
5. Identify sources and repositories that can be accessed using existing input receiver and data sink classes.
6. Design required new receiver and sink classes.

The example application has two operation modes: learning mode, where a predictive model is trained for a new welding process, and production mode, where a previously trained model is applied to obtain quality predictions for welding spots. In each mode, there are the same four processing steps, but with subtle differences in what data is processed and what is produced as output:

1. Preprocessing: standardization of data formats, normalization of values.
2. Feature extraction: calculation of variables suitable for classification.
3. Classification: training of predictive model (learning mode); prediction of welding results (production mode).

4. Presentation: visualization of classification performance (learning mode); visualization of welding results (production mode).

For the sake of simplicity, we will assume that each of these steps will be carried out by a single filter object, with parameters to adjust its behavior to the operation mode of the application. However, pools of candidate filters are required for the feature extraction and classification steps, since there are several different feature sets and classification algorithms that are potentially applicable. Searching for good candidate algorithms for these steps represents the bulk of the filter design effort in the development of the example application. The proxy filter is a useful tool for discovering candidates through experimentation and gradual modification — in essence, this means building an application based on the framework to help design a more complex application based on the framework. For the preprocessing and presentation steps, one filter for each step is sufficient.

The main data sources for the example application are the welding equipment set-ups, which generate data on the behavior of electric current and voltage during the welding event. Features for classification are computed from the current and voltage signals. Assuming that there is a standard interface through which the data is exported, a single input receiver class is sufficient for receiving data from welding equipment. Additionally, for model training, the data must be supplemented with the correct values of the output (quality) variables, obtained, for example, by mechanical testing of the welding spots. The training data is imported into the application by another input receiver class.

In the learning mode, the application will use a relational database as a repository for partially processed training data. The RDB sink class provided by the framework is suitable for this purpose. Additionally, a repository is needed for application configurations verified to deliver good classification performance for specific welding processes. For this, a data sink is not appropriate, but some kind of interface for storing and retrieving configurations will need to be designed. The repository itself can be, e.g., a file system or an XML database.

With all the individual application elements designed, the remaining design activity is to build an application architecture from them. This involves planning how the elements will be arranged into a graph (the processing section), how the processing section will be supplied with data and triggered (the input section), and how the functions of the application will be operated and their results visualized (the control section). In

the example application, both operation modes use the same structure for the execution graph. A graphical user interface, representing the control section of the application, is designed for editing the configuration of the graph and switching between operation modes. The GUI also controls the reading and processing of welding data.

The first implementation step is writing code for those application elements that cannot be implemented using existing classes. In the best case, the entire application can be built from existing elements, but it is likely that at least some new filters will need to be programmed, unless a sizable library of them has already been accumulated in previous projects. New input receivers or data sinks are needed less frequently, since an existing one can usually be reused if the application only needs to access standard data stores such as relational databases. Components mostly just direct the flow of data to and from their own filter and sink, so writing a new component class is necessary only on rare occasions.

The next step is to package the chosen filters and sinks into components. This is a relatively straightforward task, since there is probably not a great diversity of component classes to choose from. Pipe couplings inside a component always follow the same pattern and are normally handled automatically by the component class, so there is not much that developers need to code here. Setting parameter values for components, filters and sinks forms the main effort instead.

With all the required components assembled and configured, the next step is to arrange them into an execution graph. This is done by creating pipe objects and using them to connect components. The pattern of connections determines where each component gets its inputs from, where its outputs are sent to, and in what order the components are processed. If variable name mappings are necessary between some of the connected components, they are set up in this phase. Finally, one of the components is designated as the starting component, which acts as the entry point of input data into the execution graph.

For the example application, the component and sink classes provided by the framework are sufficient. Input receivers are implemented for reading welding data from a database (learning mode) and from welding equipment (production mode). The new filters identified as necessary for the application are implemented and added to a library from which they can be pulled into application configurations. The setting of parameters for application elements is always initiated via the GUI by the application operator. In learning mode, the parameters are adjusted manually until a satisfactory solution is found to the current classification problem. In production mode, all parameter values are

retrieved from the selected configuration document. The structure of the execution graph is always the same, with four components representing the processing steps occurring in linear sequence.

If an application has more than one major function, it may be necessary for the application to have more than one distinct execution graph. If this is the case, the application logic must include code for setting up each graph and for switching between graphs as different application functions are activated. One way to do this is to create an XML configuration document for each required graph, so that when a given execution graph is needed, it can be activated by loading the corresponding configuration document.

The starting component, like all other components in an SA application, has one or more input channels. Input data enters the processing section of the application through the input channels of the starting component. Each of these channels must be served by an input receiver in the input section. This is not a one-to-one relationship, however; it is possible for a single input receiver to server multiple input channels, so an application with multiple input channels does not necessarily require multiple input receivers.

Like execution graph elements, input receivers are created by instantiating the appropriate classes and setting parameter values for the class instances. In addition to this, if the UI functionality of the framework is used, each receiver must be registered by declaring it to the UI manager. The input receivers and the mining kernel can then be controlled in a centralized manner; for instance, all application sections can be started simultaneously with a single method call to the UI manager. On the other hand, it is also possible to control each input receiver individually.

In the example application, although the structure of the execution graph remains unchanged, the elements of the graph and their parameters do change. In learning mode, there is a single configuration that the application operator modifies, switching filters and adjusting their parameters until satisfactory classification accuracy is achieved. In production mode, there can be any number of possible configurations in the repository, from which the operator selects the most appropriate one for the welding process to be executed. In either case, the application is supplied by a single input receiver, which is controlled by the application-specific GUI; the console UI provided by the framework is not used.

The task of creating the UI ranges in complexity from very simple to vastly complex. For example, if the application is intended to support interactive exploration of data, the UI must provide convenient access to a variety of selection, processing and visualization techniques, making UI design and implementation a major effort. On the other hand, if

the application is meant to run in the background with little or no human interaction, it may well be that the SA console provides all the required UI functions. In such cases, it is not necessary to write any UI code at all.

The final step in SA application programming is writing the main program of the application. The complexity of this task is, in a certain sense, inversely proportional to the complexity of the UI creation task: the more the UI is in control of the application, the less there is for the main routine to do. If the application is entirely UI-driven, the main program simply launches the UI and does nothing more, whereas in a non-interactive application, the main program has all the control all the time. In such cases, the effort of writing the program may be comparable to that of creating a complex interactive UI.

The example application is controlled by a custom GUI with five major functions:

– configuring the application for training;
– execution of training;
– storage of configuration in the repository;
– retrieval of configuration from the repository;
– execution of prediction.

The GUI uses the API calls provided by the framework to implement these functions. The main program of the application simply launches the GUI; from that point on, the execution of the application is driven entirely by the GUI.

5 Satisfying the requirements

With the requirements for the augmented KD process model established in Chapter 3, and the model itself presented in Chapter 4, this chapter begins the validation of the model with respect to the requirements. Two mutually complementary validation approaches are used: In this chapter, the requirements are mapped onto specific components of the process model to show that they have been accounted for in the model. The next chapter follows a more empirical approach, demonstrating compliance with the requirements through a series of case studies involving real-world KD and KR problems.

The remainder of the chapter is once again organized to reflect the composition of the proposed process model. Therefore, Section 5.1 evaluates the fulfillment of the requirements for the data model, Section 5.2 does the same for the workflow model, and Section 5.3 for the architectural model.

5.1 Data requirements

The *data management* requirements for the data model state that the model should provide a descriptive framework for data transformation sequences, provide guidance for designing data repositories, and develop criteria for knowledge validation based on a theoretical analysis of the concept of knowledge. These requirements are addressed in Subsection 5.1.1. The *knowledge representation* requirements state that the model should provide a KR scheme founded on the same theoretical basis and general requirements as the data management scheme. These requirements are addressed in Subsection 5.1.2.

5.1.1 Data management

> *D-01: The data model should provide a conceptual framework that identifies and characterizes each state that data passes through during the KD process.*
> *D-02: The conceptual framework should identify and characterize each transformation that causes data to shift from one state to another.*

Figure 2 in Subsection 4.1.1 shows a progression of data states from raw data to knowledge, with four intermediate states between them. The discussion associated with

the figure gives textual descriptions of the states shown in the figure. Furthermore, the discussion explains for each state how data in that state is treated to transform it to the next state in the progression. Requirements D-01 and D-02 are thus satisfied by the data model.

> *D-03: The data model should provide technology-independent guidance for the creation of storage structures that cover all the identified states of data.*
> *D-04: The guidance for the creation of data stores should address the design of access methods for storing, searching for and retrieving data.*

Figure 3 shows a generic design for a data repository. The design covers both data storage and access interfaces. The components, functions and content of the repository are shown in the figure and described in more detail in the associated text. The design does not assume any particular database model or implementation technology. Requirements D-03 and D-04 are thus satisfied by the data model.

The repository design is presented on a high level of abstraction, which is, in part, necessary in order to satisfy the requirement of technology-independence. However, the abstraction also means that there is a wide gap separating the repository design from its implementation in practice, limiting its usefulness as a design reference. To close the gap, concrete guidelines and examples should be provided. The case study in Subsection 6.1.1 is offered as an example of designing a relational database.

> *D-05: The data model should establish criteria for validating the knowledge generated by the KD process.*
> *D-06: The validation criteria should be based on a theoretical understanding of the concept of knowledge.*

In Subsection 4.1.1, the concept of knowledge is broken down into the three conditions of justification, truth and belief, following the definition of knowledge in classical epistemology. From these conditions are derived a set of principles intended to ensure that the outcome of the KD process has all the qualities expected of knowledge in the epistemological sense. Requirements D-05 and D-06 are thus satisfied by the data model. However, the model offers little guidance on how to uphold the established principles, leaving several major questions open and awaiting further research.

5.1.2 Knowledge representation

D-07: The data model should provide a KR scheme for the creation of a knowledge base.
D-08: The KR scheme should share a theoretical basis with the other components of the data model.

In Subsection 4.1.2, the discussion of the concept of knowledge started in the preceding subsection is continued, and pragmatic epistemology is used to derive a representation that emphasizes knowledge as a practical asset. The representation is composed of three components that can be stored in a data repository modified from the one presented in the preceding subsection. The modified design is shown in Figure 4 and explained in the associated text. Requirements D-07 and D-08 are thus satisfied by the data model.

D-09: The KR scheme should integrate with and satisfy the same requirements as the data management scheme specified by D-03 and D-04.

A comparison of Figures 3 and 4 reveals that the latter is almost identical in structure to the former. The data and metadata stores are found in both designs, the only difference being that the KB design inserts a third store between the access interfaces and the metadata store, corresponding to the third component of knowledge in the KR scheme. The two designs can thus be integrated into a single one that covers both data management and knowledge representation: for storage and retrieval of knowledge, the interfaces interact with the function store as in Figure 4, whereas for data management operations, they can communicate directly with the data and metadata stores as in Figure 3.

Compared with the repository design presented in Subsection 4.1.1, the modified design is equally abstract and therefore equally technology-independent. Requirement D-09 is thus satisfied by the data model. However, the high level of abstraction also has the same drawback in this case, and therefore, the generic design should be supplemented with guidelines and examples. An example of designing a KR system is offered by the case study in Subsection 6.1.2.

5.2 Workflow requirements

The *actor* requirements for the workflow model state that the model should provide a descriptive framework for the human and technological actors that participate in the

KD process. These requirements are addressed in Subsection 5.2.1. The *interaction* requirements state that the framework should also describe the interactions among the actors, that the model should map the interactions to process steps, and that the model should identify potential conflicts of interest between actors. These requirements are addressed in Subsection 5.2.2.

5.2.1 Actors

> *W-01: The workflow model should provide a conceptual framework that identifies the types of technological artifacts used in the KD process, and characterizes their roles and relationships.*

Figure 5 in Subsection 4.2.1 identifies four categories of technology used in the KD process, arranged in hierarchical layers. The associated text gives more detailed descriptions of the technological artifacts in each category and the tasks they may perform. Both vertical relationships between artifacts belonging to different categories and horizontal relationships between artifacts belonging to the same category are described. Requirement W-01 is thus satisfied by the workflow model.

> *W-02: The conceptual framework should identify the human actors responsible for executing the KD process, and characterize their contributions and expectations.*
> *W-03: The conceptual framework should identify the human actors who otherwise hold a stake in the KD process, and characterize their contributions and expectations.*

Subsection 4.2.1 introduces the concept of a stakeholder to distinguish between human and technological actors. The five groups of stakeholders involved in the KD process are identified and described in Figure 6 and the associated text. Two expert groups are identified as the primary stakeholders whose contributions drive the process. Additionally, two non-expert groups are noted as possibly having a contribution but not being among the principal process drivers. Society is treated as the fifth stakeholder group due to its role in shaping the conditions under which KD projects take place.

Figure 6 shows both the five stakeholder groups and the relationships among them, expressed in terms of the expectations that each stakeholder needs others to fulfill. Requirements W-02 and W-03 are thus satisfied by the workflow model.

5.2.2 Interactions

W-04: The conceptual framework should identify the interactions that take place between human actors, and characterize their significance in the KD process.
W-05: The conceptual framework should identify the interactions that take place between human actors and technological artifacts, and characterize their significance in the KD process.
W-06: The conceptual framework should identify the interactions that take place between technological artifacts, and characterize their significance in the KD process.

Figure 8 in Subsection 4.2.2 shows the two main categories of actors, the actors within each category as identified in the preceding subsection, and the interactions that take place within and between the actor categories. The associated text supplements the figure with some discussion of the nature of the interactions, but focuses more on the relative importance of the different types of interactions and the current state of KD research and practice with respect to them.

The discussion particularly notes that interactions involving human actors, and especially those where both parties are human actors, are more poorly understood than interactions involving technology. Beyond this, however, not much is offered as contribution toward a better understanding of these interactions. Requirements W-04, W-05 and W-06 are thus satisfied by the workflow model, but in addressing these requirements, the model leaves a topic of major importance unexplored.

W-07: The workflow model should provide guidance for mapping the interactions that make up the KD process onto a flowchart of process steps.

Table 4 maps the interactions of the workflow model onto the steps of the CRISP-DM process model. In the resulting hybrid model, the high-level process flow is specified by the established model, while the finer structure within process steps is provided by the augmented model. Requirement W-07 is thus satisfied by the workflow model by making use of the existing standardized process model of KD. However, it is by no means clear or even likely that such straightforward reliance on the CRISP-DM model is the optimal way to guide the flow of the augmented process model; it should be treated as a tentative proposal, to be evaluated and refined through further research.

W-08: The workflow model should identify any potential conflicts of interest that may result from interactions between actors in the KD process.

The discussion in Subsection 4.2.2 identifies the non-expert stakeholder groups as the main source of potential conflicts. This is because non-experts, like experts, contribute resources (such as personal data or processor cycles) to a KD effort, but unlike experts, they are likely to have expectations that are not aligned with the objectives of the effort. The possibility of conflicts between experts is acknowledged but not examined further, and conflicts with society are only given a superficial treatment in the preceding subsection. Requirement W-08 is thus partially satisfied by the workflow model, but additional work is needed to plug the gaps pointed out here.

5.3 Architectural requirements

The *data* requirements for the architectural model state that the architecture should provide an interface to data sources, assist in the generation of storage structures and metadata, and allow any database technology and any data type to be used. These requirements are addressed in Subsection 5.3.1. The *workflow* requirements state that the architecture should support construction and execution of KD workflows, that it should provide a library of reusable components, and that it should provide interfaces for extension and communication. These requirements are addressed in Subsection 5.3.2. Finally, the *quality* requirements state that the architecture should promote reusability, expose multiple levels of abstraction to developers, be portable, deliver satisfactory performance, and cover the entire development lifecycle of a KD application. These requirements are addressed in Subsection 5.3.3.

5.3.1 Data architecture

A-01: The architecture should provide access to data sources via a transparent abstract interface.

In the SA framework, input receivers are used to import raw data into applications, while data sinks provide access to data repositories. In the DBSA framework, raw data is imported using devices, and all devices have access to a data repository via the DBSA knowledge base interface. In each case, the interface exposed to application developers abstracts away the mechanics of communicating with the data source,

enabling developers to access any data source using the same set of method calls. Requirement A-01 is thus satisfied by the architectural model.

> *A-02: The architecture should assist in the generation of a data repository structure that covers the entire KD process.*

In the SA framework, data sinks have the ability to automatically generate database tables with fields matching the names and data types of the output variables of each filter used in an application. In the DBSA framework, a new table is dynamically generated for each dataset produced and linked to the device that produced the data via a static metadata schema. An application based on either framework can store all the data it processes without requiring the developer or operator to set up a database. Requirement A-02 is thus satisfied by the architectural model.

> *A-03: The architecture should assist in the generation and management of metadata attached to the contents of the data repository.*

The SA framework offers limited support for metadata processing by providing a few metadata variables intended to assist in the management of time series datasets. The metadata schema of the DBSA framework is a more generic solution, but still somewhat restricted, being mainly useful for searching a database for a specific dataset. Requirements A-03 is thus partially satisfied by the architectural model, but additional work is required in order to more comprehensively support data management functions that rely on metadata. In particular, neither framework directly addresses the problem of backtracking a result to its supporting evidence; such functionality can be implemented by the developer using existing framework features, but it would be preferable for the frameworks to support it explicitly.

> *A-04: The architecture should not be bound to any database technology or vendor.*

The most convenient database management solution to pair with either SA or DBSA is the MySQL relational DBMS; using a different RDB product is possible through a minor development effort. Using non-relational (e.g., object-oriented) databases is similarly possible, but requires a more substantial effort, especially if the data sinks of SA or the metadata schema of DBSA are to be reimplemented. Requirement A-04 is thus satisfied by the architectural model, but better out-of-the-box support for different database technologies would be preferable.

A-05: The architecture should be able to process any type of data, including unstructured and multimodal data.

In the SA framework, the type dictionary and type renderers can be freely extended by developers to accommodate any data type not already supported by the framework. In the DBSA framework, a similar role is assumed by the abstract data type superclass, which developers can subclass whenever an appropriate type representation is not already available. Requirement A-05 is thus satisfied by the architectural model.

5.3.2 Workflow architecture

A-06: The architecture should support the construction of KD workflows in the form of pipe-and-filter graphs.

Both SA and DBSA use the pipes and filters style as their primary reference architecture, although the implementations differ. In SA, filter objects are packaged inside component objects, which are linked using pipe objects, and the framework pushes data through the network of components and pipes. In DBSA, the role of filter-component pairs is played by device objects, which have output queues that other devices can listen to and pull data from. Requirement A-06 is thus satisfied by the architectural model.

A-07: The architecture should support partial execution of workflows based on previous intermediate results.

In the SA framework, this functionality is provided by data sinks: the output of any filter can be stored in a sink, then pulled out and fed into the execution graph at a later time. In the DBSA framework, a similar outcome can be achieved by having a device use the metadata schema to find a previously stored dataset, retrieve the data and write it into its output queue. Requirement A-07 is thus satisfied by the architectural model.

A-08: The architecture should provide a library of reusable application elements implementing solutions to common problems.

Both SA and DBSA provide a small number of reusable application elements, most of which address various supporting tasks rather than actual data analysis. The SA framework, for example, comes with an RDB sink for storing and retrieving data, type renderers for handling various common primitive and collection data types, and a proxy filter for relaying data to an external tool for processing, but no filters implementing DM

algorithms and other important data transformations. The situation is largely similar with the DBSA framework. Requirement A-08 is thus satisfied by the architectural model only to a very limited extent, and integrating an algorithm library such as Weka would be a top priority in further development of either framework.

> *A-09: The architecture should provide internal interfaces for the purpose of extending the architecture with new application elements.*

The SA framework gives developers access to its abstract superclasses for components, filters, sinks, pipes, input receivers and type renderers, allowing any of these to be subclassed to replace the default implementations or to extend the library of reusable elements that the developers can use to build their applications. The DBSA framework allows developers to write new devices and data types, and provides templates to help them get started quickly. Requirement A-09 is thus satisfied by the architectural model.

> *A-10: The architecture should provide external interfaces for communication with other software, to support requirement W-06.*

The SA framework provides an abstract proxy filter superclass and one concrete implementation of it for the purpose of exchanging instructions and data with other software. The DBSA framework does not have a generic proxy interface, but does provide a number of product-specific proxy devices. Additionally, in both frameworks, the database interface can be used for this purpose, and in fact, the concrete proxy filter of SA uses a database for data transfer. Requirement A-10 is thus satisfied by the architectural model.

> *A-11: The architecture should provide external interfaces for communication with humans, to support requirement W-05.*

Both SA and DBSA come with a console-based user interface that enables operators to control framework functions at runtime using typed commands. Additionally, in the SA framework, the mining kernel provides control hooks that developers can attach their own custom user interfaces to. Requirement A-11 is thus satisfied by the architectural model, albeit with the notable shortcoming that neither framework provides a graphical UI for building application configurations. Creating one would be another objective of high priority in further development.

5.3.3 Architectural quality

A-12: The architecture should support development with and of reusable components.

The principal technique by which the frameworks facilitate reusability is decoupling, i.e., reducing dependencies that hinder the transportation of an application element to a different application context. The SA framework decouples components, filters and sinks from one another by never allowing them to exchange data except through pipes, which can be used to perform such data transformations as are required to match the output interface of the origin to the input interface of the destination. Furthermore, the components themselves represent an additional level of decoupling between filters. In the DBSA framework, decoupling is achieved through the use of asynchronous buffers and standardized data carrier objects to relay data from one device to another. Requirement A-12 is thus satisfied by the architectural model.

A-13: The architecture should grant developers access to it on multiple levels of abstraction, enabling them to control low-level functions.

The SA framework exhibits a layered architecture that directly addresses this requirement: developers can bypass the user interface, the application engine (mining kernel), and even the execution graph if they find it desirable to do so. The DBSA framework is closer to a black box: in its default configuration, it requires applications to be operated via the DBSA console. However, the framework can also be compiled as a dynamic link library (DLL), enabling it to be embedded in and controlled by custom application code. Requirement A-13 is thus satisfied by the architectural model, especially as expressed in the SA framework.

A-14: The architecture should be portable to multiple hardware and operating system platforms.

The SA framework, being written entirely in Java, is (theoretically) portable to any platform for which there is a Java Virtual Machine (JVM) implementation available. A C++ version was developed as a side project, aimed primarily at mobile platforms. The DBSA framework is written in standard C++ with portability in mind, the main nonstandard dependency being the Boost libraries. These, in turn, are portable libraries distributed in source code form. Requirement A-14 is thus satisfied by the architectural model.

A-15: The architecture should be able to deliver satisfactory performance under realistic operating conditions.

Both frameworks have been used to build applications that run in a real-world production environment or in a test environment that approximates real-world circumstances. Additionally, some individual features of the frameworks have been subjected to quantitative performance testing. However, the overall number of applications created is still small, and no rigorous and comprehensive scalability testing has been carried out to see how the frameworks perform when processing big data. There is therefore not enough evidence available to judge how fully requirement A-15 is satisfied by the architectural model.

A-16: The architecture should cover all the design and coding stages of the development lifecycle of KD software.

Both frameworks enable application architecture to be set up early on, before the algorithms to be used in the application have been selected. The architecture can be transformed into an executable prototype by populating it with dummy versions of the algorithms. Proxies can be used to experiment with different algorithms and parameters using an external computational tool such as MATLAB. By replacing the proxies and dummies with finalized versions of the algorithms written in the native language of the framework, developers can make a piecewise transition from prototyping and experimentation to implementation and deployment. Requirement A-16 is thus satisfied by the architectural model, although the process is not as efficient as it would be if a graphical application builder were provided.

6 Case studies

This chapter discusses four case studies involving concrete applications of the models presented in Chapter 4. Each application represents a previously published contribution in its own right, but in the context of this thesis, the primary role of the case studies is to support the arguments made in Chapter 5. Each application description is therefore followed by a discussion of its relevance to the models and their requirements.

The structure of this chapter deviates from that of the three preceding ones, because there is no simple one-to-one mapping between the case studies and the models. However, the data-management case studies in Section 6.1 pertain mainly to the data model, whereas the application-building case studies in Section 6.2 are more relevant to the architectural model. The workflow model is not addressed by any of the case studies, except indirectly through the features of the architectural model.

6.1 Managing data and knowledge

The case studies presented in this section primarily serve to show how the data model proposed in Section 4.1 can be applied in practice. The two applications discussed represent solutions to real-world problems in the domain of intelligent manufacturing. The principal objective of the case-study descriptions is to demonstrate how the data model translates into concrete databases and knowledge bases in the context of specific applications. To a lesser extent, they also demonstrate how the data model attains a concrete expression through the architectural model.

The first case study, described in Subsection 6.1.1, consists of a storage model and access interfaces for resistance spot welding quality assessment and process identification based on data mining. The storage model addresses the problem of managing the welding data being mined, using metadata to support search and retrieval operations. The interfaces provide graphical access to the data, supporting human-technology interactions between experts and the spot welding database. The welding database was originally reported in (Tuovinen *et al.* 2007).

The second case study, in Subsection 6.1.2, concerns a knowledge storage and retrieval system for reconfigurable manufacturing. The system performs task-to-method transformations, i.e., operations whereby a manufacturing task expressed in a machine-readable format is translated into a set of instructions for the machine that will execute

the task. The application represents a solution to a KR problem where a computer-based system maintains and applies a body of knowledge without depending on human supervision. Additionally, it serves as a demonstration of the DBSA framework as an application platform. A description of the system was originally published in (Tuovinen *et al.* 2010).

6.1.1 A database architecture for welding quality assurance

Resistance spot welding is a welding technique where a pair of electrodes is used to conduct an electric current through the workpieces to be joined. When the current is high enough, the resistance of the workpiece material causes it to heat and melt locally at the point of contact with the electrodes. When the current is switched off, the metal cools and solidifies, creating a spotlike joint between the workpieces. If necessary, this process is then repeated at different locations until the total strength of the individual welding spots is sufficient for whatever purpose the resulting object is intended to serve.

The quality of a welding spot can be determined by a physical test, but such tests increase the expenses of the manufacturing process. Thus, a more desirable alternative is to predict the quality of the spot automatically by observing and analyzing the welding event. This can be achieved by applying data mining methods to measurements of electrode current and voltage gathered during the execution of the welding protocol (Laurinen *et al.* 2004a, Junno *et al.* 2004b). To support this task, a database application was developed.

The welding database consists of four main types of data:

- raw data, consisting of current and voltage signals recorded in controlled welding tests;
- preprocessed data, consisting of filtered signals, including resistance and power signals derived from the measured data;
- features computed from the preprocessed data;
- metadata, consisting of human-readable descriptions of the attributes of the datasets stored in the database.

By providing storage for raw data, preprocessed data and features, the database satisfies the basic requirement of making it possible to record the intermediate states of the data mining process. The quality of welding spots can be predicted by applying a classification algorithm to a suitable subset of the features. The features can also be

used to assess the similarity of two welding processes; this is useful for the purpose of determining the optimal process parameters (Junno *et al.* 2004a, 2005, Haapalainen *et al.* 2005, 2006, Koskimäki *et al.* 2007).

The purpose of the metadata is to assist users in the task of selecting data to export from the database. The metadata section contains descriptions of attributes that the users can use as selection terms when formulating queries on the database. The metadata plays no role in the actual query, but it serves to remind the user of the names of potential selection terms and the syntax and semantics associated with them.

The welding database was implemented in the form of a relational database and three graphical interfaces for interacting with the database. One of these is a standalone application for importing raw data into the database. The other two are plug-in modules in an application created for processing welding data: one for exporting data for processing and the other for importing the results. The actual processing is done by other modules, executed in sequence in the style of pipes-and-filters architecture. Figure 13 illustrates this arrangement.

This case study is relevant principally to the following requirements:

> *D-03: The data model should provide technology-independent guidance for the creation of storage structures that cover all the identified states of data.*

The case study serves as an example of how to design a database to support a real-world KD process. In the original paper can be found an ER diagram that specifies the data structures used to store the welding data. However, the case study is not complete in addressing the requirement, for two reasons:

1. The database design is not entirely technology-independent. Although an ER model is an abstract representation that does not necessarily imply a relational database, it does imply a preference for such. In the case of the welding database, an RDB happens to be an appropriate solution, but it is unclear how the lesson might be generalized to cover other types of databases.
2. The database design covers only a part of the KD process. In terms of the data states identified in Subsection 4.1.1, the welding database accommodates the first three of a sequence of six states: raw data, prepared data, and condensed data. The database has a metadata section, but it is used only for searching, not tracking or semantic annotation.

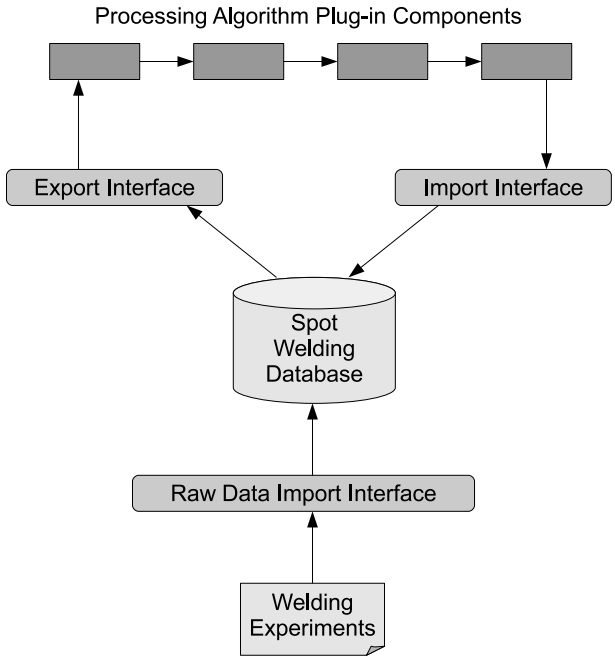


Fig 13. The spot welding database and its interfaces. The raw data import interface is used only once per test set, to store the original data. The data is then refined in cycles, so the export and import interfaces may be needed any number of times.

These limitations reduce the value of the case study as an illustration of how to apply the data model. Thus, while the case study is instructive, it is not sufficient by itself.

> *D-04: The guidance for the creation of data stores should address the design of access methods for storing, searching for and retrieving data.*

The import and export interfaces are a major component of the welding database, and the original paper provides details on how the interfaces are operated, including screen captures of their UI layouts. Technology-dependence is again an issue here, since the interfaces (apart from the raw data import interface) were designed specifically for the plug-in framework used in the project in which the welding database was created. However, the validity of the UI designs is not dependent on the underlying framework, and the design principles of the UIs are not specific to any application domain, either. The case study can therefore serve as an illustrative example of how to design interfaces that comply with the data model.

6.1.2 A task-to-method transformation system for reconfigurable manufacturing

Task-to-method transformation (T2MT) is defined here as a process whereby a formal description of a manufacturing task is transformed into a method for executing the task. The method is a set of instructions to the controller of the machine, generated such that when the method is executed, all the goals of the task are achieved and none of its constraints are violated. The T2MT system described below was developed for the purpose of giving production-line components the ability to independently reconfigure themselves when the line configuration changes.

The task description is an XML document containing all the information necessary for choosing a suitable method for the task. Most importantly, the task description specifies the physical objects involved in the task: tools, workpieces, fixtures, and obstacles. The objects can be arranged into various situations, e.g., one to represent the starting conditions of the task and another one to represent the objective. The task description also specifies what quality information (e.g., execution time) is expected to be produced when the task is executed.

Method descriptions, unlike task descriptions, have no common structure because the presentation format of the method depends entirely on the method consumer, leading to unlimited variation. A method for a welding controller, for instance, cannot be expected to bear much resemblance to a method for an assembly robot. From this, it follows that the T2MT system cannot be expected to understand the method description; it only needs to be able to identify the method consumer and choose a method that the consumer will understand.

Two instances of the T2MT system were deployed, both based on the same generic architecture, illustrated in Figure 14. The interface client sends the T2MT request over an HTTP connection to the interface server, which unpacks the request and hands it over to the session manager. The session manager analyzes the request and sends the included document (task or method description), along with instructions for further processing, to the parser. The parser processes the document and invokes either the method generator, if the document represents a task to be transformed into a method, or the method writer, if the document represents a method to be stored in the knowledge base.

The manufacturing task in both application scenarios was the same: handling a resistance spot welding gun. However, in the first scenario, the method consumer was a robot, whereas in the second one it was a human assisted by a guidance and

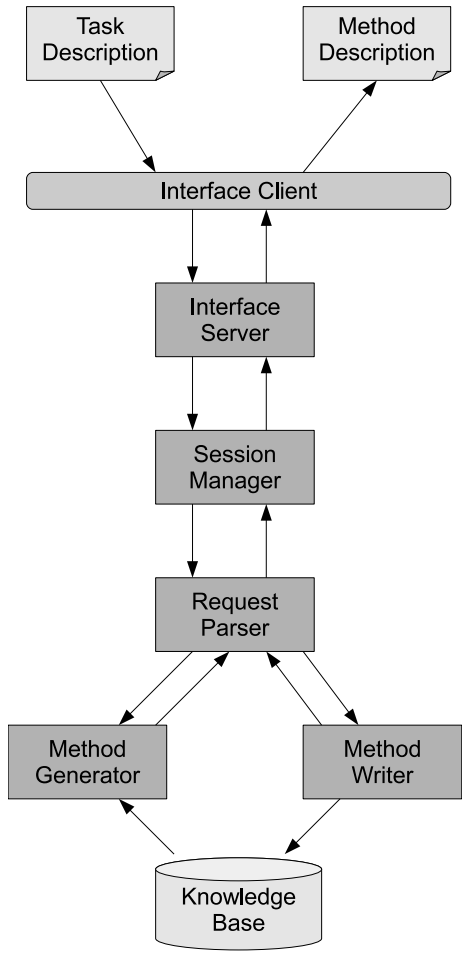


Fig 14. The software architecture of the T2MT system. The figure shows the case where a method retrieval operation is executed; in the other case, storing a new method, the input to the system includes both a task description and a method description, and the output is simply an acknowledgment that the requested operation was performed.

monitoring system displaying instructions and feedback. The methods in the first scenario — programs written in the control language of the robot controller — were therefore completely different from those in the second scenario, where the primary component of the method description was a recording of a previous execution of the same task, shown to the worker as a model for completing the task successfully.

Since the tasks were similar, the principle of storing and looking up methods was also similar in the two scenarios, even though there was no similarity between them in

the methods themselves. The manufacturing knowledge required by the T2MT process was represented as specified in Subsection 4.1.2, using data, metadata and functions to represent the passive, descriptive and active components of the knowledge, respectively. The T2MT system operates on this knowledge representation to match manufacturing tasks with suitable methods.

The data used by the T2MT system consists of methods. Each method in the knowledge base is there because it has been demonstrated to provide a satisfactory solution to some problem (manufacturing task). In both application scenarios, the methods were stored in an XML-based format, but this, as already stated, is unimportant from the perspective of the T2MT system: interpreting the contents of method descriptions is a task for method consumers only.

The metadata used by the T2MT system consists of information required by the system to determine whether a given method in the KB is appropriate for a given task. This information, in turn, consists of attributes of the task itself and attributes of the task execution environment (manufacturing cell). Finding a method for a task is based largely on comparing the task description to metadata stored in the KB, so object specifications, the most important part of the task description, are also the most important part of the metadata.

The functions representing the active component of the manufacturing knowledge handled by the T2MT system were implemented as embedded scripts. Each script is responsible for executing knowledge base queries, i.e., matching task descriptions with metadata, for one category of tasks, e.g., handling tasks performed by a robot. The scripting language is interpreted, so the system can select the appropriate script at runtime and execute it dynamically when a request arrives.

T2MT queries are executed in three phases such that each phase rules out candidate methods until, ideally, after the third phase there is exactly one candidate left. In the first phase, the query script passes only those methods that are valid, i.e., appropriate for the task execution environment (causing no collisions, for example). In the second phase, with all invalid methods ruled out, the script passes only those methods that are adequate, i.e., achieve all task objectives. Finally, in the third phase, the script chooses the optimal method, i.e., the best known valid and adequate method, given a cost function. The query thus returns exactly one method, provided that there is at least one valid and adequate method in the KB and the result of the cost function is unambiguous.

This case study is relevant principally to the following requirements:

D-07: The data model should provide a KR scheme for the creation of a knowledge base.
D-08: The KR scheme should share a theoretical basis with the other components of the data model.
D-09: The KR scheme should integrate with and satisfy the same requirements as the data management scheme specified by D-03 and D-04.

As established in Subsection 5.1.2, the data model does provide a KR scheme, based on a theoretical understanding of the concept of knowledge and compliant with the other components of the data model. The case study provides an example of the KR scheme applied in practice: it defines a representation of manufacturing knowledge based on the three components specified by the data model, and presents a software architecture for a knowledge storage and retrieval system. Two application scenarios for the system are presented.

The main problem of this case study as an illustration of how to implement the data model is that the knowledge processed by the T2MT system is not arrived at through a process of knowledge discovery. Although the origin of the knowledge as such is not relevant to the KR scheme, this nevertheless means that there is no sequence of intermediate states of data leading up to it, so again the case study addresses only a fraction of the KD process. What has not been demonstrated yet is a complete implementation of the data model, with data management and knowledge representation integrated in a single system.

A-01: The architecture should provide access to data sources via a transparent abstract interface.
A-02: The architecture should assist in the generation of a data repository structure that covers the entire KD process.
A-03: The architecture should assist in the generation and management of metadata attached to the contents of the data repository.

The implementation of the system is based on the DBSA framework, and the knowledge base is implemented using the database interface and schema provided by the framework. The successful completion of the two application scenarios supports the claim that the framework is an appropriate tool for implementing the KR scheme defined by the data model. However, as already pointed out above, only the storage and retrieval of knowledge is addressed by the case study — not the generation of knowledge through KD. Furthermore, the system uses the DBSA database schema in a domain-specific

fashion that may not be readily generalizable to support a wide range of applications. The scope of the evidence provided by the case study on the data management capabilities of the DBSA framework is therefore quite narrow.

> *A-05: The architecture should be able to process any type of data, including unstructured and multimodal data.*

The system uses a number of custom data types: the manufacturing knowledge is represented in the form of task and method description documents, and additionally the devices of the system need to exchange information on how to process the documents. Furthermore, the data contained in the method descriptions differs considerably between the two application scenarios. The programmable data types of DBSA enabled all these to be processed by the framework, supporting the claim that the framework is capable of processing any data type.

> *A-06: The architecture should support the construction of KD workflows in the form of pipe-and-filter graphs.*
> *A-09: The architecture should provide internal interfaces for the purpose of extending the architecture with new application elements.*

The architecture of the system is based on a bidirectional pipes-and-filters workflow, implemented using DBSA devices. Several new devices were written for the purpose, in addition to the new data type classes created in order to accommodate the custom data types required by the application. These are the two most basic application-building features of the DBSA framework, and the successful completion of the case study indicates that they make the framework suitable for practical application development.

> *A-10: The architecture should provide external interfaces for communication with other software, to support requirement W-06.*

One of the external interfaces of the DBSA framework, the Python interface, is used extensively by the T2MT system. The scripts representing the active component of knowledge are written in Python, and additionally the interface server device uses Python scripts to process HTTP requests, making it easier to adapt the system to different application scenarios. The communication between the interface client and the interface server thus demonstrates a successful integration of the DBSA framework with another software system using the external interfaces of the framework, although the HTTP server itself is not part of the framework; for this, the interface server relies on Mongoose, a lightweight embeddable HTTP server implementation.

A-12: The architecture should support development with and of reusable components.

Both the architecture of the system and the individual devices were designed to be reusable across the application domain, so that the same system could be deployed regardless of the specific properties of the manufacturing tasks to which it is applied. The Python interface of the DBSA framework, as mentioned above, proved useful here, since it enables the behavior of devices to be modified without recompiling their C++ code. Conceivably, the system, or individual components of it, could be reused also outside the original application domain, but a new case study would be required in order to demonstrate this.

A-14: The architecture should be portable to multiple hardware and operating system platforms.
A-15: The architecture should be able to deliver satisfactory performance under realistic operating conditions.

The system was developed in a desktop environment, whereas the target environment was an industrial PC setup. Porting the system to the target environment proved problematic due to its reliance on external systems, particularly Python. It seems reasonable to assume that the bare DBSA framework, without any such dependencies, would port more smoothly, but the case study has no evidence to offer in support of this.

Owing in part to the difficulties encountered in deploying the system to the target platform, there is no data available on the performance of the system in a realistic manufacturing scenario, with the KB populated with real manufacturing knowledge. Tests performed in the development environment using simulated data indicate that the system can deliver acceptable search performance, but the tests simplify the actual application scenario considerably. For a more reliable estimate, the system would have to be tested under circumstances that approximate a real factory environment more closely.

6.2 Building knowledge discovery applications

The case studies discussed in this section serve first and foremost as demonstrations of KD software development using the Smart Archive framework. The two applications highlight different aspects of the workflow model presented in Section 4.2 and how the framework accommodates them. They also illustrate some of the diversity of the

application architectures that can be achieved using the framework. To some extent, the case studies address the implementation of the data model and the fulfillment of the general quality requirements as well. Both case studies were originally reported in (Tuovinen *et al.* 2008).

The first case study, in Subsection 6.2.1, concerns a system for training and maintenance of predictive DM models used in steel production. The system has two operation modes, with a straightforward sequential workflow in the basic mode where it monitors the performance of the models in an unsupervised fashion. When a possible need for model update is detected, the system switches to an interactive mode where a human expert is involved in a decision-making role. The case study demonstrates human-technology interaction and integration of an SA-based application with an existing information system.

The second case study, in Subsection 6.2.2, examines a distributed application for time series data mining. The application consists of multiple instances of the SA analysis engine executed on multiple hosts by human operators, in a manner that resembles volunteer computing. The case study demonstrates an arbitrary number of SA instances interacting in a coordinated fashion to form a larger system. The integration of the system is realized by means of the data model implementation of SA, which is used to set up a database-centered software architecture.

6.2.1 A model maintenance system for the steel industry

The model maintenance system is a semi-automatic system for monitoring and maintaining the performance of mechanical property models in the steel industry. The models are used to predict certain properties of steel plates based on various properties of the production process, most importantly the relative quantities of elements in the steel alloy. Such predictive models are important planning tools for production engineers. A more thorough discussion of the case can be found in (Juutilainen *et al.* 2011).

The reason why a model maintenance system is needed is the tendency of model performance to decay over time. Decay refers to the gradual loss of predictive accuracy due to changing circumstances, which makes regular recalibration of models a necessity. The model maintenance system is a tool that helps production engineers do this by alerting them to the situation whenever the accuracy of one of the maintained models drops below a given threshold. The same system can then be used to train a new model to replace the decaying one.

The models maintained by the system predict three properties of steel: tensile strength, yield strength, and elongation. For each property, there are separate models for expectation and variance. Thus, there are six models altogether, each represented in the application by a Smart Archive filter. The models themselves are statistical regression polynomials whose terms are stored in text files in a format that the filters can read. This makes it easy to update the models, since it is not necessary to modify the code of the filters.

The top-level software architecture of the model maintenance system consists of two processing subsystems, the model evaluator and the model trainer, and two data stores, the process database and the model repository. Each processing subsystem has access to both data stores. The organization of subsystems is illustrated in Figure 15.

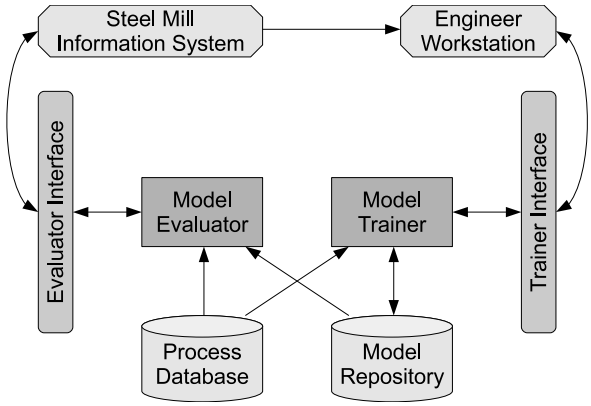


Fig 15. Subsystems and interfaces of the model maintenance system. Although the two operation modes of the application are implemented as two separate subsystems, the subsystems are connected via shared data repositories.

The two processing subsystems of the maintenance system correspond to the two operation modes of the application: the non-interactive mode is handled by the model evaluator and the interactive mode by the model trainer. As seen in Figure 15, the model trainer has a user interface for interacting with humans, whereas the model evaluator only interacts with other information systems of the steel mill.

The non-interactive operation mode of the maintenance system is invoked at steady intervals to evaluate the currently active models with new data accumulated in the process database. If the evaluator detects signs of performance decay exceeding what is considered acceptable, it alerts the engineer in charge of model maintenance. Since the

maintenance system is connected to the other information systems of the steel mill, it can notify the engineer via e-mail.

When the engineer responsible for the maintained models receives the notification e-mail, they invoke the interactive operation mode of the maintenance system. In this mode, the engineer can view the properties of the active models, as well as all other models stored in the model repository. If the engineer judges that the decay detected by the evaluator warrants further action, they can initiate the training of a new model.

If the engineer decides that the active model needs to be updated, they select a training dataset from the process database. The model trainer then uses the selected data to train new expectation and variance models. Once the training is complete, the engineer can compare the predictive accuracy of the new model to that of the active one. If the performance improvement is deemed sufficiently large, the engineer stores the new model in the model repository and designates it as the new active model.

This case study is relevant principally to the following requirements:

> *A-01: The architecture should provide access to data sources via a transparent abstract interface.*
> *A-02: The architecture should assist in the generation of a data repository structure that covers the entire KD process.*
> *A-06: The architecture should support the construction of KD workflows in the form of pipe-and-filter graphs.*
> *A-09: The architecture should provide internal interfaces for the purpose of extending the architecture with new application elements.*

The system developed in the case study makes heavy use of the database functionality of the SA framework. Especially the non-interactive operation mode, which estimates the need for model update, depends on a long processing pipeline in which several intermediate result datasets are generated and stored before the final result is produced. The successful realization of the application scenario indicates that the data repository interface of SA is well suited to implementing such applications. The processing pipelines themselves were realized by implementing the required algorithms as SA filters and assembling them into SA execution graphs, demonstrating the validity of the framework as a platform for component-based pipe-and-filter applications.

> *A-10: The architecture should provide external interfaces for communication with other software, to support requirement W-06.*

A-11: The architecture should provide external interfaces for communication with humans, to support requirement W-05.

The case study demonstrates successful integration of an application based on the SA framework with an existing information system (the non-interactive operation mode). Furthermore, communication with a human expert via a user interface is also demonstrated (the interactive operation mode). However, the implementation of these interfaces is not significantly aided by SA features specifically intended to support communication with external entities, such as the proxy filter. Therefore, the case study mainly serves to show that such interfaces can be created, owing to the white-box nature of SA that allows developers to embed the SA data processing engine in any application.

A-12: The architecture should support development with and of reusable components.

A good example of developing for reusability is provided by the implementation of the filters representing the models used to predict the mechanical properties of steel. A generic regression filter was first designed that reads the terms of the regression polynomial from a file supplied as a parameter to the filter object. This filter could then be used, with minor adjustments, to implement each of the three property models. Furthermore, since there is a vast range of possible applications for polynomial regression models, the generic regression filter has a high reuse potential and is a good candidate for inclusion in a library of general-purpose SA filters.

A-14: The architecture should be portable to multiple hardware and operating system platforms.
A-15: The architecture should be able to deliver satisfactory performance under realistic operating conditions.

Like the T2MT system described in Subsection 6.1.2, the model maintenance system was developed in a desktop environment. However, unlike the T2MT system, the model maintenance system was successfully deployed in a factory environment, providing an example of the portability of the SA framework. This deployment was experimental in nature, but a DBSA-based production-ready version of the system was later developed and deployed, demonstrating that at least the DBSA framework delivers sufficient performance to support real-world KD applications.

6.2.2 A distributed application for analyzing activity data

The distributed application is a system for analyzing time-series datasets representing physical activity measurements. The measurements are recorded from human subjects engaged in various forms of physical exercise such as running, cycling, or tennis. The measurements consist of various indicators of the subject's activity level such as two-dimensional acceleration values measured by a wrist-worn sensor device.

Activity data, such as that processed by the case-study application, has a number of uses related to fitness and health. One possible use is the identification of the type of activity performed, which in turn can be used to maintain an automatic diary of sports activities (Siirtola *et al.* 2009). Another one is the prediction of energy expenditure during an activity (Haapalainen *et al.* 2008a,b). The case-study application, however, is only a preprocessing system that prepares data for further analysis.

The data preparation task itself is somewhat uninteresting in this case; the principal problem here is the heavy computing involved. The application solves this problem by partitioning the data into chunks that can be processed independently from all others. These are then distributed to be processed in parallel by different computers, which significantly reduces the workload of an individual computer.

The distribution model of the application is depicted in the architectural diagram in Figure 16. The system is controlled by a broker unit, which gives instructions to an undefined number of worker units. All units have access to a central database, from which the data to be processed is read and into which the results of the processing are written. There is no direct link from the broker to the workers, so all communication between units is done indirectly, through the database.

The broker unit is in charge of partitioning the data to be processed. The unit is implemented as a Smart Archive application instance. Each partition is stored in the central database along with a tag that indicates it to be ready for processing. It is through this tag that the broker sends its instructions to the workers. A worker may choose to process any data chunk tagged as ready by the broker.

The worker units, like the broker unit, are implemented as SA application instances. When a worker unit selects a data chunk for processing, is changes the corresponding tag in the database to indicate that the chunk is being processed. This prevents other workers from taking on the same task. Once the task has been completed, the worker stores the results in the database and changes the tag to indicate that the selected partition has been

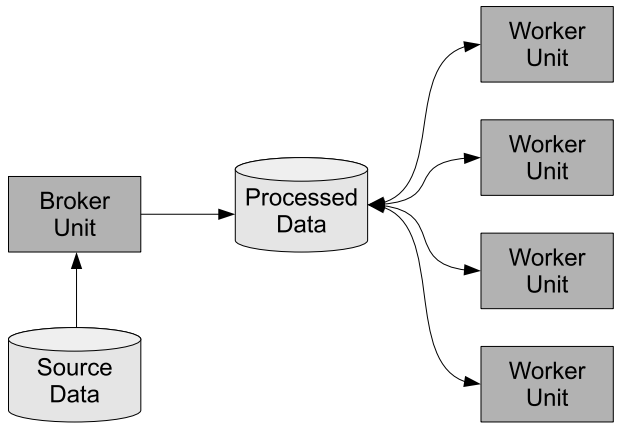


Fig 16. Distributed data mining with Smart Archive, using a database-centered architecture. The boxes represent individual SA instances, which communicate via the processed data repository.

processed. It then proceeds to scan the database for a new task that has not been claimed yet by another worker.

Being based on the SA framework like the model maintenance system, this case study addresses many of the same requirements as the previous one, and most of these will not be repeated here to avoid redundancy. However, the following requirements highlight aspects of the framework that the previous case study did not:

> *A-01: The architecture should provide access to data sources via a transparent abstract interface.*
> *A-05: The architecture should be able to process any type of data, including unstructured and multimodal data.*
> *A-10: The architecture should provide external interfaces for communication with other software, to support requirement W-06.*
> *A-15: The architecture should be able to deliver satisfactory performance under realistic operating conditions.*

Like the model maintenance system, the system developed in this case study uses the data repository interface of the SA framework to access the database. However, rather than simply using the database as a persistent store for the results generated, the system uses it to mediate communication between multiple SA instances running in parallel in a coordinated fashion. The case study thus demonstrates the use of the data repository interface for integration of systems composed of multiple applications, and also the use

of distribution to increase the overall performance of a system by harnessing additional computing resources. Additionally, the case study provides an example of using the SA framework to process time series datasets.

7 Evaluation of contributions

The purpose of this chapter is to present a qualitative summary of the strengths and weaknesses of the proposed augmented KD process model. The evaluation is based on all the material discussed in Chapters 2 through 6, and its aim is to build a balanced view of how the various components of the model contribute to the scientific value of the thesis. Another important objective of the evaluation is to suggest how the model should be applied and which parts of it should most urgently be improved, given the identified strong and weak points.

Section 7.1 discusses the merits of the proposed solution. Each of the three sub-models is discussed in its own subsection, with particular emphasis on what is novel about the models and what is the significance of the associated case-study applications. Section 7.2 is similarly partitioned into subsections and discusses the shortcomings of the models. There the emphasis is on notable omissions and the limits of the applicability of the models. Section 7.3 gives an overview of topics that are potentially relevant to the proposed process model but not addressed by the thesis.

7.1 Merits and significance

The data model covers the progression of data from raw data to knowledge as specified by the requirements defined for the model. It analyzes the concept of knowledge and applies the results of the analysis to KD, showing how various aspects of knowledge as an epistemological concept can be defined in a computing context. The model presents a reference architecture for data repositories that addresses both data management and knowledge representation, and the case studies associated with the model demonstrate a significant portion of the architecture in practice. The merits of the data model are discussed in more detail in Subsection 7.1.1.

The workflow model identifies the actors that participate in the KD process and the interactions between them, as specified by the requirements. Explicitly recognizing the role of ethical considerations in the process is a particularly novel feature of the workflow model. Conventional wisdom is also made part of the model by using the CRISP-DM model to guide the overall flow of the process. The merits of the workflow model are discussed in Subsection 7.1.2.

The architectural model has two concrete manifestations in the form of software frameworks for KD application development. The frameworks provide a number of novel features addressing various aspects of the KD process highlighted by the data and workflow models. Realistic case studies implemented using the frameworks demonstrate their suitability for programming KD applications. The merits of the architectural model are discussed in Subsection 7.1.3.

7.1.1 Data model

The description of data states and transformations presented as part of the data model lays the groundwork for all the other components of the augmented KD process model. This descriptive framework is the common root from which all three sub-models of the augmented model stem, expanding it in different dimensions. It is thus significant as an internal reference model through which the different aspects of the proposed process model are linked.

The next contribution of the data model is the abstract architecture for data management in the KD process. Its most significant impact comes from informing the design of the data management-related components of the architectural model. However, it also has a direct embodiment in the spot welding database described in Subsection 6.1.1, which is a demonstrably viable solution to the problem of data management in the context of DM-based quality assurance of welding joints. The validity of the concept and the potential of the technology have been proven (Laurinen *et al.* 2004a, Junno *et al.* 2004b,a, Haapalainen *et al.* 2005, 2006, Koskimäki *et al.* 2007, Tuovinen *et al.* 2007), and through its external interfaces the database integrated smoothly with the DM tool used to analyze the welding data.

With respect to the data model, the welding database represents an application-specific implementation of the storage of data at different points along the KD process. The implementation consists of a relational database schema and access interfaces for storing and retrieving data at three levels of abstraction: raw data, preprocessed data and features. These correspond to the first three states of data identified by the data model, and the solution can be viewed as a means of supporting the data transformations involving these states. The higher levels of the data hierarchy are not addressed, but the principle of the solution can be extended to cover them as well, and similarly, although the detailed design and implementation of the solution are specific to a particular application domain, the architecture is not.

The task-to-method transformation system discussed in Subsection 6.1.2 is a special case that specifically addresses the topmost part of the data hierarchy by supporting interactions with the most refined form of data, knowledge. Again, the solution consists of a set of interfaces and an underlying database schema on which the interfaces operate. There is a notable asymmetry in the way the different interfaces are intended to be used: the input of knowledge into the database is done by human experts, but the stored knowledge is looked up and used by other software applications. The idea is that human expertise in the form of semantic metadata has to be injected into the process in the input phase, but after that, the knowledge can be accessed independently by an application that executes queries on the metadata. This allows the T2MT system to be used as a subsystem in larger systems, as was done in the T2MT test cases (see (Tuovinen *et al.* 2010)).

Like the welding database, the T2MT system is restricted to a specific application domain, but one may ask how much genericity is actually required; the approach followed is not suitable for developing highly generic knowledge systems, but it is viable and potentially preferable in situations where a solution that covers a restricted class of applications is sufficient. Furthermore, the architecture and working principles of the system — its knowledge representation model and the associated search method, in particular — are more widely applicable than the system itself. The pragmatic concept of knowledge underlying the KR scheme is very generic, and since the concept of knowledge in KD is essentially pragmatic, it is natural to pair it with a KR model based on pragmatic epistemology.

Based on the case study, the knowledge system architecture underlying the T2MT system can be considered proven as a concept. The system makes innovative use of the pipeline architecture and data management functionality of the DBSA framework to set up a platform for knowledge base searches and other knowledge-based services. Its central session manager component plays the dual roles of maintaining session state, enabling the system to serve multiple clients in parallel, and directing service requests to the components that implement the corresponding services, allowing the addition of new services via the plug-in API of DBSA. The possibility of using Python scripting to modify component behavior dynamically provides additional flexibility.

It is a noteworthy feature of the T2MT system that it stores the passive and descriptive components of the knowledge — data and metadata — in an ordinary relational database. All data definitions are done in SQL, so there is no need to use an ontology description language to define the properties and relationships of tasks, methods and other related

concepts. Retrieving information from the KB is also based on SQL queries, constructed and executed by an intermediate interface written in C++. Similarly, the implementation of the active component of the knowledge does not require a special reasoning engine. The scripts that represent the active component in the case of the T2MT system were written the Python language, executed by the C++ code of the T2MT system. The scripts, in turn, use the C++ interface of the knowledge base to search the KB for methods.

The most purely theoretical contribution of the thesis is the observation that the theory of knowledge discovery is necessarily linked to the theory of knowledge. It was further observed that this connection is seldom made explicit, and that knowledge as an abstract concept is poorly understood in KD as a result of this. The data model applies epistemological thought and theory to the concept of knowledge as used in KD, uncovering the implicit assumptions and values embedded in it and pointing out aspects of knowledge overlooked by it. This work thus contributes to a better understanding of the theoretical concept, which can be transformed into a better understanding of how to generate and represent knowledge digitally. The results of the analysis are codified as a set of principles to be heeded by KD experts.

7.1.2 Workflow model

The other significant theoretical contribution of the thesis is also its binding theme: a remodeling of the knowledge discovery process is proposed, based on a nonlinear graph of interactions rather than a linear sequence of transformations. The intent is not to supplant the established process model but rather to extend or complement it by adding entities and relationships that affect the outcome of the process but are not represented in the established model. To integrate the established process model with the proposed augmentations, the workflow model presents a mapping of the activities of the augmented model onto the steps of the standardized CRISP-DM model. Thus, the established model can still be used to direct the overall flow of the KD process, while simultaneously using the augmented model to ensure that other important aspects of the process are not neglected.

The workflow model is based on two observations made by analyzing the concept of knowledge, especially those aspects of it that are ignored by the established model, and by studying real-world KD processes that pose problems not addressed by the established model. First, it was concluded that human actors are underrepresented in the established model, considering the diversity and weight of the roles they play in the

process. Furthermore, it was observed that the human aspect of the KD process is best described in terms of interactions, since humans are typically required in the process to perform interactive tasks such as negotiating with other humans or analyzing data using exploratory methods. As an extrapolation of this observation, it was seen that software artifacts can also be viewed as actors and therefore the process in its entirety can be represented by a model composed of interactions.

Extending the definition of actor to cover software as well as people might be dismissed as just a conceptual trick invented to make the model appear more coherent, but it is necessary if we are to account for the full variety of different ways to organize the KD process. The volunteer computing model examined in Subsection 4.2.2 serves well as an example of how the roles of computers and humans are sometimes overlapping or interchangeable, making the process much less straightforward than the conventional KD process model would indicate. Much of the significance of the workflow model lies in explicitly identifying the various actors, their contributions and expectations, and the interactions by which they make their contributions and satisfy one another's expectations.

An additional merit of the workflow model is that it accounts for the contributions and expectations of non-expert stakeholders and enforces consideration of how they should be treated by the experts who drive the KD process. Social and ethical issues such as privacy preservation are usually studied in isolation, but the thesis argues that they should be integrated into the process model to ensure that the rights of all stakeholders are properly respected. It is observed that doing so is both a moral obligation and a long-term practical necessity, since failure to execute the KD process in an ethical manner will reflect poorly on the responsible parties and diminish their chances of carrying out successful KD efforts in the future.

7.1.3 Architectural model

With respect to software architectural support for the KD process, the thesis makes its main contributions through the two knowledge discovery software frameworks, Smart Archive and DBSA. Smart Archive was the first to be developed, with the thesis author as the lead designer and implementer, based on earlier work by Väinämö *et al.* (2000) and Laurinen *et al.* (2005). DBSA is the more recent of the two and corrects some of the problems that were found to hinder KD application development with SA, but

the contribution of the thesis author, who only participated in the specification and architectural design phases of the work, is considerably smaller.

Both SA and DBSA have been demonstrated in practice to be applicable as platforms for KD software development, but in this respect, the frameworks are not by any means unique, since there are several other similar frameworks available, some of which were reviewed in Subsection 2.2.3. However, while the concept of SA and DBSA is not novel in itself, the frameworks do have certain novel features, developed in response to specific problems that are frequently encountered in the knowledge discovery process. These individual features represent much of the significance of the frameworks as research results.

One distinguishing feature of both SA and DBSA is how they handle interactions with data repositories. Both provide components for importing data into applications, but additionally, there is a direct link between algorithm components and data repositories. This is especially useful for writing the outputs of algorithms in a database, which the frameworks make fully automatic; the benefits of this are discussed in Subsections 3.1.1 and 3.3.2. Another benefit is that a component can use its inputs to retrieve additional data from a repository, either to support the transformation it performs or to use the retrieved data as its output. An example of the latter is found in the DBSA-based T2MT system, where the purpose of one component is to search the knowledge base, using data received from another component to build the search conditions, and return the results of the search.

Another special feature of SA and DBSA is data type management. Although the frameworks approach the problem from different angles, they are similar in that both impose certain conditions on the data flowing through the framework and provide an API for implementing data types that satisfy those conditions. Both solutions also offer similar benefits: interacting with data is made easier by automatic conversions between different representation formats, and interactions between algorithms are also helped because type management ensures that there is a common interface to data passed from one algorithm to another.

Indirectly, through the configuration file, data type management also supports interactions between a framework and its users. Both SA and DBSA use an XML-based configuration that defines the composition and operational parameters of applications, and one of the functions of type management is to control the representation format of parameter values. This holds especially for SA, where the same mechanism that converts data types between their internal and database representations also handles

conversions between internal representations and XML renditions. In DBSA, type management has a smaller role, being primarily concerned with data passed between components and stored in databases while leaving the interpretation of parameter values up to the components.

A distinguishing property of SA and DBSA that goes beyond individual features and into the realm of design philosophy is that compared to other KD frameworks, the design of SA and DBSA is based on a different approach to application development. While other frameworks tend to emphasize rapid development of applications that are constructed and executed via the same GUI, SA and DBSA conform to a more conventional development process model, especially SA with its layered white-box architecture. The choice of development approach depends on development goals, so the particular strengths of an SA/DBSA-style framework are useful in some projects, while in others it would make more sense to adopt one of the mainstream frameworks.

In summary, while SA or DBSA alone may be too cumbersome as an exploration and prototyping tool, either one is suitable for implementing the final solution once its design has been completed. Furthermore, both frameworks have a feature that can be of considerable use in the exploration phase: interacting with other applications. SA offers a generic API for external application proxies, whereas DBSA provides a small set of application-specific connectors, but both approaches lead to the same result: developers can opt to code and run some of their algorithms in an environment outside the one defined by the framework. This makes it possible to experiment with different algorithms and parameter sets using a more convenient tool such as MATLAB or R, the favored approach of the researchers whose needs the frameworks were originally designed to serve.

The other major use case for this feature is building the final solution such that it is composed of multiple collaborating applications. This is the system level in the technology hierarchy and has been experimented with using both SA and DBSA, although not in solutions intended for a production environment. However, the principal reason why the feature was considered desirable is that it supports the working style of the researchers involved in the development of the frameworks, which in itself is another example of the interactive nature of the KD process and of the importance of human involvement in it: while the established KD process model may be an apt description of how the final solution works, the process of designing the solution is much less straightforward.

In terms of real-world impact, the most significant application contribution of the thesis is the predictive model maintenance system discussed in Subsection 6.2.1. The system is now deployed for production use at a steel mill, although the version in use is not the original Smart Archive-based implementation reported in (Tuovinen *et al.* 2008) and (Juutilainen *et al.* 2011). Instead, the application has been reimplemented using the DBSA framework, which became available later. Also, since most of the design and implementation work on the application was done by co-authors of (Tuovinen *et al.* 2008) and not by the thesis author, its main significance here is as a demonstration of the capabilities of SA and DBSA rather than a contribution in its own right.

7.2 Omissions and restrictions

Although the data model addresses the requirements defined for it, there are parts of it that are too abstract to apply directly. The case studies aim to alleviate this problem by serving as reference implementations, but their applicability suffers from the fact that they address a relatively narrow application domain. Furthermore, the case studies do not, either separately or together, cover the entire KD process as captured by the data model. The flaws of the data model are discussed in more detail in Subsection 7.2.1.

The workflow model similarly addresses its respective requirements, but in doing so it is relatively simplistic. The model does not go very far beyond naming the various actors and requirements, and while it highlights interesting problems related to, for example, the privacy of data subjects, it does not have much to offer toward solving such problems. The manner in which the CRISP-DM model is tied in is very straightforward and requires better justification. The flaws of the workflow model are discussed in Subsection 7.2.2.

The current implementation of the architectural model, as manifested in the two software frameworks, is behind the state of the art in certain important respects. Also, the number of case studies implemented is small, which raises the question of how much the existing case studies can really tell about the applicability of the frameworks. The range of situations in which choosing one of the frameworks described in the thesis would be justified is severely limited by these considerations. The flaws of the architectural model are discussed in Subsection 7.2.3.

7.2.1 Data model

It may be argued that the philosophical concept of knowledge is not pertinent to KD, since there is nothing concrete philosophy can offer that would aid the process of building computational models from data. It is indeed true that the discussion of knowledge as a theoretical concept is, for the most part, not immediately applicable in practice. The formulation of the concept as imperatives to be fulfilled by KD experts is a step in the right direction, but the formulations are abstract and offer, for example, no criteria for deciding whether a given imperative has been fulfilled to an acceptable degree.

To the argument that knowledge as an epistemological concept is not relevant at all, the response is that operating on data and models is not a departure from the realm of philosophy — on the contrary, it represents a decision to prefer certain philosophical ideas and theories over others. Specifically, empiricism is favored over rationalism in the question of where knowledge comes from, and pragmatism over other theories in the question of what is true, but these choices do not simply render the alternatives irrelevant. They should, instead, be seen as a compromise that makes the problem of KD solvable in practice but also sets bounds to what any given solution can achieve. To understand the nature of the compromise, it is necessary to understand that knowledge as defined in KD is just one face of the entire complexity of the concept of knowledge.

The knowledge base of the T2MT system represents another attempt to derive a practical application from the theoretical discussion of knowledge: its KR scheme is based on a naive interpretation of the classical definition of knowledge. Naivete, in this context, is not simply a flaw but rather another compromise — it implies that the solution lacks the sophistication of a truly generic KB system, but on the other hand, its simplicity means that the concept is easy to grasp and relatively straightforward to implement. However, the reality of the knowledge system architecture falls somewhat short of the ideal, largely due to resourcing and scheduling considerations at the time of its development. Ideally, for instance, new service components should be pluggable in without disturbing other parts of the system, but in practice, such an addition always requires modifications to the session manager and external interface components.

Furthermore, the built-in KB solution of DBSA is not always the optimal one, and a better one probably could have been designed for the T2MT system, which currently represents the only test case of the architecture. The fact that there are no other test cases is another weakness, since it means that there may be elements in the design of

the architecture that are unsuitable for applications that differ significantly from the T2MT case. To expose and improve such elements, it would be important to apply the architecture to a wider variety of problems in order to see which parts of it are genuinely reusable and which ones should be revised. The T2MT system could thus be refined into a properly generic and versatile architecture for KB management.

The weakness of the welding database as an application contribution is that it is fairly narrow in scope, addressing only one application domain and only one layer of the proposed solution. The role of the database in the solution is an important one, but still, the most significant scientific advances were made in the area of algorithms for analyzing the data. The database as an independent contribution has some reuse value within the welding domain, but ultimately, it may be more valuable for the more generalized lessons that can be derived from it.

The strengths and weaknesses of the T2MT system as an application contribution are essentially opposite to those of the welding database. The system is a complete and integrated solution, designed with modifiability and reusability in mind, so the scope of the contribution is much wider in this case. The potential for reuse across diverse application domains is demonstrated by the application case studies, which address considerably different problems, even though the goal in both cases is to achieve high quality in handling a welding tool.

What the T2MT lacks is a degree of maturity comparable to the welding database or the model maintenance system, the two most extensively tested and validated application contributions of the thesis. The case-study applications were developed to a proof-of-concept level, but tests with realistic quantities of data in a realistic factory environment are still missing, leaving open, for instance, the question of whether the KB search algorithm performs adequately under real-world circumstances. The architecture and working principles of the system therefore outweigh the impact of its concrete implementation as it currently stands.

If one ignores the domain- and technology-specific aspects of the T2MT system and the welding database and focuses on those aspects that have the highest reuse potential, combining them comes close to spanning the entire hierarchy of data from top to bottom. Thus, the best way to use these results in the service of the KD process would be to follow this lead and build a database system that integrates all states of data under a single coherent set of interfaces, reusing the generic aspects of the two systems as appropriate. The system could take its deployment model from the T2MT system, with generic components providing hooks for application-specific functionality implemented

via dynamic scripting, enabling the reuse of not just the architecture but the majority of the code as well. Such a system could serve as a reference implementation of the proposed data model, the current absence of which diminishes the utility of the model considerably.

7.2.2 Workflow model

The major problem with the nonlinear, graph-structured workflow model is that it does not intuitively translate into a coherent plan for executing the process. The most acute shortcoming is that although the augmented process model covers tasks that are not included in the established model, it contributes little towards an understanding of how to perform these tasks effectively. Before the augmented model can be made into a practical and comprehensive tool for KD project management, it would have to be synthesized with relevant results from a diverse set of related disciplines. Section 7.3 includes a brief discussion of this topic.

The workflow model is, for the most part, an enumeration of actors and interactions, which on its own is not a sufficient reference for steering the KD process; what is needed is a sequential process flowchart that can be superimposed on the workflow model in order to make the process manageable in practice. Merging the workflow model with the CRISP-DM model represents a step in this direction, but a tentative and simplistic one. The overall effect is that the process is still managed using the established process model, with the augmented model mainly serving to draw attention to activities or issues that might otherwise be taken for granted or ignored.

Generally speaking, it of course makes sense to build upon existing work rather than completely discard something that has established itself as a standard solution. That said, it must be remarked that the increment made by the proposed workflow model is fairly small and straightforward. What the thesis does not address is the question of what would be the outcome of a bolder reinvention of the KD process model, resulting in something analogous to what has replaced the waterfall model in modern software engineering.

The completeness of the proposed process model is another question to which the thesis cannot give a definitive answer. The model has been pieced together from a variety of elements developed separately, and there is no case study available that would demonstrate a complete KD process featuring all the different elements. The case studies reported in Chapter 6 address several elements of the data and architectural

models, but the workflow model is left without significant empirical support. As a result, there is no data available that would provide conclusive answers to such questions as whether the workflow model is exhaustive, i.e., whether it covers all the actors and interactions that have a significant effect on the progress of a KD project.

7.2.3 Architectural model

A notable feature that the SA and DBSA frameworks currently lack is the graphical user interface that several other architecturally similar frameworks provide. Both offer a run-time console interface that allows users to send commands to the framework and to individual components, but there is no tool for editing applications configurations in graphic form, and since the frameworks also do not come with a comprehensive algorithm library, creating an application is likely to require an algorithm programming effort as well. This diminishes the suitability of SA and DBSA for the rapid, exploratory style of development encouraged by the other frameworks.

The best way to use SA or DBSA to support the KD process, then, would be to adopt a more conventional development style that clearly differentiates between the roles of developers and end users. Developers interact with the framework at the source code level, tailoring the application logic and the user interface according to the specific requirements of the problem at hand. End users, on the other hand, interact with the framework exclusively through the interface defined by the developers, and need not be aware of the framework at all; the framework exists mainly to help the developers implement those aspects of KD application logic that are always or frequently required, such as controlling data flows and communicating with databases.

A natural follow-up question to this is whether there is a rationale for preferring SA or DBSA over other frameworks for this style of development. Such decisions are determined in part by factors that have little if anything to do with the properties of the frameworks themselves, such as the strength of the support available from the vendor and the associated developer community. Currently, neither SA nor DBSA are being actively developed and distributed, a weakness that for many potential adopters would probably outweigh their strengths.

With respect to the general quality requirements, the main problem with the frameworks is the uncertainty concerning their performance when processing large datasets. Particularly with KD problems for which distributed computing is the only feasible solution, there is much that the existing case studies leave unclear. Although

both frameworks do provide interfaces that demonstrably can be used to distribute the workload of an application among multiple hosts, neither framework is specifically designed for this purpose. A probable implication of this is that compared to grid-based platforms, there are higher processing overheads and a greater need for manual labor in setting up the system.

Ultimately, for the purposes of this thesis, the SA and DBSA frameworks are less important than the architectural model underlying them. Clearly, neither framework implements the model in its entirety, and it is uncertain whether the development of either one will be picked up again at some point. The frameworks are therefore academically more significant for the novel architectural concepts they embody than for their actual code. A complete reference implementation of the architectural model would combine these concepts and the architectural foundations of the frameworks with a state-of-the-art GUI and other features needed in order to satisfy the remaining architectural requirements.

7.3 Untouched topics

It was already established in Chapter 2 that the study of knowledge discovery is spread across a considerable variety of disciplines within the field of computer science and engineering. Databases and knowledge bases, pattern recognition and machine learning, algorithm theory, software engineering, and information systems science are among the most important areas of knowledge required to build successful KD solutions. Between them, these computing disciplines span the technology hierarchy of Subsection 4.2.1 from top to bottom, but to look at technology only is to overlook two thirds of the interaction model of the KD process. Human involvement in the process is an essential part of the model, and this part cannot be adequately understood without the help of disciplines unrelated to computing.

Interactions between humans and technology are addressed by disciplines such as human-computer interaction (HCI) and usability research, which are not in themselves unrelated to computer science. They do, however, require the help of such disciplines — ergonomics and certain branches of psychology are obvious examples, since HCI must work within bounds established by the mental and physical capabilities of humans. Especially important are those areas of study that help us visualize data in an intuitive manner, since the goal of the KD process is essentially to help humans understand a

body of data. One could add even aesthetics to the list of disciplines that may be of some service in the pursuit of this goal.

The importance of social and behavioral sciences in the knowledge discovery process is further emphasized when we consider interactions that take place between human actors. Managing a KD project efficiently — finding the necessary actors and assigning roles to them, balancing the expectations of stakeholders, etc. — requires skills of organization, negotiation and persuasion, and these skills are based on knowledge of psychology, organizational science, business administration, and possibly others. Furthermore, as we saw in Subsection 4.2.2, knowledge of ethics is essential because negotiation and persuasion should never result in violation of a stakeholder's rights.

Besides ethics and aesthetics, there are other branches of philosophy that are relevant to knowledge discovery, albeit in a manner that pertains to knowledge acquisition in general rather than to the practice of knowledge discovery in particular. Epistemology explores the question of what it means to know something, while logic and philosophy of science are concerned with different methods of deriving knowledge, and all of these are therefore part of the essential theoretical background to technology designed to discover new knowledge. The remainder of this section looks at areas of philosophical inquiry that could provide additional insights but had to be left outside the scope of the thesis.

There is a pair of important epistemological concepts that have not been brought up yet, except in passing in the previous section, namely empiricism and rationalism. These are the main schools of thought concerning the ultimate source of human knowledge: empiricism claims that all knowledge is derived from observation and experience, whereas rationalism holds that proper knowledge is only achieved through logic and reasoning. In knowledge discovery through data mining, the acquisition of knowledge is based on inductive reasoning from a set of observations, as opposed to deductive reasoning from a set of axiomatic truths. DM thus represents an empirical approach to KD. This opens up an interesting question: is there also a rationalist approach to KD?

There are some examples of techniques and paradigms that may qualify, depending on which definition of KD one subscribes to. There are, for instance, software systems designed for proving mathematical theorems automatically or semi-automatically (Sutcliffe & Suttner 2001), arguably a form of rationalist KD in the domain of mathematics. Logic programming, as embodied, e.g., in the programming language Prolog (Warren *et al.* 1977), is another candidate, and is in fact used in the implementation of automated theorem provers (e.g., (Manthey & Bry 1988)). A more thorough investigation in this

area would be of some philosophical interest, but for this thesis, the question is out of scope.

As a final note on the philosophy of KD, let us consider the relationship between KD and philosophy of science. Whereas logic studies the rules of acquiring knowledge by rational means, philosophy of science is concerned with the principles of empirical knowledge acquisition. The most important result to come out of this study is the modern scientific method, which, simplified somewhat, is a cycle that begins with the formulation of a hypothesis intended to explain an observed phenomenon. This is followed by the design and execution of an experiment to test the hypothesis. If the outcome of the experiment supports the hypothesis, additional experiments are devised to rule out alternative explanations for the positive result. The more alternatives are successfully rejected in this way, the more likely it is that the proposed explanation is correct. If, on the other hand, an experiment fails to support the hypothesis, it must be revised or rejected and the cycle starts over.

Over time, as this approach is systematically applied, verified hypotheses accumulate and are eventually synthesized to form theories. The most important test of a scientific theory is its predictive power: it will survive only if it can successfully predict the results of future experiments. This represents a parallel between the scientific method and the KD process — in both, models of reality are induced from observations and evaluated by their accuracy in predicting further observations of the same subject. On a more fundamental level, science and KD share the pragmatic definition of truth; in science, it is codified in the concept of falsifiability (Popper 1959), which is an important criterion that any explanation must satisfy in order to be considered scientific. A falsifiable explanation is one that can conceivably be refuted experimentally, and the ability of an explanation to withstand falsification attacks from multiple angles can be viewed as a measure of its scientific strength.

A crucial element where the scientific method departs from its similarity to the KD process is the submission of results to peer review. The concept of peer review is based on the idea that it is not enough that scientists test their hypotheses: to be sure that the results are reliable, the tests by which they were attained should also be evaluated. The importance assigned to the peer review process is evident in the culture of science: papers are not considered properly scientific unless they have passed peer review, and journals and conferences are not considered properly scientific unless every paper submitted to them is subjected to peer review. The quality of papers published affects both the career advancement of a researcher and the reputation of a scientific publisher,

so mutual self-interest discourages both parties from trying to subvert the peer review system. It is interesting to note here the similarity to something we observed when discussing the impact of ethical behavior on the KD process: the ideal is reinforced by an environmental pressure that makes following the ideal a rational choice.

More important, however, is the lesson that for any given result, one should evaluate both the quality of the result and the soundness of the method by which the result was acquired. The absence of either evaluation should be taken as an indicator that the verity of the result is in doubt. From the perspective of the KD process, the implication of this is that testing the method of inquiry should be an integral part of the process, just as testing the result is. Testing the soundness of the method satisfies the requirement laid down in Subsection 4.1.1 that a piece of knowledge acquired using KD should be supported by a clear and comprehensive argument. The task of constructing and evaluating such an argument would make a significant research topic in its own right, but it is one that cannot be covered in this work beyond noting its place in the KD process.

8 Conclusion

Over the course of this thesis, a number of apparently divergent and only superficially related topics have been discussed: the concepts of data and knowledge, the processes of data mining and knowledge discovery, software frameworks, databases, knowledge bases, and computer ethics. However, all these topics are interlinked through the idea of a computer capable of possessing and generating knowledge. The idea is philosophically challenging, much like the idea of a computer capable of thinking is, but that is not to say that creating such a computer is an insurmountable problem. We can use the tools of philosophy to attack it, to determine what exactly we mean by the concept of knowledge and how we can transfer that concept to the realm of computing. A substantial portion of the thesis was devoted to applying major philosophical ideas concerning knowledge to computing and exploring their ramifications to the branch of computing known as knowledge discovery.

The individual results presented in the thesis address various aspects of KD, including the development of KD software and the management and representation of data and knowledge. However, the aspect that the thesis especially seeks to transform is the process of KD. There is an established model for the process that depicts it as a waterfall model-style progress from one step to the next, but this depiction, the thesis argues, is too simplistic a representation of the tasks and events that comprise the KD process in reality.

It was proposed that instead of a linear sequence of steps, the KD process should be modeled as a network of interactions involving human stakeholders and technological artifacts. The role of human actors is particularly emphasized — another departure from the established model, which is mainly concerned with the source data and the transformations applied to it in order to extract the hidden knowledge. The proposed new process model fully recognizes the importance of the latter, but at the same time, it points out that the technology for handling the transformations is relatively mature. Interactions involving human stakeholders are poorly understood in comparison, so the thesis advocates a reallocation of research priorities towards problems associated with them.

An inevitable consequence of the proposed modifications to the KD process model is a significant increase in complexity, which limits the usefulness of the model as a

practical tool for project management. However, to some extent, the added complexity is held in check by the partitioning of the augmented process model into three sub-models. These are the data model, the workflow model, and the architectural model.

The data model addresses the management of data and knowledge in the KD process. The model recapitulates the established process model by representing it in terms of the states and transformations that data goes through as it is refined into knowledge. On top of this, the model sets up an abstract architecture for a data repository that supports the execution of the KD process. An analysis of the concept of knowledge is carried out to establish a set of principles concerning the affirmation of the outcome of the process as true knowledge. Finally, the analysis and the abstract repository architecture are extended to cover knowledge representation.

The workflow model deals with the process activities required to execute the data transformations depicted by the data model. First, the model defines two sets of actors: technological artifacts and human stakeholders. Following this, the model defines three sets of interactions among the actors: interactions between human stakeholders, interactions between technological artifacts, and interactions between humans and technology. Interactions that may raise ethical issues, such as those involving the acquisition and use of personal information, are treated as a special case. The established KD process model is harnessed by using the standard CRISP-DM model to overlay a sequential process onto the actors and interactions of the workflow model.

The architectural model is concerned with the development of software to support the KD process. The model is manifested in two object-oriented software frameworks designed specifically with KD applications in mind. Both frameworks are based on the same architectural style, namely the pipes-and-filters style commonly found in KD frameworks. However, different objectives were emphasized in their development, and as a result, different design and implementation decisions were made. The architectural model describes how various aspects of the KD process are addressed by these decisions.

The three sub-models of the augmented process model were evaluated against a set of requirements defined for them before the models themselves were presented. This was done first analytically, by means of a requirements trace where each requirement was matched with those elements of the corresponding model that address the requirement. Following this, four application case studies were examined for additional evidence on how the models measure up to the requirements. Two of these pertain mainly to the data model, addressing problems related to the management of data in the KD process in the context of industrial manufacturing, while the other two are mainly relevant to the

architectural model, being KD applications developed using one of the frameworks discussed in the presentation of the architectural model.

A problem with the case studies is that they cover only a limited portion of the entire augmented process model. Most notably, none of the case studies offer direct empirical support for the workflow model, leaving a significant part of the overall model resting on analytical argumentation alone. Another weakness is the high abstraction level of the sub-models combined with the relatively low maturity level of the reference implementations. The current implementations of the architectural model, for instance, lack some major features that a modern KD software framework would be expected to provide.

The strength of the frameworks, and the proposed models in general, is in the novel ideas they include rather than the current state of their implementation. The thesis merges the conventional wisdom represented by the established process model and the new perspectives offered by the three sub-models into an integrated model that governs the planning and execution of the KD process in a holistic manner. Although the model as presented in the thesis resembles a patchwork of components developed to varying levels of detail, it provides an overarching framework within which the individual components can be improved by future work.

While the thesis advocates a greater emphasis on the role of human actors in the KD process, its concrete contributions are primarily related to the technological aspect of the process. As for human issues, there is no accepted theory concerning how to proceed with them, and the thesis has only a few minor contributions to offer regarding them. These are mainly in the field of ethics and address the rights and duties of stakeholders in the KD process. Perhaps more important than any of these is the general observation that there are problems associated with the process that cannot be solved by technology alone — the collaboration of a variety of research disciplines, including ethics and other humanities, is required for success.

We can thus conclude that to adopt the proposed view of the KD process is not to make the process more complex but to acknowledge the complexity that is already there in the reality of KD. Tools to manage the complexity can be developed, but it requires a broader perspective on KD and the concept of knowledge in general than the conventional KD process model can offer. This thesis makes an effort towards opening up such a perspective, while also proposing concrete solutions to a number of technical problems associated with it.

References

Adderley R & Musgrove PB (2001) Data mining case study: Modeling the behavior of offenders who commit serious sexual assaults. Proc. Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 215–220.

Agrawal R, Imieliński T & Swami A (1993) Mining association rules between sets of items in large databases. SIGMOD Record 22(2): 207–216.

Agrawal R & Srikant R (1994) Fast algorithms for mining association rules in large databases. Proc. 20th International Conference on Very Large Data Bases, 487–499.

Aizerman MA, Braverman EM & Rozonoer LI (1964) The probability problem of pattern recognition learning and the method of potential functions. Automation and Remote Control 25: 1175–1190.

AlSairafi S, Emmanouil FS, Ghanem M, Giannadakis N, Guo Y, Kalaitzopoulos D, Osmond M, Rowe A, Syed J & Wendel P (2003) The design of Discovery Net: Towards open grid services for knowledge discovery. International Journal of High Performance Computing Applications 17(3): 297–315.

Anderson DP, Cobb J, Korpela E, Lebofsky M & Werthimer D (2002) SETI@home: An experiment in public-resource computing. Communications of the ACM 45(11): 56–61.

Antal P, Fannes G, Timmerman D, Moreau Y & de Moor B (2003) Bayesian applications of belief networks and multilayer perceptrons for ovarian tumor classification with rejection. Artificial Intelligence in Medicine 29(1–2): 39–60.

Apache Software Foundation (2012) Couchdb wiki — technical overview. URI: http://wiki.apache.org/couchdb/Technical%20Overview. Cited 2014/03/10.

Auster M (2005) Proposed draft technical report on C++ library extensions. Technical report, ISO/IEC. URI: http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2005/n1745.pdf. Cited 2014/03/10.

Bass L, Clements P & Kazman R (2003) Software Architecture in Practice. Addison-Wesley, Boston, MA, 2nd edition.

Bayes T (1763) An essay towards solving a problem in the doctrine of chances. Philosophical Transactions 53: 370–418.

Beberg AL, Ensign DL, Jayachandran G, Khaliq S & Pande VS (2009) Folding@home: Lessons from eight years of volunteer distributed computing. Proc. IEEE International Symposium on Parallel & Distributed Processing.

Bedau MA (2008) Artificial life. In: Floridi L (ed) The Blackwell Guide to the Philosophy of Computing and Information. Blackwell Publishing, Oxford, UK.

Bélanger F & Carter L (2009) The impact of the digital divide on e-government use. Communications of the ACM 52(4): 132–135.

Berglund A, Boag S, Chamberlin D, Fernández MF, Kay M, Robie J & Siméon J (2010) XML path language (XPath) 2.0 (second edition). Technical report, World Wide Web Consortium. URI: http://www.w3.org/TR/xpath20/. Cited 2014/03/10.

Berthold MR, Cebron N, Dill F, di Fatta G, Gabriel TR, Georg F, Meinl T, Ohl P, Sieb C & Wiswedel B (2006) KNIME: The Konstanz information miner. Proc. 4th Annual Industrial Simulation Conference, Workshop on Multi-Agent Systems and Simulation.

Bertino E, Fovino I & Provenza L (2005) A framework for evaluating privacy preserving data mining algorithms. Data Mining and Knowledge Discovery 11(2): 121–154.

Boag S, Chamberlin D, Fernández MF, Florescu D, Robie J & Siméon J (2010) XQuery 1.0: An XML query language (second edition). Technical report, World Wide Web Consortium. URI: http://www.w3.org/TR/xquery/. Cited 2014/03/10.

Boehm B (2006) A view of 20th and 21st century software engineering. Proc. 28th International Conference on Software Engineering, 12–29.

Bonchi F, Giannotti F, Mainetto G & Pedreschi D (1999) Using data mining techniques in fiscal fraud detection. Proc. First International Conference on Data Warehousing and Knowledge Discovery, 369–376.

Boser BE, Guyon IM & Vapnik VN (1992) A training algorithm for optimal margin classifiers. Proc. Fifth Annual Workshop on Computational Learning Theory, 144–152.

Boyd D & Hargittai E (2010) Facebook privacy settings: Who cares? First Monday 15(8): 13–20.

Brause R, Langsdorf T & Hepp M (1999) Neural data mining for credit card fraud detection. Proc. 11th International Conference on Tools with Artificial Intelligence, 103–106.

Broadie S (1993) Ethics with Aristotle. Oxford University Press, Cary, NC.

Brown DE & Oxford RB (2001) Data mining time series with applications to crime analysis. Proc. 2001 IEEE International Conference on Systems, Man and Cybernetics, 3: 1453–1458.

Brown JL, Ferner CS, Hudson TC, Stapleton AE, Vetter RJ, Carland T, Martin A, Martin J, Rawls A, Shipman WJ & Wood M (2005) GridNexus: A grid services scientific workflow system. International Journal of Computer & Information Science 6(2): 72–82.

Buschmann F, Meunier R, Rohnert H, Sommerlad P & Stal M (1996) Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, Chichester.

Bynum TW & Moor JH (eds) (1998) The Digital Phoenix: How Computers are Changing Philosophy. Blackwell Publishers, Oxford, UK.

Cannataro M, Congiusta A, Pugliese A, Talia D & Trunfio P (2004) Distributed data mining on grids: Services, tools, and applications. IEEE Transactions on Systems, Man, and Cybernetics—Part B: Cybernetics 34(6): 2451–2465.

Cattell RGG, Barry D, Berler M, Eastman J, Jordan D, Russell C, Schadow O, Stanienda T & Velez F (eds) (2000) The Object Data Standard: ODMG 3.0. Morgan Kaufmann, San Francisco.

CERN (2008) Worldwide LHC computing grid. URI: http://public.web.cern.ch/public/en/lhc/Computing-en.html. Cited 2014/03/10.

Chamberlin DD & Boyce RF (1974) SEQUEL: A structured English query language. Proc. 1974 ACM SIGFIDET Workshop on Data Description, Access and Control, 249–264.

Chandrasekaran B, Josephson JR & Benjamins VR (1999) What are ontologies, and why do we need them? IEEE Intelligent Systems and their Applications 14(1): 20–26.

Chen J, Fan Q & Xu B (2004) Research on the application of data-mining for quality analysis in petroleum refining industry. Proc. 5th World Congress on Intelligent Control and Automation, 4314–4318.

Chen PPS (1976) The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems 1(1): 9–36.

Chou PB, Grossman E, Gunopulos D & Kamesam P (2000) Identifying prospective customers. Proc. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 447–456.

Christensen C, Aina T & Stainforth D (2005) The challenge of volunteer computing with lengthy climate model simulations. Proc. First International Conference on e-Science and Grid Computing.

Clifton C, Kantarcioglu M & Vaidya J (2002) Defining privacy for data mining. Proc. National Science Foundation Workshop on Next Generation Data Mining, 126–133.

Clifton C & Marks D (1996) Security and privacy implications of data mining. Proc. 1996 ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery, 15–19.

Clifton C & Thuraisingham B (2001) Emerging standards for data mining. Computer Standards & Interfaces 23(3): 187–193.

Codd EF (1970) A relational model of data for large shared data banks. Communications of the ACM 13(6): 377–387.

Codd EF (1971) A data base sublanguage founded on the relational calculus. Proc. 1971 ACM SIGFIDET Workshop on Data Description, Access and Control, 35–68.

Collier K, Carey B, Grusy E, Marjaniemi C & Sautter D (1998) A perspective on data mining. Technical report, The Center for Data Insight, Nothern Arizona University.

Connolly T & Begg C (2005) Database Systems: A Practical Approach to Design, Implementation, and Management. Pearson Education, Harlow, England, 4th edition.

Cooley R, Mobasher B & Srivastava J (1997) Web mining: Information and pattern discovery on the World Wide Web. Proc. Ninth IEEE International Conference on Tools with Artificial Intelligence, 558–567.

Cooper S, Khatib F, Treuille A, Barbero J, Lee J, Beenen M, Leaver-Fay A, Baker D, Popović Z & Foldit players (2010) Predicting protein structures with a multiplayer online game. Nature 466(7307): 756–760.

Cser L, Korhonen AS, Gulyás J, Mäntylä P, Simula O, Reiss G & Ruha P (1999) Data mining and state monitoring in hot rolling. Proc. Second International Conference on Intelligent Processing and Manufacturing of Materials, 1: 529–536.

Data2Knowledge (2005) D2K Developer White Paper. URI: http://www.d2k.com/d2k/pdf/D2K_Developer_White_Paper.pdf. Cited 2014/03/10.

Davis E (1990) Representations of Commonsense Knowledge. Morgan Kaufmann, San Mateo, CA.

Dean J & Ghemawat S (2008) MapReduce: Simplified data processing on large clusters. Communications of the ACM 51(1): 107–113.

Dean J & Ghemawat S (2010) MapReduce: A flexible data processing tool. Communications of the ACM 53(1): 72–77.

DeBarr D & Eyler-Walker Z (2006) Closing the gap: Automated screening of tax returns to identify egregious tax shelters. SIGKDD Explorations 8(1): 11–16.

Debatin B, Lovejoy JP, Horn AK & Hughes BN (2009) Facebook and online privacy: Attitudes, behaviors, and unintended consequences. Journal of Computer-Mediated Communication 15(1): 83–108.

Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Patil S, Su MH, Vahi K & Livny M (2004) Pegasus: Mapping scientific workflows onto the grid. Proc. Second European AcrossGrids Conference (AxGrids 2004), 11–20.

Dempster AP, Laird NM & Rubin DB (1977) Maximum likelihood from incomplete data via the EM algorithm. Journal of the Royal Statistical Society B 39(1): 1–38.

Denning PJ (2005) Is computer science science? Communications of the ACM 48(4): 27–31.

des Rivières J & Wiegand J (2004) Eclipse: A platform for integrating development tools. IBM Systems Journal 43(2): 371–383.
Dietterich TG & Michalski RS (1985) Discovering patterns in sequences of events. Artificial Intelligence 25(2): 187–232.
Dimitriadou E, Barth M, Windischberger C, Hornik K & Moser E (2004) A quantitative comparison of functional MRI cluster analysis. Artificial Intelligence in Medicine 31(1): 57–71.
Domingues P, Sousa B & Silva LM (2007) Sabotage-tolerance and trust management in desktop grid computing. Future Generation Computer Systems 23(7): 904–912.
Engelmore R & Morgan T (eds) (1988) Blackboard Systems. Addison-Wesley, Wokingham, England.
Fan TF, Hu WC & Liau CJ (2001) Decision logics for knowledge representation in data mining. Proc. 25th Annual International Computer Software and Applications Conference (COMPSAC 2001), 626–631.
Fayad ME & Schmidt DC (1997) Object-oriented application frameworks. Communications of the ACM 40(10): 32–38.
Ferrucci D, Brown E, Chu-Carroll J, Fan J, Gondek D, Kalyanpur AA, Lally A, Murdock JW, Nyberg E, Prager J, Schlaefer N & Welty C (2010) Building Watson: An overview of the DeepQA project. AI Magazine 31(3): 59–79.
Finnish Ministry of Justice (1999) Personal data act (523/1999). URI: http://www.finlex.fi/en/laki/kaannokset/1999/en19990523.pdf. Cited 2014/03/10.
Fisher RA (1936) The use of multiple measurements in taxonomic problems. Annals of Eugenics 7: 179–188.
Fomichev A, Grinev M & Kuznetsov S (2006) Sedna: A native XML DBMS. Proc. 32nd International Conference on Current Trends in Theory and Practice of Computer Science, 272–281.
Foo CY & Koivisto EMI (2004) Defining grief play in MMORPGs: Player and developer perceptions. Proc. 2004 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology, 245–250.
Gamma E, Helm R, Johnson R & Vlissides J (1994) Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.
Gauss CF (1809) Theoria motus corporum coelestium in sectionibus conicis solem ambientum. Hamburg.
Genereseth MR & Fikes RE (1992) Knowledge interchange format version 3.0 reference manual. Technical report, Computer Science Department, Stanford University.
Gersten W, Wirth R & Arndt D (2000) Predictive modeling in automotive direct marketing: Tools, experiences and open issues. Proc. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 398–406.
Giloi WK (1997) Konrad Zuse's Plankalkül: The first high-level, "non von Neumann" programming language. IEEE Annals of the History of Computing 19(2): 17–24.
Goebel M & Gruenwald L (1999) A survey of data mining and knowledge discovery software tools. SIGKDD Explorations 1(1): 20–33.
Gotterbarn D, Miller K & Rogerson S (1999) Software engineering code of ethics is approved. Communications of the ACM 42(10): 102–107.
Grier DA (2005) When Computers Were Human. Princeton University Press.
Grossman RL, Hornick MF & Meyer G (2002) Data mining standards initiatives. Communications of the ACM 45(8): 59–61.

Grossman RL, Kamath C, Kegelmeyer P, Kumar V & Namburu RR (eds) (2001) Data Mining for Scientific and Engineering Applications, chapter 5. Kluwer.

Ha SH & Park SC (2006) Service quality improvement through business process management based on data mining. SIGKDD Explorations 8(1): 49–56.

Haapalainen E, Laurinen P, Junno H, Tuovinen L & Röning J (2005) Methods for classifying spot welding processes: A comparative study of performance. Proc. 18th International Conference on Industrial & Engineering Applications of Artificial Intelligence & Expert Systems, 412–421.

Haapalainen E, Laurinen P, Junno H, Tuovinen L & Röning J (2006) Feature selection for identification of spot welding processes. Proc. 3rd International Conference on Informatics in Control, Automation and Robotics, 40–45.

Haapalainen E, Laurinen P, Röning J & Kinnunen H (2008a) Estimation of exercise energy expenditure using a wrist-worn accelerometer: a linear mixed model approach with fixed-effect variable selection. Proc. Seventh International Conference on Machine Learning and Applications, 796–801.

Haapalainen E, Laurinen P, Siirtola P, Röning J, Kinnunen H & Jurvelin H (2008b) Exercise energy expenditure estimation based on acceleration data using the linear mixed model. Proc. IEEE International Conference on Information Reuse and Integration, 131–136.

Hall M, Frank E, Holmes G, Pfahringer B, Reutemann P & Witten IH (2009) The WEKA data mining software: An update. SIGKDD Explorations 11(1): 10–18.

Hartmanis J (1993) Some observations about the nature of computer science. Lecture Notes in Computer Science 761: 1–12.

Helbig N, Gil-García JR & Ferro E (2009) Understanding the complexity of electronic government: Implications from the digital divide literature. Government Information Quarterly 26(1): 89–97.

Hettinger EC (1989) Justifying intellectual property. Philosophy & Public Affairs 18(1): 31–52.

Heyer LJ, Kruglyak S & Yooseph S (1999) Exploring expression data: Identification and analysis of coexpressed genes. Genome Research 9(11): 1106–1115.

Hollinger RC (1991) Hackers: Computer heroes or electronic highwaymen? ACM SIGCAS Computers and Society 21(1): 6–17.

Houstis EN, Catlin AC, Rice JR, Verykios VS, Ramakrishnan N & Houstis CE (2000) PYTHIA-II: A knowledge/database system for managing performance data and recommending scientific software. ACM Transactions on Mathematical Software 26(2): 227–253.

Introna LD (1997) Privacy and the computer: Why we need privacy in the information society. Metaphilosophy 28(3): 259–275.

ISO (2007) ISO/IEC 24707:2007 - information technology - Common Logic (CL): a framework for a family of logic-based languages. Technical report, International Organization for Standardization.

Jacobson I, Griss M & Jonsson P (1997) Software Reuse: Architecture, Process and Organization for Business Success. ACM Press, New York.

Jensen D, Rattigan M & Blau H (2003) Information awareness: A prospective technical assessment. Proc. Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 378–387.

Jicheng W, Yuan H, Gangshan W & Fuyan Z (1999) Web mining: Knowledge discovery on the web. Proc. 1999 IEEE International Conference on Systems, Man, and Cybernetics, II/137–II/141.

Jones H & Soltren H (2005) Facebook: Threats to privacy. Social Science Research 1–76.

Junno H, Laurinen P, Haapalainen E, Tuovinen L & Röning J (2005) Resistance spot welding process identification using an extended KNN method. Proc. IEEE International Symposium on Industrial Electronics, I: 7–12.

Junno H, Laurinen P, Haapalainen E, Tuovinen L, Röning J, Zettel D, Sampaio D, Link N & Peschl M (2004a) Resistance spot welding process identification and initialization based on self-organizing maps. Proc. 1st International Conference on Informatics in Control, Automation and Robotics, 296–299.

Junno H, Laurinen P, Tuovinen L & Röning J (2004b) Studying the quality of resistance spot welding joints using self-organising maps. Proc. Fourth International ICSC Symposium on Engineering of Intelligent Systems.

Juutilainen I, Tuovinen L, Laurinen P, Koskimäki H & Röning J (2011) Semi-automatic maintenance of regression models: an application in the steel industry. Journal of Computing and Information Technology 19(3): 193–206.

Kallio MA, Tuimala JT, Hupponen T, Klemelä P, Gentile M, Scheinin I, Koski M, Käki J & Korpelainen EI (2011) Chipster: user-friendly analysis software for microarray and other high-throughput data. BMC Genomics 12(507).

Kanellopoulos Y, Dimopulos T, Tjortjis C & Makris C (2006) Mining source code elements for comprehending object-oriented systems and evaluating their maintainability. SIGKDD Explorations 8(1): 33–40.

Kätevä J, Laurinen P, Rautio T, Suutala J, Tuovinen L & Röning J (2010) DBSA — a device-based software architecture for data mining. Proc. 2010 ACM Symposium on Applied Computing (SAC 2010), 2273–2280.

Kifer M & Lausen G (1989) F-Logic: A higher-order language for reasoning about objects, inheritance, and scheme. Proc. 1989 ACM SIGMOD International Conference on Management of Data (SIGMOD '89), 134–146.

Kim W, Agrawal R, Faloutsos C, Fayyad U, Han J, Piatetsky-Shapiro G, Pregibon D & Uthurusamy R (2003) "Data mining" is NOT against civil liberties. URI: http://www.sigkdd.org/civil-liberties.pdf. Cited 2012/09/10.

King RD, Karwath A, Clare A & Dephaspe L (2000) Genome scale prediction of protein functional class from sequence using data mining. Proc. Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 384–389.

Klyne G & Carroll JJ (2004) Resource Description Framework (RDF): Concepts and abstract syntax. Technical report, World Wide Web Consortium. URI: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/. Cited 2014/03/10.

Koskimäki H, Laurinen P, Haapalainen E, Tuovinen L & Röning J (2007) Application of the extended KNN method to resistance spot welding process identification and the benefits of process information. IEEE Transactions on Industrial Electronics 54(5): 2823–2830.

Ku Y & Gupta S (2008) Online gaming perpetrators model. In: Intelligence and Security Informatics, volume 5075 of *Lecture Notes in Computer Science*, 428–433. Springer.

Kusiak A & Kurasek C (2001) Data mining of printed-circuit board defects. IEEE Transactions on Robotics and Automation 17(2): 191–196.

Laurinen P, Junno H, Tuovinen L & Röning J (2004a) Studying the quality of resistance spot welding joints using Bayesian networks. Proc. Artificial Intelligence and Applications, 705–711.

Laurinen P, Tuovinen L, Haapalainen E, Junno H, Röning J & Zettel D (2004b) Managing and implementing the data mining process using a truly stepwise approach. Proc. Sixth International Baltic Conference on Databases & Information Systems, 246–257.

Laurinen P, Tuovinen L & Röning J (2005) Smart Archive: a component-based data mining application framework. Proc. Fifth International Conference on Intelligent Systems Design and Applications (ISDA 2005), 20–25.

Legendre AM (1806) Nouvelles méthodes pour la détermination des orbites des comètes. Paris.

Lenat DB & Guha RV (1991) The evolution of CycL, the Cyc representation language. ACM SIGART Bulletin 2(3): 84–87.

Lenat DB, Prakash M & Shepherd M (1985) CYC: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. AI Magazine 6(4): 65–85.

Li L, Tang H, Wu Z, Gong J, Gruidl M, Zou J, Tockman M & Clark RA (2004) Data mining techniques for cancer detection using serum proteomic profiling. Artificial Intelligence in Medicine 32(2): 71–83.

Lloyd SP (1982) Least squares quantization in PCM. IEEE Transactions on Information Theory 28(2): 129–137.

Lopes DM (2008) Digital art. In: Floridi L (ed) The Blackwell Guide to the Philosophy of Computing and Information. Blackwell Publishing, Oxford, UK.

Loui MC (1995) Computer science is a new engineering discipline. ACM Computing Surveys 27(1).

Ludäscher B, Altintas I, Berkley C, Higgins D, Jaeger E, Jones M, Lee EA, Tao J & Zhao Y (2006) Scientific workflow management and the Kepler system. Concurrency and Computation: Practice and Experience 18(10): 1039–1065.

Lukas L, Devos A, Suykens JAK, Vanhamme L, Howe FA, Majós C, Moreno-Torres A, van der Graaf M, Tate AR, Arús C & van Huffel S (2004) Brain tumor classification based on long echo proton MRS signals. Artificial Intelligence in Medicine 31(1): 73–89.

Manion M & Goodrum A (2000) Terrorism or civil disobedience: Toward a hacktivist ethic. ACM SIGCAS Computers and Society 30(2): 14–19.

Mannila H, Toivonen H & Verkamo AI (1994) Efficient algorithms for discovering association rules. Proc. AAAI Workshop on Knowledge Discovery in Databases, 181–192.

Manthey R & Bry F (1988) SATCHMO: A theorem prover implemented in Prolog. Proc. 9th International Conference on Automated Education, 415–434.

Maron ME (1961) Automatic indexing: An experimental inquiry. Journal of the ACM 8(3): 404–417.

Melli G, Zaïane OR & Kitts B (2006) Introduction to the special issue on successful real-world data mining applications. SIGKDD Explorations 8(1): 1–2.

Mierswa I, Wurst M, Klinkenberg R, Scholz M & Euler T (2006) YALE: Rapid prototyping for complex data mining tasks. Proc. 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 935–940.

Mulligan DK & Perzanowski A (2007) The magnificence of the disaster: Reconstructing the Sony BMG rootkit incident. Berkeley Technology Law Journal 22: 1157–1232.

Oinn T, Addis M, Ferris J, Marvin D, Senger M, Greenwood M, Carver T, Glover K, Pocock MR, Wipat A & Li P (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17): 3045–3054.

Park JS, Chen MS & Yu PS (1995) An effective hash-based algorithm for mining association rules. Proc. 1995 ACM SIGMOD International Conference on Management of Data, 175–186.

Pinheiro CAR, Evsukoff AG & Ebecken NFF (2006) Revenue recovering with insolvency prevention on a Brazilian telecom operator. SIGKDD Explorations 8(1): 65–70.

Popper K (1959) The Logic of Scientific Discovery. Hutchinson, London.

Raddick MJ, Bracey G, Gay PL, Lintott CJ, Murray P, Schawinski K, Szalay AS & Vandenberg J (2010) Galaxy Zoo: Exploring the motivations of citizen science volunteers. Astronomy Education Review 9(1).

Rautio T, Laurinen P & Röning J (2009) Component-based framework for mobile data mining with support for real-time sensors. Proc. International Conference on Agents and Artificial Intelligence, 208–213.

Roddick JF & Spiliopoulou M (1999) A bibliography of temporal, spatial and spatio-temporal data mining research. SIGKDD Explorations 1(1): 34–38.

Roddick JF & Spiliopoulou M (2002) A survey of temporal knowledge discovery paradigms and methods. IEEE Transactions on Knowledge and Data Engineering 14(4): 750–767.

Rodríguez A, Mencke M, Alor-Hernandez G, Posada-Gomez R, Gomez JM & Aguilar-Lasserre AA (2009) MEDBOLI: Medical diagnosis based on ontologies and logical inference. Proc. International Conference on eHealth, Telemedicine, and Social Medicine (eTELEMED '09), 233–238.

Rosenblatt F (1958) The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65(6): 386–408.

Rumelhart DE, Hinton GE & Williams RJ (1986) Learning Internal Representations by Error Propagation. MIT Press.

Sarmenta LFG (2002) Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems 18(4): 561–572.

Schafer JB, Konstan JA & Riedl J (2001) E-commerce recommendation applications. Data Mining and Knowledge Discovery 5(1): 115–153.

Schollmeier R (2001) A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Proc. First International Conference on Peer-to-Peer Computing, 101–102.

Setiabudi DH, Budhi GS, Purnama IWJ & Noertjahyana A (2011) Data mining market basket analysis using hybrid-dimension association rules, case study in Minimarket X. Proc. 2011 International Conference on Uncertainty Reasoning and Knowledge Engineering (URKE), 196–199.

Shannon CE (1940) A symbolic analysis of relay and switching circuits. Master's thesis, Massachusetts Institute of Technology.

Shao H, Zhao H & Chang GR (2002) Applying data mining to detect fraud behavior in customs declaration. Proc. First International Conference on Machine Learning and Cybernetics, 1241–1244.

Shearer C (2000) The CRISP-DM model: The new blueprint for data mining. Journal of Data Warehousing 5(4): 13–22.

Shen W, Hao Q, Yoon HJ & Norrie DH (2006) Applications of agent-based systems in intelligent manufacturing: An updated review. Advanced Engineering Informatics 20(4): 415–431.

Siirtola P, Laurinen P, Haapalainen E, Röning J & Kinnunen H (2009) Clustering-based activity classification with a wrist-worn accelerometer using basic features. Proc. 2009 IEEE Symposium on Computational Intelligence in Data Mining, 95–100.

Singh P, Thomas AC & Sepulveda A (2006) Market basket recommendations for the HP SMB store. SIGKDD Explorations 8(1): 57–64.

Solove DJ (2008) Data mining and the security-liberty debate. The University of Chicago Law Review 75(1): 343–362.
Spafford EH (1992) Are computer hacker break-ins ethical? Journal of Systems and Software 17(1): 41–47.
Sutcliffe G & Suttner C (2001) Evaluating general purpose automated theorem proving systems. Artificial Intelligence 131(1–2): 39–54.
Tan KC, Yu Q, Heng CM & Lee TH (2003) Evolutionary computing for knowledge discovery in medical diagnosis. Artificial Intelligence in Medicine 27(2): 129–154.
Tavani HT (1999) Informational privacy, data mining, and the Internet. Ethics and Information Technology 1(2): 137–145.
Taylor I, Shields M, Wang I & Philp R (2003) Distributed P2P computing within Triana: A galaxy visualization test case. Proc. International Parallel and Distributed Processing Symposium (IPDPS '03).
Taylor RC (2010) An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics. BMC Bioinformatics 11(Suppl 12).
Thuraisingham B (2002) Data mining, national security, privacy and civil liberties. ACM SIGKDD Explorations Newsletter 4(2): 1–5.
Thuraisingham B, Clifton C, Maurer J & Ceruti MG (2001) Real-time data mining of multimedia objects. Proc. 4th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, 360–365.
Tipping ME (2000) The relevance vector machine. Proc. Advances in Neural Information Processing Systems 12, 652–658.
Tuovinen L, Laurinen P, Juutilainen I & Röning J (2008) Data mining applications for diverse industrial application domains with Smart Archive. Proc. IASTED International Conference on Software Engineering (SE 2008), 56–61.
Tuovinen L, Laurinen P, Koskimäki H, Haapalainen E & Röning J (2007) Building a database to support intelligent computational quality assurance of resistance spot welding joints. Proc. 2007 IEEE International Symposium on Industrial Electronics (ISIE 2007), 1980–1985.
Tuovinen L, Laurinen P & Röning J (2009a) Data type management in a data mining application framework. Proc. First International Conference on Agents and Artificial Intelligence (ICAART 2009), 333–338.
Tuovinen L, Laurinen P & Röning J (2009b) External tool integration with proxy filters in a data mining application framework. Proc. 4th International Conference on Software and Data Technologies (ICSOFT 2009), 249–255.
Tuovinen L & Röning J (2005) Balance of power: the social-ethical aspect of data mining. Proc. Sixth International Conference of Computer Ethics: Philosophical Inquiry (CEPE 2005), 367–379.
Tuovinen L & Röning J (2007) Baits and beatings: vigilante justice in virtual communities. Proc. 7th International Conference of Computer Ethics: Philosophical Enquiry (CEPE 2007), 397–405.
Tuovinen L & Röning J (2009) Everybody wins: challenges and promises of knowledge discovery through volunteer computing. Proc. 8th International Conference of Computer Ethics: Philosophical Enquiry (CEPE 2009), 821–842.
Tuovinen L, Talus T, Koponen E, Laurinen P & Röning J (2010) A task-to-method transformation system for self-configuration in manufacturing. Proc. 8th IEEE International Conference on Industrial Informatics (INDIN 2010), 245–252.

Turing AM (1937) On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathemathical Society s2-42(1): 230–265.

Turing AM (1950) Computing machinery and intelligence. Mind LIX(236): 433–460.

Underson LFG (2002) Using data mining and judgment analysis to construct a predictive model of crime. Proc. 2002 IEEE International Conference on Systems, Man and Cybernetics, 7.

Väinämö K, Laurinen P & Röning J (2000) Smart Archive for on-line learning systems. Proc. 3rd International Symposium on Tool Environments and Development Methods for Intelligent Systems, 152–169.

Velásquez JD & Palade V (2007) A knowledge base for the maintenance of knowledge extracted from web data. Knowledge-Based Systems 20(3): 238–248.

Verykios VS, Bertino E, Fovino IN, Provenza LP, Saygin Y & Theodoridis Y (2004) State-of-the-art in privacy preserving data mining. SIGMOD Record 33(1): 50–57.

Viaene S, Derrig RA & Dedene G (2004) A case study of applying boosting naïve bayes to claim fraud diagnosis. IEEE Transactions on Knowledge and Data Engineering 16(5): 612–620.

von Neumann J (1945) First draft of a report on the EDVAC. Technical report, University of Pennsylvania.

Včelák P, Klečková J & Rohan V (2010) Cerebrovascular diseases research database. Proc. 3rd International Conference on Biomedical Engineering and Informatics (BMEI 2010), 2687–2691.

W3C OWL Working Group (2009) OWL 2 Web Ontology Language document overview. Technical report, World Wide Web Consortium. URI: http://www.w3.org/TR/2009/REC-owl2-overview-20091027/. Cited 2014/03/10.

Wahlstrom K & Roddick JF (2000) On the impact of knowledge discovery and data mining. Proc. Selected Papers from the Second Australian Institute Conference on Computer Ethics, 1: 22–27.

Walton CD & Barker AD (2004) An agent-based e-science experiment builder. Proc. 1st International Workshop on Semantic Intelligent Middleware for the Web and the Grid.

Wang K, Wang L, Yuan Q, Luo S, Yao J, Yuan S, Zheng C & Brandt J (2001) Construction of a generic reaction knowledge base by reaction data mining. Journal of Molecular Graphics and Modelling 19(5): 427–433.

Warren DHD, Pereira LM & Pereira F (1977) Prolog - the language and its implementation compared with Lisp. Proc. 1977 Symposium on Artificial Intelligence and Programming Languages, 109–115.

Woolridge M & Jennings NR (1995) Intelligent agents: theory and practice. The Knowledge Engineering Review 10(2): 115–152.

Wyns B, Sette S, Boullart L, Baeten D, Hoffman IEA & de Keyser F (2004) Prediction of diagnosis in patients with early arthritis using a combined Kohonen mapping and instance-based evaluation criterion. Artificial Intelligence in Medicine 31(1): 45–55.

Zaffalon M, Wesnes K & Petrini O (2003) Reliable diagnoses of dementia by the naive credal classifier inferred from incomplete cognitive data. Artificial Intelligence in Medicine 29(1–2): 61–79.

Zaïane OR, Han J, Li ZN & Hou J (1998) Mining multimedia data. Proc. 1998 Conference of the Centre for Advanced Studies on Collaborative Research, 24.

Zhang CH, Di L & An Z (2003a) Welding quality monitoring and management system based on data mining technology. Proc. Second International Conference on Machine Learning and Cybernetics, 13–17.

Zhang N & Zhao W (2007) Privacy-preserving data mining systems. Computer 40(4): 52–58.
Zhang Z, Salerno JJ & Yu PS (2003b) Applying data mining in investigating money laundering crimes. Proc. Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 747–752.
Ziegler GM (2004) The great prime number record races. Notices of the AMS 51(4): 414–416.

481. Peschl, Michael (2014) An architecture for flexible manufacturing systems based on task-driven agents

482. Kangas, Jani (2014) Separation process modelling : highlighting the predictive capabilities of the models and the robustness of the solving strategies

483. Kemppainen, Kalle (2014) Towards simplified deinking systems : a study of the effects of ageing, pre-wetting and alternative pulping strategy on ink behaviour in pulping

484. Mäklin, Jani (2014) Electrical and thermal applications of carbon nanotube films

485. Niemistö, Johanna (2014) Towards sustainable and efficient biofuels production : use of pervaporation in product recovery and purification

486. Liu, Meirong (2014) Efficient super-peer-based coordinated service provision

487. Väyrynen, Eero (2014) Emotion recognition from speech using prosodic features

488. Celentano, Ulrico (2014) Dependable cognitive wireless networking : modelling and design

489. Peräntie, Jani (2014) Electric-field-induced dielectric and caloric effects in relaxor ferroelectrics

490. Aapaoja, Aki (2014) Enhancing value creation of construction projects through early stakeholder involvement and integration

491. Rossi, Pekka M. (2014) Integrated management of groundwater and dependent ecosystems in a Finnish esker

492. Sliz, Rafal (2014) Analysis of wetting and optical properties of materials developed for novel printed solar cells

493. Juntunen, Jouni (2014) Enhancing organizational ambidexterity of the Finnish Defence Forces' supply chain management

494. Hänninen, Kai (2014) Rapid productisation process : managing an unexpected product increment

495. Mehtonen, Saara (2014) The behavior of stabilized high-chromium ferritic stainless steels in hot deformation

496. Majava, Jukka (2014) Product development : drivers, stakeholders, and customer representation during early development

497. Myllylä, Teemu (2014) Multimodal biomedical measurement methods to study brain functions simultaneously with functional magnetic resonance imaging

Book orders:
Granum: Virtual book store
http://granum.uta.fi/granum/

UNIVERSITY OF OULU P.O. Box 8000 FI-90014 UNIVERSITY OF OULU FINLAND

ACTA UNIVERSITATIS OULUENSIS

SERIES EDITORS

A **SCIENTIAE RERUM NATURALIUM**

Professor Esa Hohtola

B **HUMANIORA**

University Lecturer Santeri Palviainen

C **TECHNICA**

Postdoctoral research fellow Sanna Taskila

D **MEDICA**

Professor Olli Vuolteenaho

E **SCIENTIAE RERUM SOCIALIUM**

University Lecturer Veli-Matti Ulvinen

F **SCRIPTA ACADEMICA**

Director Sinikka Eskelinen

G **OECONOMICA**

Professor Jari Juga

EDITOR IN CHIEF

Professor Olli Vuolteenaho

PUBLICATIONS EDITOR

Publications Editor Kirsti Nurkkala

ISBN 978-952-62-0523-6 (Paperback)
ISBN 978-952-62-0524-3 (PDF)
ISSN 0355-3213 (Print)
ISSN 1796-2226 (Online)

UNIVERSITY of OULU
OULUN YLIOPISTO