UNIVERSITÉ PARIS XIII – INSTITUT GALILÉE
LIPN – UMR CNRS 7030

№ attribué par la bibliothèque

T H È S E

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ PARIS XIII

Discipline : Informatique

présentée et soutenue publiquement

par

Moez ESSAIDI

le Mardi 02 Juillet 2013

Titre :

Model-Driven Data Warehouse And Its Automation Using Machine Learning Techniques

Directeurs de thèse :
Aomar OSMANI et Céline ROUVEIROL

JURY

Pr. Yann CHEVALEYRE,	Président
Pr. Jacky AKOKA,	Rapporteur
Pr. Céline ROUVEIROL,	Directrice de thèse
Pr. Nicolas PRAT,	Examinateur
Pr. Christel VRAIN,	Rapporteur
Pr. Aomar OSMANI,	Co-Directeur de thèse
Dr. Jean-Marc VANEL,	Examinateur

Abstract

This thesis aims at proposing an end-to-end approach which allows the automation of the process of model transformations for the development of data warehousing components. The main idea is to reduce as much as possible the intervention of human experts by using once again the traces of transformations produced on similar projects. The goal is to use supervised learning techniques to handle concept definitions with the same expressive level as manipulated data. The nature of the manipulated data leads us to choose relational languages for the description of examples and hypothesises. These languages have the advantage of being expressive by giving the possibility to express relationships between the manipulated objects, but they have the major disadvantage of not having algorithms allowing the application on large scales of industrial applications. To solve this problem, we have proposed an architecture that allows the perfect exploitation of the knowledge obtained from transformations' invariants between models and metamodels. This way of proceeding has highlighted the dependencies between the concepts to learn and has led us to propose a learning paradigm, called dependent-concept learning. Finally, this thesis presents various aspects that may influence the next generation of data warehousing platforms. The latter suggests, in particular, an architecture for business intelligence-as-a-service based on the most recent and promising industrial standards and technologies.

Keywords: *Model-Driven Data Warehouse, Dependent-Concept Learning, Inductive Logic Programming, Business Intelligence-as-a-Service.*

Résumé

L'objectif de ce travail de thèse est de proposer une approche permettant l'automatisation complète du processus de transformation de modèles pour le développement d'entrepôts de données. L'idée principale est de réduire au mieux l'intervention des experts humains en utilisant les traces de transformations réalisées sur des projets similaires. L'objectif est d'utiliser des techniques d'apprentissage supervisées pour traiter les définitions de concepts avec le même niveau d'expression que les données manipulées. La nature des données manipulées nous a conduits à choisir les langages relationnels pour la description des exemples et des hypothèses. Ces langages ont l'avantage d'être expressifs en donnant la possibilité d'exprimer les relations entres les objets manipulés mais présente l'inconvénient majeur de ne pas disposer d'algorithmes permettant le passage à l'échelle pour des applications industrielles. Pour résoudre ce problème, nous avons proposé une architecture permettant d'exploiter au mieux les connaissances issues des invariants de transformations entre modèles et métamodèles. Cette manière de procéder a mis en lumière des dépendances entre les concepts à apprendre et nous a conduits à proposer un paradigme d'apprentissage dit de concepts-dépendants. Enfin, cette thèse présente plusieurs aspects qui peuvent influencer la prochaine génération de plates-formes décisionnelles. Elle propose, en particulier, une architecture de déploiement pour la business intelligence en tant que service basée sur les normes industrielles et les technologies les plus récentes et les plus prometteuses.

Mots clés: *Entrepôts de Données Dirigés par les Modèles, Apprentissage de Concepts-Dépendants, Programmation Logique Inductive, Business Intelligence en tant que Service.*

Acknowledgements

The completion of this Ph.D thesis would not have been possible without the support and encouragement of several people. Hence, I would like to seize this opportunity to show my gratitude to those who have assisted me in a myriad of ways.

I would like to thank gratefully and sincerely my principal supervisor, Prof. Aomar Osmani for his guidance, understanding, patience, and most importantly, his friendship during my graduate studies. His constant encouragement, support and crucial contributions have made work successful.

I would additionally like to thank my second supervisor, Prof. Céline Rouveirol for his support, assistance, and guidance. His good advice, suggestions and help have been precious for both academic and a personal level, for which I am extremely grateful.

This research was financed in part by the ANRT (*National Association for Research and Technology*), Grant CIFRE-1341/2008, and *Intelligence Power Company*. I extend my sincere thanks to Clarisse Angelier, head of CIFRE department at the ANRT. My sincere acknowledgements also go to Tahar Arib, manager of Intelligence Power, which provided the financial and technical support necessary for this research.

I would also like to express my gratitude to the thesis committee members, Prof. Jacky Akoka, Prof. Christel Vrain, Prof. Yann Chevaleyre, Prof. Nicolas Prat and Dr. Jean-Marc Vanel, who gave me helpful advice that greatly improved this work.

Furthermore, I would like to thanks all of the members of the *Computer Science Lab of Paris-Nord (LIPN)*, and especially Dominique Bouthinon for providing an external opinion on several occasions. Also, I would like to recognize and thank

the team in Intelligence Power, Nicolas Krief, Dhafrallah Dahmane and François Bureau for all their help and support.

My gratitude also goes to my wife Olfa and my son Pasha Essaidi, for the personal support, love, great patience and dedication because together, as my family, they have been witnesses of my professional development and they have been the main actors in the implementation of my career.

My heartfelt thanks also to my parents, to whom I owe a great deal. To my late father Mustapha, thank you and rest in peace. To my mother Massaouda El-Kafi, she has been a constant source of encouragement and support. I would like to thank my brothers Montassar and Mohamed for their support, and also thank my sister Imen.

My sincere thanks to my friends Nabila Charifi, Farhat Hmida, Iheb Younes and Ali Mansour for their orientation as well as their permanent motivation and concern about my projects.

Finally, I would like to thank everybody who has been important for the success of this thesis and apologize for not mentioning personally each and everyone.

Contents

Abbreviations

BI	**B**usiness **I**ntelligence
BIaaS	**B**usiness-**I**ntelligence-**a**s-**a**-**S**ervice
CWM	**C**ommon **W**arehouse **M**etamodel
CWMX	**C**ommon **W**arehouse **M**etamodel e**X**tensions
DSS	**D**ecision **S**upport **S**ystems
DW	**D**ata **W**arehouse
ETL	**E**xtraction **T**ransformation **L**oading
ILP	**I**nductive **L**ogic **P**rogramming
ML	**M**achine **L**earning
MDA	**M**odel-**D**riven **A**rchitecture
MDDW	**M**odel-**D**riven **D**ata **W**arehouse
MDE	**M**odel-**D**riven **E**ngineering
MOF	**M**eta-**O**bject **F**acility
MTBD	**M**odel **T**ransformation **B**y-**D**emonstration
MTBE	**M**odel **T**ransformation **B**y-**E**xample
OCL	**O**bject **C**onstraint **L**anguage
ODM	**O**ntology **D**efinition **M**etamodel
OLAP	**O**n**L**ine **A**nalytical **P**rocessing
QVT	**Q**uery-**V**iew-**T**ransformation
SaaS	**S**oftware-**a**s-**a**-**S**ervice
2TUP	**T**wo **T**rack **U**nified **P**rocess
UML	**U**nified **M**odeling **L**anguage

in memory of my father, Mustapha ESSAIDI.

Chapter 1

Introduction

Decision support systems and *business intelligence systems* [1, 2] are the areas of the information systems discipline that is focused on supporting and improving decision-making across the enterprise. The decision-making process is a strategic asset that helps companies to differentiate themselves from competitors, improve service, and optimize performance results. The data warehouse [3, 4] is the central component of current decision support and business intelligence systems and is responsible for collecting and storing useful information to improve decision-making process in organization. The context of development of *decision support systems* is defined around several business, conceptual and technical constraints (figure 1.1). The data warehousing and business intelligence projects are still exposed to several risks and require more knowledge about the underlying business domain. The data warehousing architecture is defined through several heterogeneous and interrelated layers: data warehouse layers are mainly interrelated by design constraints and the data exchange flow (i.e., *Extraction, Transformation, and Loading - ETL*). Also, each layer contains different components and using different modelling profiles.

From the technical point-of-view, the urbanization of the decision support and data warehousing systems can be made using one or more technologies (for example Oracle as database, SAP Business Objects for reporting, etc.). From the conceptual point-of-view, actual data warehouse modelling approaches use several standard formalisms and profiles (e.g., UML and CWM). The *Unified Modelling Language (UML)* is a general-purpose modelling language for specifying, constructing, and documenting the artefacts of systems. The main purpose of the *Common Warehouse*

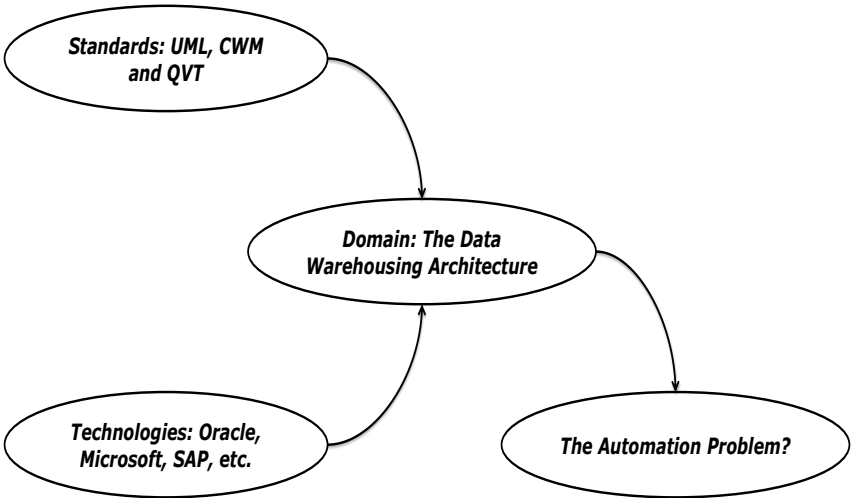

FIGURE 1.1: Context of Decision Support Systems Developpement.

Metamodel (CWM) is to enable easy interchange of warehousing and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments. This complexity of the *decision support systems* development process, makes the definition of a solution for an automatic generation of data warehousing components increasingly difficult.

Several data warehouse design frameworks and engineering processes have been proposed during the last few years. However, the *framework-oriented* approaches [5–9] fail to provide an integrated and a standard framework that is designed for all layers of the data warehousing architecture. The *process-oriented* approaches [10–14] fail, also, to define an engineering process that handles the whole development cycle of data warehouse with an iterative and incremental manner while considering both the business and the technical requirements. In addition, not much effort was devoted to unify the framework and the process into a single integrated approach. Moreover, no intelligent and automatic data warehouse engineering method is provided. However, in the current context, successful organizations need an infrastructure to automate processes that ensure development, consistency, changes, flexible deployment and manageability of their data warehousing environments. By implementing automated and agile solutions, organizations are able to codify best practices, reduce the burden on expensive engineering talent, and increase the effectiveness of the development and delivery teams. Agile business intelligence needs agile deployment architecture. Hence, *Software-as-a-Service (SaaS)* [15, 16]

is a new information delivery model for business intelligence and data warehousing. This architectural model has many advantages compared to the traditional deployment model and it can be a good choice for our solution.

This thesis deals with these issues by studying (i) the use of the *model-driven engineering* paradigm [17] for data warehouse components development, (ii) its automation using machine learning techniques and (iii) the recommended deployment architecture to reduce costs of data warehousing infrastructures. The proposed approach is based on the extension of the *model-driven* framework using inductive machine learning [18, 19] and it is inspired from existing methods [20–23] and recognized standards [24–26]. The proposed deployment architecture addresses *business intelligence-as-a-service* requirements and it is also based on promising architectural model, open standards and tools.

The *model-driven data warehouse* gathers approaches [27, 28] that align the development of the data warehouse with a general *model-driven engineering* paradigm [17]. The *model-driven engineering* is mainly based on models, meta-models and transformation design. Indeed, model- driven strategy encourages the use of models as a central element of development. The models are in accordance with metamodels and the transformation rules are applied to refine them. So, the *model-driven data warehouse* represents a first stage of automatic data warehouse components development and deployment. But, only are addressed the components of intermediate models (i.e., conceptual, logical, etc.): derivation and code generation are considered in the automation process. Because in reality, model transformation design is ensured by human experts and in general it generates an exorbitant cost.

Transformations are the central components of each model-driven process. However, transformation development is a very hard task that makes the model-driven approach more complex and entails additional costs. So, designers or programmers must have high skills in the corresponding metamodels and the transformation languages, e.g., the *Query-View-Transformation (QVT)* [29]. In addition, data warehousing projects require more knowledge about the underlying business domain and requirements. This raises many risks and challenges during the transformation design. One of the main challenges is to automatically learn these transformations from existing project traces. In this context, *model transformation by-example* (introduced by [30]) is an active research area in model-driven software engineering that uses artificial intelligence techniques and proposes to automatically derive

transformation rules. It provides assistance to designers in order to simplify the development of model transformations and it reduces complexity, costs and time of development. This process of transformation learning represents a second stage of an automatic approach for data warehousing components engineering. But actually, in a *model-driven data warehouse* context, several steps are needed to automatically learn the transformation rules.

The first step, which has been addressed as first part of this work, is a modelling of the global solution. It consists in isolating steps where it is necessary to induce transformation rules; in identifying the metamodels used to define the input/output models of these transformations and in designing a conceptual framework for transformation learning that uses adequate representation language. In this step, the *model-driven data warehouse* framework is extended by machine learning in order to support the expert in the transformation process. The architecture of *model-driven data warehouse* is a composite architecture that uses multiple components and presents several technical risks. Therefore, we focus on effective modelling of the *model-driven warehousing* architecture in order to understand the machine learning approach integration, how to make learning and how to deploy the application effectively. This modelling step, considered as modelling bias (or architecture bias) is important to manage these risks and make efficient the task of transformation learning.

The attribute-based approaches are limited to non-relational descriptions of objects. In fact, the learned descriptions do not specify relations among the objects' parts. The background knowledge is expressed in rather limited form and the concept description language is usually inappropriate for domains when relationships between objects must be designed. In the proposed framework, the learning process will use data-models (or data schemas) to set examples and background knowledge. So, the definition of relations between model elements (e.g., class, attribute, association), is required to set-up the learning process. Relational learning provides the appropriate approach to answer this problem. This framework provides several advantages, because the defined relational information plays an important role in the resulted transformation rules. Thus, we focus on providing an optimized representation of the language bias. The language bias (also known as declarative bias) aims to restrict the representation to clauses that best define the transformation rules. The *unified modelling language* and the *common warehouse metamodel* standards

metamodels are used to find relations and to define the representation language. As part of the modelling step, the task addresses the issues of translating a CWM-UML transformation problem into a relational learning problem. Indeed, scaling in the use of learning systems is a fundamental issue. This requires improving the design process of the overall architecture of *model-driven data warehouse* that allows thus an effective evaluation process.

We propose to express the model transformation problem as an *Inductive Logic Programming (ILP)* one [31]. The ILP is interested in learning logic programs, which allow to express complex relationships between models. The proposed framework uses existing project traces to define ILP inputs (i.e., background knowledge B and examples E) and to find the best transformation rules. The *inductive logic programming* method then, learns the underlying concepts and outputs as a *solution* a logic program to encode the learned concepts. ILP methods choose the solution from a set of well-formed *hypotheses*, known as the *hypothesis language, denoted* $\mathcal{L}_h$. In this thesis, we show optimized ways to specify hypothesis languages in the context of the studied application: *model-driven data warehouse* and its automation using *by-example* techniques.

The second step, which has been addressed as second part of this work, consists of defining an optimal approach to learn the transformations. In fact, within a *model-driven data warehouse* application, *dependencies* exist between transformations. We investigate a new machine learning methodology stemming from the application needs: *learning dependent-concepts*. We propose a *Dependent-Concept Learning (DCL)* approach where the objective is to build a pre-order set of concepts on this dependency relationship: first learn non-dependent concepts, then, at each step, add the learned concept as background knowledge for next concepts to be learned according to the pre-order. This DCL methodology is implemented and applied to our transformation learning problem. The experimental evaluation shows that the DCL system gives significantly better results compared to learning concepts independently.

The automation of data warehousing deployment and business intelligence delivery require implementing web-based services in order to meet market demands for speed and agility. Without a doubt, web-based business intelligence can be implemented very fast and it is easy to configure and to scale-up. In addition, the emergence

of *software-as-a-service* architecture has influenced current data warehousing infrastructures. Therefore, cloud-based business intelligence known as *business intelligence-as-a-service* is an active research field and is a topic worth noting. This field focuses on improving and optimising the deployment of data warehousing components and services. So, considering these recent aspects, the third part of our work is to propose a functional and a technical *business intelligence-as-a-service* architecture to deploy the designed solution. We introduce the *model-driven data warehouse-as-a-service* architecture and the promising open industry standards and technologies recommended for use. This architecture offers many advantages, such as lower *Total Cost of Ownership (TCO)* and better *Return On Investment (ROI)* and an accessible, scalable and quick deployment. This work of architecture design and implementation is important to improve the performance of the approach. But also, it is a much anticipated result in this work, conducted as part of a partnership between *Intelligence Power Company* (as industrial partner) and the *Computer Science Lab of Paris-Nord (LIPN)*.

The thesis starts with an introductory chapter clarifying the nature and the motivations of this work. Chapter 2 presents the terminologies used and outlines the concerned fields: the *data warehousing architecture*, the *model-driven engineering* and *machine learning techniques*. It also presents and discusses the main related efforts in each research field. Chapter 3 describes the research context and the work scope. It also gives the definition of the problem and the main challenges. It finally presents our vision, methods and tools to resolve the problem, and an overview of the proposed solution. Chapter 4 covers our modelling effort and the architectural aspects of the solution. It provides a detailed study of the proposed *model-driven data warehouse* method. First, it presents standards and practices used for the method specification. Then, it describes the *model-driven design framework* part and the *model-driven engineering process* part. Finally, it provides the architecture of the conceptual framework for transformations learning in this context. In Chapter 5, the learning aspects of the solution are detailed. The chapter starts by the formalisation of key concepts used in our approach. Then, it studies the proposed machine learning approach (dependent-concepts) to learn transformation rules. Chapter 6 gives experimental results and discussion. Chapter 7 covers the implementation aspects of the solution. It introduces a design of a tool to support our proposals. It examines the *software-as-a-service* architectural model for software deployment and its use for business intelligence applications. It also presents the

proposed functional and technical architectures for *business intelligence-as-a-service* and *model-driven data warehouse-as-a-service*. The main perspectives and the future research challenges are presented in Chapter 8. Chapter 9 summarizes our contributions and gives our final conclusions and remarks.

Chapter 2

Background

In the problem that we are going to deal with, several concepts and research fields are considered. This chapter investigates the definition of these fields and the associated terminologies. It brings together the elements that are necessary to understand the context of our work. It presents the data warehousing architecture and gives an overview of current works on data warehouse components development. The model-driven architecture specification is explained and the used meta-modelling framework is defined.

The chapter also gives a summary of machine learning methods and presents with more details the inductive logic programming approach. It provides a review of related work in various areas (i.e., model-driven data warehouse approaches, model transformation by-example framework and concept learning strategies) that concern the solution provided by this dissertation.

2.1 Data Warehousing Architecture

Data models are the result of the application of design practices and informal rules specific to the organization. And this is more common today with the emergence of the business model concept that is increasingly diverse and complex. Furthermore, in the case of data warehousing systems, the model depends on the business goals to achieve. So, there are important requirements and constraints, which may be redundant, to include in the model. In this context, we need to know more about the data warehousing architecture and relevant works that concern the development of the data warehouse components. The data warehouse has become the central element of current *decision support systems* because it provides crucial business information to improve strategic decision-making processes [32]. The data warehousing architecture is defined through several heterogeneous and interrelated layers. Moreover, each layer contains different components, using different modelling profiles, and depends on several technologies. We distinguish five main layers (figure 2.1) that compose the data warehousing architecture: the *data-source* layer, the *integration* layer, the *multidimensional* layer, the *analysis* layer, and the *data-access* layer. In this architecture, we consider that the staging area is part of the integration layer, and the meta-data layer is an implicit layer shared between all other layers.

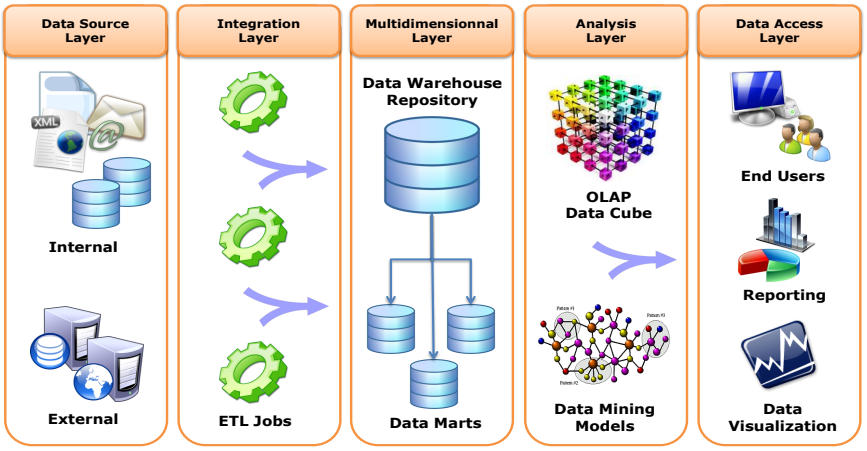

FIGURE 2.1: Data Warehousing Architecture.

The *data-source* layer defines the sources of information used to feed the data warehouse. It can be internal sources (transactional information systems, content management systems or files) or external sources (remote systems, web data). The

integration layer is responsible for data extraction, data transformation and data loading into data warehouse. It defines the *Extraction Transformation and Loading (ETL)* jobs and the mapping between data sources and data warehouse. The *multidimensional* layer defines the structure of the data warehouse repository. The dimensional modelling techniques [3, 32] are used to design the multidimensional structures (i.e., *facts* and *dimensions*) of this layer. The *multidimensional* layer can be organized as one companywide warehouse or/and multiple independent data marts. The *analysis* layer defines the mapping between the *multidimensional* layer and end-user applications. It contains special data structures (data cube and data mining models) that are used by the end-user applications for goals of analysis. The *data-access* layer defines the end-user applications used to access and to analyze data from data warehouse repository through the analysis layer. It contains *Online Analytical Processing (OLAP)* client's tools, reporting tools and so on.

Each of these layers presents its own problems and constraints, such as semantic schemas integration studied in [33, 34], semantic data integration [35], ETL process design and generation [36], multidimensional and analysis models derivation [37, 38]. Interoperability between layers, components portability and adaptability are the commons problems encountered in such architecture. In addition, business intelligence projects are still exposed to several technical risks and require more knowledge about the underlying business domain. These aspects increase the costs and the time of data warehouse development and make it a very difficult and challenging task.

Current data warehouse development approaches can fall within three categories: (1) The *framework-oriented* approaches focus on the data warehouse system design; (2) The *process-oriented* approaches focus on the data warehouse development process; and (3) The *unified* approaches that address the two problems (this category includes also the model-driven approaches). We summarize below the purpose of these *common approaches*, then we focus on the study of the *model-driven data warehouse* approaches which interests us most. In the *framework-oriented* category, we distinguish the approaches that focus on the design of one layer of the data warehousing architecture [6, 39, 40], and those dealing with more than one layer or all data warehouse layers [27]. We focus more on research efforts on data warehouse repository design and *extraction transformation and loading*

process development. For *data-sources* reverse and integration efforts you can refer to [33, 34, 41], and to [37] for the *analysis* layer modelling, and concerning *data-access* layer the works presented in [42, 43] will give more details.

Concerning the design of the data warehouse repository, authors in [6] propose a *Unified Modelling Language (UML)* profile for multidimensional modelling in data warehouses. Some standards are used in this approach such as, the *Object Constraint Language (OCL)* to specify the constraints attached to the multidimensional model and the *Query-View-Transformation (QVT)* for an automatic generation of the implementation in a target platform. In [7] a UML-based data warehouse design method that spans the three design phases (conceptual, logical and physical) is presented. A set of metamodels is used to design each phase, as well as a set of the OCL transformations to map schemas. However, these approaches focus only on the design of the data warehouse repository and no other data warehouse component is considered. Thus, these UML profiles do not provide a general standard framework like the *common warehouse metamodel* that is integrated to cover other layers of the data warehouse. Different other approaches [3, 44–46] for the data warehouse conceptual design are also proposed. These approaches share the same disadvantages with those previously presented and have other limitations such as: no standard notation and framework are adopted; no formal transformations are defined, and so on.

Concerning the development of *Extraction, Transformation, and Loading* processes, a conceptual model for ETL scenarios is proposed in [39]. But, no standard notation such as the *unified modelling language* is adopted. In [5] a UML based approach for modelling ETL processes in data warehouses is proposed. However, this UML conceptual metamodel does not comply with the *common warehouse metamodel* profile. Therefore, metadata interchange becomes difficult. The logical design of ETL scenarios was presented in [8] using a generic and customizable framework. In [9] a formal transformation between these conceptual and logical *extraction transformation and loading* models has proposed, the physical design and optimization of the ETL processes were studied in [47]. However, these transformations (conceptual to logical and logical to physical) are not based on the *query-view- transformation* standard. An approach for designing *extraction transformation and loading* processes using semantic web technologies is presented in [35, 48] when an ontology is used to model the domain and formally specify the

semantic of the datastore schemata. However, the definition of the ontology does not use a standard metamodel such the *Ontology Definition Metamodel (ODM)* that can be easily integrated in a *model-driven architecture* approach. In [40], authors present Orchid, a system that converts declarative mapping specifications into data flow specifications and vice versa. Orchid provides the *OHM (Operator Hub Model)*; a model for representing data transformation operations independently of specific ETL platforms. However, the definition of this model does not comply with the UML or the CWM. In addition to that, no standard approach using the *model-driven architecture* and the *query-view-transformation* is defined to generate mapping and ETL jobs.

The *process-oriented* approaches can fall within three major groups: *data-driven*, *goal-driven* and *user-driven*. A detailed comparison and discussion about the three approaches are presented in [49]. For example, Inmon [32] argues that data warehouse environments are *data-driven*, while the Kimball et al. [3] approach focuses on business processes in order to deliver consistent information throughout the organization. The data warehouse development process can also be a mix of these three approaches. In [10] a *data-driven* approach has been presented. Authors propose a semi-automated methodology to build a dimensional data warehouse model from the pre-existing E/R schemas that represent operational databases. In [11] authors present an approach based on the *SOM (Semantic Object Model)* process modelling technique in order to derive the initial data warehouse structure. Finally, *user-driven* approaches are presented in [12, 13], and a mix-driven approach is proposed in [14]. The main shortcomings of these approaches are as follows: (i) there is no proposal for an iterative and an incremental development using a standard process such as the *Unified Process (UP)* or the *Two Track Unified Process (2TUP)*; (ii) they do not propose a clear set of steps or phases including development best practices; (iii) the definition of the technical requirements is not taken into account; (iv) they do not address the whole data warehouse process; in most cases only the data warehouse repository is considered.

The other and general methods for the data warehouse [50] and *decision support systems* development [51] using the *unified process* and the *unified modelling language* are proposed. These methods offer the advantage of integrating the engineering process part and the design framework part. In fact, they propose at the same time the *data warehousing framework* and the *data warehousing process*.

These methods offer the advantage of integrating the engineering process part and the design framework part. But, they present several disadvantages such as: (i) they do not use any model driven approach to generate data warehouse diagrams and no transformation process using the *query-view-transformation* is defined; (ii) no standards metamodels for data warehouse development such as the *common warehouse metamodel* or the *CWM-eXtensions (CWMX)* are explicitly defined to design data warehouse layers; (iii) the development process does not take into account the definition of the technical architecture of data warehouses.

2.2 Model-Driven Data Warehouse

The *model-driven engineering* represents a promising approach to support software development practices [17, 52, 53]. This approach is mainly based on models, metamodels, and model transformations design. Indeed, the *model-driven development* strategy encourages the use of models as central element of development. The models are conformed to metamodels and the transformations rules are applied to refine them. Thus, transformations are the central components of the each model-driven process. The *Model- Driven Architecture (MDA)* standard [24] represents the *Object Management Group, Inc.*[1] implementation to support the model-driven approach. Founded in 1989, the *object management group* is an open membership, not-for-profit computer industry standards consortium that produces and maintains computer industry specifications for interoperable, portable and reusable enterprise applications in distributed, heterogeneous environments. Membership includes information technology vendors (e.g., IBM, Oracle) , end users, government agencies and academia.

The MDA starts with the well-known and long established idea of separating the specification of the operation of a system from the details of the way that system uses the capabilities of its platform. The three primary goals of the MDA are portability, interoperability and reusability. The *model-driven architecture* standard base includes also many specifications. These include the *unified modelling language*, the *Meta-Object Facility (MOF)*, specific platforms models (i.e., CORBA, JEE), and the *common warehouse metamodel* to design data warehouse components. The MDA

[1] http://www.omg.org/

includes as well: the *Ontology Definition Metamodel (ODM)* to design ontologies and to enable semantic model-driven development, the *Query-View-Transformation (QVT)* as a standard language for model transformation, and the *XML Metadata-Interchange (XMI)* as a standard format for models exchange and serialization. The tables 2.1 and 2.2 below describes main modelling and metadata specifications provided by the *object management group*. These standards which are part of the *model-driven architecture* framework, define a powerful tool for modelling complex systems that also helps companies develop future-proof technology investment.

TABLE 2.1: Core Standards of the OMG Metadata-Architecture.

Specification	***Summary***
Meta-Object Facility	The *Meta-Object Facility (MOF)* is metadata interface standard that can be used to define and manipulate a set of interoperable metamodels and their instances (models). The MOF also defines a simple meta-metamodel with sufficient semantics to describe metamodels in various domains.
Unified Modelling Language	The *Unified Modelling Language (UML)* is a standard modelling language for specification, construction, visualization, and documentation of the artefacts of a software system.
XML Metadata Interchange	The *XML Metadata Interchange (XMI)* is a standard mechanism for the stream-based interchange of MOF-compliant metamodels. The XMI is essentially a mapping of the *W3C's eXtensible Markup Language (XML)* to the MOF.

TABLE 2.2: Data Warehousing Standards of the OMG Metadata-Architecture.

Specification	***Summary***
Common Warehouse Metamodel	The main purpose of CWM is to enable easy interchange of warehouse and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments.
CWM Metadata Interchange Patterns	The purpose of CWM MIP specification is to add a semantic context to the interchange of metadata in terms of recognized sets of objects or object patterns.

The *model-driven architecture* is based on architecture with four meta-levels (figure 2.2), described as follows:

- The *meta-metamodel* level forms the foundation of the metamodeling hierarchy. The primary responsibility of this level is to define the language for specifying a metamodel. The level is often referred to as $M3$. The MOF *(Meta-Object Facility)* and the EMF *(Eclipse Modelling Framework)* are examples of meta-metamodels. MetaClass, MetaAttribute and MetaOperation represent elements of the model in this level.

- A *metamodel* is conformed to a meta-metamodel. The primary responsibility of the metamodel layer is to define a language for specifying (and describing) models. The metamodel layer is often referred to as $M2$. The UML and the CWM are examples of metamodels. Class, Attribute, Operation and Component define elements of the model in this level.

- A *model* is conformed to a metamodel. The primary responsibility of the model layer is to define languages that describe semantic domains, i.e., to allow users to model a wide variety of different problem domains, such as software, business processes and requirements. This layer is often referred to as $M1$. Product, Unit Price, Customer, and Sale are examples of model elements at this level.

- The metamodeling hierarchy bottoms out at $M0$ which contains the runtime instances (or data instances) of model elements. $< Chair >$, $< Desk >$, \$100, and \$200 are examples of information that we found in this level.

The *model-driven architecture* supports the idea of separating the specification of a system from the details of the way that system uses the capabilities of its platform. Thus, the MDA approach specifies several viewpoints (CIM, PIM, PDM and PSM) on a system (see figure 2.3 and below the description of these viewpoints):

1. The MDA transformation process starts by the definition of a CIM *(Computation Independent Model)* viewpoint. The CIM defines the requirements and goals for a system,

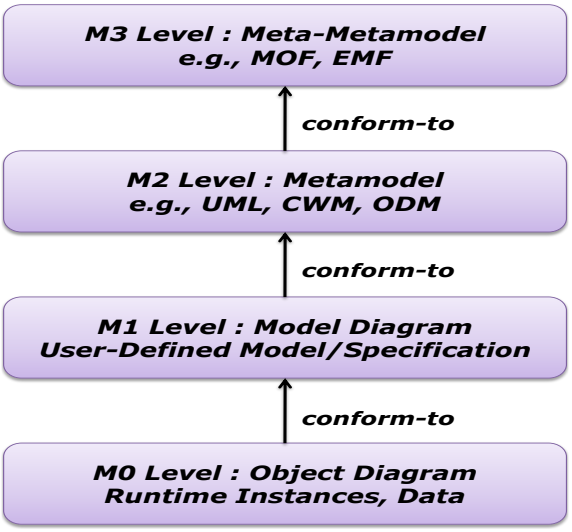

FIGURE 2.2: The Metamodeling Architecture.

2. Then, based on the CIM, the *model-driven architecture* encourages specifying the PIM *(Platform Independent Model)*. The PIM represents the system concepts with no specific information on the technology used to realize it,

3. This PIM is transformed into a PSM *(Platform Specific Model)* in order to include specific technology or platform information defined by a PDM *(Platform Dependent Model)*. The PDM represents the technical conceptual model, derived from the technical requirements description TCIM *(Technical CIM)*.

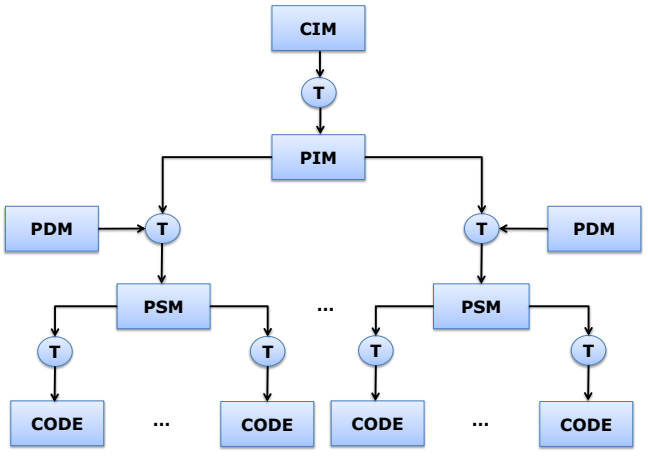

FIGURE 2.3: The MDA Transformation Process.

The *model-driven data warehouse* represents approaches that align the development of the data warehouse with a general *model-driven engineering* paradigm.

In [27, 28], model-driven oriented approaches for the development of data warehouses are presented. These approaches provide several advantages (simplified development, reusability, interoperability, etc.) resulting from the use of the *model-driven architecture* and the *query-view-transformation*. The approach presented in [28] describes derivation of *OnLine Analytical Processing (OLAP)* schemas from ER schemas. The source and target PIMs are respectively conform to ER and OLAP metamodels of the *common warehouse metamodel*. Authors describe how an ER schema is mapped to an OLAP schema and provide, also, a set of *query-view-transformation* rules (e.g., EntityToCube, AttributeToMeasure, RelationShipToDimension, etc.) to ensure this. The approach presented in [27] extends a previous work presented in [54]. It describes an integrated framework for data warehousing layers development with the *model-driven architecture* and the *query-view-transformation*. Authors focus on the MDA for multidimensional modelling and provide an extension of the *unified modelling language* and the *common warehouse metamodel* to build the different MDA models. They provide, using the QVT language, transformations (e.g., Fact2Table, Dimension2Table, etc.) between the MDPIM *(Multidimensional PIM)* and the MDPSM *(Multidimensional PSM)*.

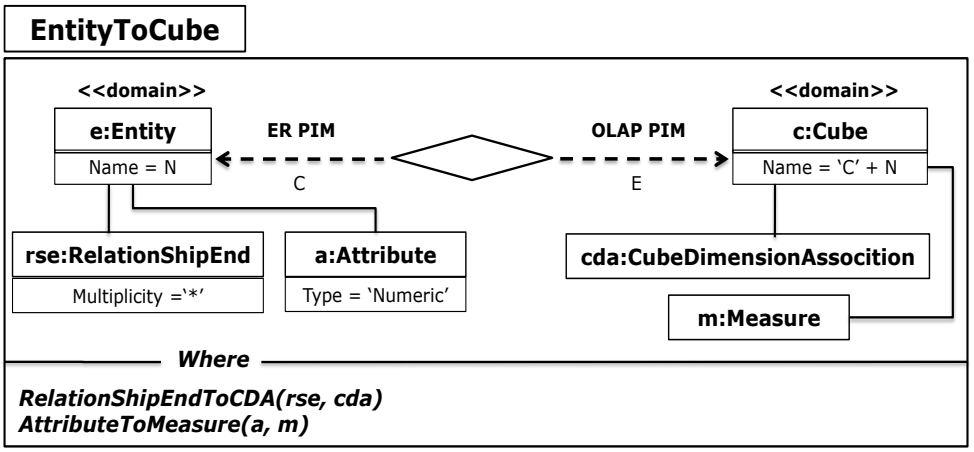

FIGURE 2.4: Graphical Notation of EntityToCube Relation.

The designed transformation rule in figure 2.4 (from [28]) shows a candidate Entity that gets transformed to a corresponding Cube. The generated Cube has the same name of the Entity but prefixed with a "C". Also, the transformation rules RelationShipEndToCDA and AttributeToMeasure must be done as post-conditions. The left part of the rule check the data-source elements (i.e., Entity,

Attribute, etc) while the right part defines the derived multidimensional elements (i.e., Cube, Measure, etc). By the transformation AttributeToMeasure, the numeric attributes of the candidate Entity, gets transformed to a corresponding measures of the Cube. Also, through the transformation rule RelationShipEndToCDA, each RelationShipEnd role with multiplicity equal to many is matching with a CubeDimensionAssosiation. Therefore, in the context of these two works, it is necessary to understand the *query-view-transformation* framework. Furthermore, the transformations are not built from previous projects which include knowledge on how source and target models are interrelated. Therefore, we need a machine learning based approach to synthesize transformation rules and assist developers.

2.3 Model Transformation By-Example

The *model transformation by-example* is related to several others supervised approaches: *query-by-example*, *programming-by-example*, and XSLT *(eXtensible Stylesheet Language Transformations)* generation by-example. The *query-by-example* approach [55] aims at proposing a language for querying relational data constructed from sample tables filled with example rows and constraints. Then *programming-by-example* [56, 57], where the programmer (often the end-user) demonstrates actions on example data, and the computer records and possibly generalizes these actions, has also proven quite successful. The by-example approach has also been proposed in the XML world to derive XML schema transformers [58–60], which generate XSLT code to carry out transformations between XML documents.

Varró et al. in [20], present an automated *model transformation by-example* approach using the *inductive logic programming*, an improvement of an initial proposal in [61]. The proposed method (based on Aleph ILP implementation) aims at the inductive construction of first-order clausal theories from examples and background knowledge (restricted to Prolog clauses). A running example is provided where a source class's diagram (based on the *unified modelling language*) is mapped into a target relational database diagram. In [62], authors present a general architecture for automating metamodel mapping using machine learning. They explore machine learning techniques and their applicability to *model-driven engineering* automation. Authors use the candidate elimination algorithm and

formalism is defined with a vector representation of hypotheses. However, no advanced experimentations and evaluation are presented.

Authors in [63], present a conceptual framework for *model transformation by-example* to derive ATL *(Atlas Transformation Language)* [64] rules. The approach uses the inter-model mappings representing semantic correspondences between concrete domain models which is more user-friendly then directly specifying model transformation rules or mappings based on the abstract syntax. The inter-model mappings between domain models can be used to generate the model transformation rules, by-example, taking into account the already defined mapping between abstract and concrete syntax elements. Strommer et al., in [65], extend the *model transformation by-example* approach to the domain of business process modelling languages. The definition of requirements for *model transformation by-example* in the context of business process modelling and the specification of proper mapping operators comprise the main contribution of authors.

In [66], authors present a by-example approach, named *MOdel Transformation as Optimization by Example (MOTOE)* which combines transformation blocks extracted from examples to generate a target model. Authors use an adapted version of *Particle Swarm Optimization (PSO)* where transformation solutions are modelled as particles that exchange transformation blocks to converge towards the optimal transformation solution. In a second paper [67] authors use the *Simulated Annealing (SA)* to improve the performances of the approach. Dolques et al. in [68], study the generation of transformation rules form transformations traces (transformations examples) using an extension of the *Formal Concept Analysis (FCA)*. FCA is based on the philosophical understanding that a concept is constituted by two parts: its extension which consists of all objects belonging to the concept, and its intention which comprises all attributes shared by those objects. Authors use the *Relational Concept Anaysis (RCA)*, one of the extensions of *formal concept analysis* that considers links between objects in the concept construction. Then, lattices allow rules classification and help navigation among the generated results to choose the relevant transformation rule. The experimental evaluations are provided using LATEX to HTML transformation examples.

Authors in [69] discuss the limitations of above approaches and introduce a new approach called *model transformation by-demonstration* instead of the *model transformation by-example* approach. The *model transformation by-example* idea is

about inferring the model transformation rules from a prototypical set of mappings. However, the *model transformation by-demonstration* approach asks users to *demonstrate* how the model transformation should be done by directly editing (e.g., add, delete, connect, update) the model instance to simulate the model transformation process step by step. Authors describe the *model transformation by-demonstration* steps and provide a motivating example. Finally, ontology-based approaches allow semantic reasoning techniques for metamodels alignment or matching. For example, in [70], metamodels are mapped to a pivot ontology, then an ontology-based reasoning is used to generate a Relational-QVT transformation. In [71] authors apply refactoring to metamodels in order to make explicit hidden concepts of metamodels and obtain an ontology where all concepts are reified before mapping. The *Similarity Flooding* [72] algorithm allows similarity values propagation in a labelled graph whose vertices are potential mappings, authors in [73] adapt it for metamodel alignment.

However, in all these cases, no relational approach well suited to the context of data models has been presented. Also, the notion of dependency between the concepts of the model has not been Addressed. According to the work about *layered learning* [21, 74], *context learning* [23, 75], *predicate invention* [76, 77] and *cascade learning* [78, 79] methods, We propose the *dependent-concept learning* approach that is based on the *inductive logic programming* framework.

2.4 Machine Learning Techniques

The goal of machine learning is to design algorithms that use example data or past experience to improve their performance for solving a given problem [80]. Many successful applications of machine learning exist already, including systems that analyze past sales data to predict customer behaviour, recognize faces or spoken speech, and optimize robot behaviour so that a task can be completed using minimum resources, and extract knowledge from bioinformatics data. In the past fifty years, a wide variety of machine learning techniques have been developed, just like many successful applications of machine learning. These applications vary from data-mining programs that learn to detect fraudulent credit card transactions, to autonomous vehicles that learn to drive on public highways. Machine learning

algorithms and techniques have long been used for various purposes in software engineering (testing, validation, security, etc.). For instance [81, 82] have studied the advances and perspectives in applications of such approaches in the software and data engineering fields. In our *model-driven data warehouse* framework, we propose to discover transformation rules from previous project experiences using supervised learning techniques.

2.4.1 Algorithms and Applications

One of the matters of interest is the steps to design a learning system. Mitchell in [83], discusses the following steps: (i) choosing the training experience, (ii) choosing the target function, (iii) choosing a representation for the target function and (iv) choosing a function approximation algorithm. A well-posed learning problem requires a well-specified task, performance metric, and source of training experience. A more precise definition of this is provided by Mitchell:

Definition 2.1. (**Learning from Experience**) A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

With respect to this definition, we can specify, as examples, the following learning tasks: (1) *play checkers learning problem* where the task T = *playing checkers*, the performance measure P = *percent of games won against opponents* and the experience E = *playing practice games against itself*; and (2) *handwriting recognition learning problem* where T = *recognizing and classifying handwritten handwriting words within images*, P = *percent of words correctly classified* and E = *a database of handwritten words with given classification.*

In many cases, the learning task involves acquiring general concept definition from specific training examples. This task is frequently referred to as concept learning, or approximating a boolean-valued function from examples. A more precise definition of this is also provided by Mitchell [83]:

Definition 2.2. (**Concept Learning**) When the learning task is to infer a boolean-valued function from training examples of its input and output, this is known as concept learning.

Considering this definition, examples of concept learning tasks are given below: (1) *movie suggestion* where the task is defined as T = *determine which movies a user will like*, performance as P = *percent of unwatched movies correctly predicted (like/dislike)* and experience as E = *examples of movies that the user likes and doesn't like*; and (2) *graduate school admissions* where T = *predict which incoming grad students will eventually complete their PhD, based on their applications*, P = *the percentage of correct predictions*, E = *past grad students and their eventual grad school outcome*.

Machine learning approaches and algorithms includes *Decision Tree Learning* as presented in [84–86], *Support Vector Machines*, studied in [87, 88], *Artificial Neural Networks* [89, 90], *Bayesian Learning* [91], *Genetic Algorithms* as works of [92, 93] and *Relational Learning* [31, 94–96]. In what follows, a definition of each category is given.

Decision Tree Learning is one of the most widely used and practical method for inductive inference. It is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. *Support vector machines (SVM)* are a group of supervised learning methods that can be applied to classification or regression. SVMs deliver state-of-the-art performance in real-world applications such as text categorisation, hand-written character recognition, image classification, bio-sequences analysis, etc., and are now established as one of the standard tools for machine learning and data mining. An *Artificial Neural Network (ANN)* is a system based on the operation of biological neural networks, in other words, is an emulation of biological neural system. ANNs combine artificial neurons in order to process information and to perform tasks that a linear program cannot.

A *Bayesian Learning* model (naive Bayes classifier) is a probabilistic model of the observed data based on applying the Bayes theorem. One of the most common approaches is the *Naive Bayesian Learner*, where the estimation of the likelihood is performed by means of the simplistic (naive) assumption that an attributes is independent from each other, given the class. *Genetic Algorithms (or GA for short)* is a programming technique that mimics biological evolution as a problem-solving strategy. GA-based methods are also known as evolutionary computation. Given a specific problem to solve, the input to the GA is a set of potential solutions to that problem, encoded in some fashion, and a metric called a fitness function that allows each candidate to be quantitatively evaluated.

Logical and Relational Learning is the study of machine learning and data mining within expressive knowledge representation formalisms encompassing relational or first-order logic. It specifically targets learning problems involving multiple entities and the relationships amongst them. This field covers *Inductive Logic Programming (ILP)*, *Multi-Relational Data Mining (MRDM)* and *Statistical Relational Learning (SRL)* subfields. The domain knowledge in traditional learning techniques has to be hardcoded into an algorithm and changes to domain knowledge often require substantial modification of the program. Having sufficient background knowledge often makes *inductive logic programming* learning much easier and more effective. Indeed, learned hypotheses from ILP are more understandable than many other learning techniques due to its powerful first-order representation. This has made ILP particularly attractive and a popular machine learning technique. The next subsection provides more details about the ILP framework, which will be used to extend to extend our model-driven method.

2.4.2 Inductive Logic Programming

The term logical and relational learning to refer to the subfield of artificial intelligence, machine learning and data mining that is concerned with learning in expressive logical or relational representations. The *inductive logic programming* [94] is also an active research subfield of machine learning that addresses relational learning and uses a first-order representation of the problem domain and examples. Its objective is to provide practical algorithms for inductively learning hypotheses, expressed as logical rules [95]. It was only with the advent of *inductive logic programming* in the early 1990s that the field of relational learning became popular. Whereas initial work was often concerned with logical (or logic programming) issues, the focus has rapidly changed to the discovery of new and interpretable knowledge from structured data, often in the form of rules, and has soon generated important achievements in many application domains. The ILP approach extends the theory and practice of computational logic by investigating induction rather than deduction as the basic mode of inference [31].

The *inductive logic programming* approach offers several advantages: it uses a powerful representation language and the given results are easy to understand. It is possible, also, to add information about the domain (the domain theory).

So, complex models can be constructed and it is easy to understand the given results. The ILP systems have been applied to various problem domains which benefit from the generated relational descriptions. Finally, a lot of free and open-source ILP implementations are available online such as: Aleph [97], Foil [98, 99], Progol [100], Tilde [101], Golem [102], etc., and this will help us to evaluate our learning approach. At the same time ILP has some disadvantages, such as: in complex situation it takes long time to get the results; it is difficult to use and it is necessary a experienced in ILP user to implement the systems; also the search space grows very quickly with the number of relations in the background knowledge.

An ILP learning task is defined by four aspects: the *model theory* defines the semantic constraints on hypotheses, i.e., what to search for; the *proof theory* describes the strategy to perform the search; the *declarative bias* explicitly defines the hypothesis space, i.e., where to search; and the *preference bias* is concerned with the generalisation performance of ILP. For a hypothesis to become a solution, semantic requirements set by an ILP learning task have to be satisfied. The ILP learning tasks can adopt two important settings: *descriptive* and *predictive settings*. The descriptive setting is basically concerned with learning a set of rules (in the form of a clause program) that holds for all examples. It is often considered as a first-order upgrade to propositional data mining, such as association rules [103]. ILP systems that adopt the descriptive setting include Claudien [104–106], HR [107–109], and Warmr [110–112]. The predictive setting is a form of supervised learning, which is concerned with learning a set of rules (in the form of a clause program) that best separate *positive examples* (denoted by E^+) from *negative ones* (denoted by E^-). The well-known predictive ILP systems include Progol, Aleph, Foil, Tilde, and ICL [113]. In our approach, we are concerned by the predictive setting using the well known Aleph framework. The predictive setting in ILP is also defined on the basis of the *cover* relation. In fact, a key to understand this setting is the concept of the *cover* relation. The *cover* relation defines a boolean relationship between a hypothesis and an example. The *cover* relation is the basis of hypothesis completeness and consistency definitions. The predictive setting and the *cover* relation are defined below.

Definition 2.3. (**Predictive Setting**) Given background knowledge B, some positive examples E^+ and negative examples E^-, the predictive setting is to learn

a hypothesis H, such that H covers all positive examples and none of the negative examples, with respect to B.

Definition 2.4. (**Cover Relation**) An hypothesis H covers (or satisfies) an example e, denoted $covers(H, e)$, if and only if $H(e) = true$. $covers(H, E)$ if and only if $\forall e \in E, covers(H, e)$.

The task of learning of a single concept (predicate) in ILP can be formulated in this way: **Given**: A set of examples E described in a language $\mathcal{L}_e$ with: positive examples, E^+ and negative examples, E^-; A target predicate p; A language description of hypotheses, $\mathcal{L}_h$, giving the syntactic restrictions in the definition of the predicate p; The background knowledge B, described in a language $\mathcal{L}_b$, defining predicates that can be used in the definition of p and can give additional information about arguments from the examples of the target relation; and a cover relatioship between hypotheses of $\mathcal{L}_h$ and examples of $\mathcal{L}_e$.

In general, we seek for a complete hypothesis (covering all positive examples) and correct (does not cover any negative example). More formally it is defined as follows: **Find**: A definition H for p, expressed in $\mathcal{L}_h$, so that $B \wedge H = E^+$ (i.e., completeness); and $B \wedge H \neq E^-$ (i.e., consistency). Therefore, when we consider the background knowledge, the completeness and consistency also have to be redefined as:

Definition 2.5. (**Completeness and Consistency**) A hypothesis H is *complete* with respect to background knowledge B and examples E if all the positive examples are covered, i.e., if $covers(B, H, E^+) = E^+$. A hypothesis H is *consistent* with respect to background knowledge B and examples E if no negative example is covered, i.e., if $covers(B, H, E^-) = \emptyset$.

Any mechanism employed by a learning system to constrain the search for hypotheses is named a bias [95]. The language bias defines the space of formulas used to represent hypotheses and can be considered as part of the background knowledge. Bias can either determine how the hypotheses space is searched (i.e., the search bias) or determine the hypotheses space itself (i.e., the language bias). In general there are three ways how to limit the size of the set generated by a refinement operator: to define bias, to accept assumptions on the quality of examples, or to use an expert [114]. The search bias refers to how the system makes the search between

the space of clauses. The exhaustive search is responsible to run all the space clauses and the heuristic search indicates us which parts of the search should be used or ignored. The language bias defines the space of formulas used to represent hypotheses. By selecting a stronger language bias the search space becomes smaller and learning more efficient; however, this may prevent the system from finding a solution which is not contained in the less expressive language. The bias should be weak enough to allow complete and consistent programs and, simultaneously, enough to have a good performance. Regarding the predictive setting and language bias definitions, the objective of the ILP is to induce a hypothesis H that, with the given background knowledge B, explains the examples E^+ and is consistent with E^- [95].

2.5 Summary and Further Reading

In the proposed approach, we will define the transformation learning problem as a concept search one. Concept learning can be viewed as search of the space of concept descriptions. The predictive setting is applied and a knowledge base of projects traces is used to define background knowledge B, some positive examples E^+ and negative examples E^-. A new architecture based on optimized metamodels is used to define the hypothesis language $\mathcal{L}_h$ and to determine the search space. We propose metamodels (e.g., UML and CWM) that allow extracting a set of effective predicates to define the hypothesis language, and that best meet the *model- driven data warehouse* problem. Then, we will also investigate a new machine learning methodology stemming from the application needs: learning dependent-concepts. Based on existing concept-search methods as *layered learning*, *context learning*, *predicate invention* and *cascade learning*, we propose a new methodology that automatically updates the background knowledge of the concepts to be learned (i.e., the learned child-concepts are used to update the background knowledge of parent-concepts). In the following paragraphs we provide a brief summary of each approach. A comparison between these methods and the proposed *Dependent-Concept Learning (DCL)* approach will be given in Chapter 5.

The *layered learning* machine learning paradigm is firstly introduced in [21]. Authors in [74] study the problem of constructing the approximation of higher-level

concepts by composing the approximation of lower-level concepts. In [115, 116] authors present an alternative to standard genetic programming that applies layered learning techniques to decompose a problem. The layered learning approach presented by Muggleton in [117] aims at the construction of a large theory in small pieces. Within the field of *inductive logic programming*, the term *predicate invention* has been introduced in [118] and involves the decomposition of predicates being learned into useful sub-concepts. Muggleton in [76] defines *predicate invention* as the augmentation of a given theoretical vocabulary to allow finite axiomatisation of the observational predicates. Stahl in [77, 119], studies the utility of *predicate invention* task in ILP and its capabilities as a bias shift operation. Authors in [120], investigate a specification language extension when no examples are explicitly given of the invented predicate. In [22], authors introduce the *cascade generalization* method. This approach is compared to other approaches that generate and combine different classifiers like the *stacked generalization* approach [121–123]. In [79] author proposes several speed-up variants of the original cascade generalization and show that the proposed variants are much faster than the original one. The *model transformation by-example* approach aims to find contextual patterns in the source model that map contextual patterns in target model. This task is defined as *context analysis* in [30]. The machine learning approaches that exploit context to synthesize concepts are proposed in [23, 75]. In [23] author provides a precise formal definition of context and list four general strategies for exploiting contextual information. Authors in [75] introduce an enhanced architecture that enables contextual learning in the *Neurosolver* (a problem solving system).

Chapter 3

Methodology

Decision support systems development is defined, by taking into account several business, conceptual and technical constraints. Therefore, the definition of a solution for an automatic generation of data warehouse components becomes increasingly difficult. Also, current approaches propose a general framework applicable only in simple examples of data-models, and away from the business and technical complexity of data warehousing platforms. From the business point-of-view, the generation of a decision (or analysis) model in accordance with needs and goals is required. From the technical point-of-view, the urbanization of the decision support and data warehousing systems can be made using one or more technologies. From the implementation (also the deployment) point-of-view, the Software-as-a-Service architecture is suitable as delivery model for business intelligence.

In this chapter, we describe the context of this work and we summarize the purpose of each task. The problem statement will be discussed with more detail in order to motivate the reason for this study. We explain what we mean by model-driven data warehouse automation and business intelligence-as-a-service. Research questions and the general and specific objectives were tackled. This is followed by our vision and an overview of the proposed solution.

3.1 Context

We have showed that the design of transformations require an expert who knows the business domain, the principles of model-driven approach, and the transformation languages [29, 124, 125]. Thus, the actual *model-driven data warehouse* automation processes are partial because the design transformations remain manual (for example, no proposal of a framework to learn these transformations). This could increase the time and cost of developing the decision support information system. Also, there is no guarantee that the proposed transformations are used for any given data-model, and that elements defining the mapping are consistent with the initial (business and technical) requirements. The basic idea is that: the dependencies between metamodels concepts (e.g., Entity, Attribute, Cube, and Measure) create a post-conditions dependency within the definition of transformations (e.g., EntityToCube and AttributeToMeasure). And so, this kind of dependencies may change the way of transformations design and thus enable better finding elements involved in the mapping.

We also showed that recent approaches, called *model transformation by-example*, propose to automatically generate the transformations. These methods are based on artificial intelligence techniques and try to learn the transformation from experiences of previous developments. However, we identify three main limitations of current by-example approaches: (i) They always treat a simple software design or example; no adjustment for *decision support systems* has been made taking into account the technical and the business complexity. (ii) There is no proposal for an advanced learning system with a modelling bias to reduce the search space of transformation rules and thus to increase performance. (iii) Current works do not offer advanced experiments on a reference databases allowing the definition of the best way to learn rules, and so supporting the adaptation the learning system (for example the problem of dependent-concepts).

In the light of this context, this research aims to provide an unified and intelligent method for decision support and data warehousing systems development and so contributes to improve current approaches. Thus, this thesis deals with the *model-driven data warehouse* engineering and how it can be automated using machine learning techniques. We study the architecture of a framework that allows connecting the data warehousing architecture, the model-driven paradigm and

inductive learning approach. We focus on *relational learning* and we use the *inductive logic programming* framework. The aim of ILP is to induce a compact, correct, generalised, and *understandable-by-humans* description of facts. It induces a set of rules from a given set of facts in a very expressive formal language. The knowledge base (of projects experience) is used to define those facts and in general the inputs of the inductive learning process. We propose, by using an ILP approach to express the model transformation context, which transformation rules in the framework of *model-driven data warehouse* can be discovered or synthesized from previous projects experience. To this end, the proposed approach extends the application scope of the relational learning and contributes to define the strategy that answers best the use of machine learning in the model-driven context. A new language based on the first-order logic for representing transformation rules is defined. The new language, with the expressive power of first-order logic, allows revealing of much complicated relations and information among model objects, attributes of the objects and the domain theory (for effective reduction of the UML-CWM transformation problem into ILP).

The first task of this work is the integration of the *framework-oriented* and the *process-oriented* approaches in order to provide a unified model-driven approach. We mainly use of the *unified modelling language* and the *common warehouse metamodel* as standards notations for defining the *data warehouse design framework*, and we propose the *Model-Driven Architecture (MDA)* and the *Two Track Unified Process (2TUP)* standards implementations for defining the *data warehouse engineering process*. This part of the work will provide answers to questions on data warehouse components derivation, standards used for design and the engineering process and as a result it defines the structure bias (a first level to simplify and reduce the search problem).

The second task concerns learning dependent-concepts. We find that in the *model-driven data warehouse* application, dependencies exist between transformations. So, we investigate a new machine learning methodology, the *Dependent-Concept Learning (DCL)*, that is suitable to solve this kind of problem. The objective of the DCL approach is to build a pre-order set of concepts on this dependency relationship (a dependency-graph is defined): first learn non dependent concepts, then, at each step, add the learned concept as background knowledge for next concepts to be learned according to the pre-order. On the contrary, the *Independent-Concept*

Learning (ICL) approach, widely used by current *model transformation by-example* frameworks, proposes to learn the set of considered concepts independently. The *DCL* approach is implemented and applied to our transformation learning problem. And, compared to the ICL, the experimental evaluation shows that the dependent-concept learning approach gives significantly better results.

The third task of our work is to design and to implement a new platform with the best practices in engineering and in deployment of the solution (including the proposed *model-driven data warehouse* framework). Indeed, in recent years the data warehousing infrastructures have undergone many changes in various aspects. This is usually due to many factors: the emergence of *Software-as-a-Service (SaaS)* architecture model [15, 16, 126]; the success of agile and iterative data warehouse development approaches; the changing needs of organizations and the extension of the data warehouse into new application areas; and the evolving of standards and open-source technologies. Also, the partner company (i.e., Intelligence Power) plans to dispose an innovative solution that addresses current and future issues of business intelligence. We study the *software-as-a-service* through successful business intelligence implementations that have embraced this model and we propose a functional and technical architecture for a common *business intelligence-as-a-service* deployment infrastructure [127, 128]. We introduce the model-driven warehousing development environment, named *model-driven data warehouse-as-a-service* as part of the common infrastructure. We give recommendations for standards and technologies used to implement the proposed architectures design. This will provide a complete support for data warehouse engineering approaches and to contribute to improve current data warehouse development tools.

3.2 Research Problem

The automation of information systems engineering and, in particular, data warehousing systems remains a very challenging task. Several data warehouses design frameworks [6, 37, 39, 40, 47] and engineering processes [10–14] have been proposed during the last few years. However, the *framework-oriented* approaches fail to provide an integrated and standard framework that addresses the design of all data warehouse layers. The *process-oriented* approaches fail also to define an engineering

process that addresses the whole development cycle of data warehouse with an iterative and incremental manner while considering both the business and the technical requirements. In addition, not much effort was devoted to unify the framework and the process into a single integrated approach and no automated architecture for engineering or a learning model to support the evolution of the system are proposed.

In current architectures and modelling solutions it is difficult to answer several questions needed to ensure a reliable automatic process. For example, from the business point-of-view, the following questions are partially treated: how to generate a decision (or analysis) model in accordance with needs and integrating new and existing business constraints defined in previous projects? how to ensure the development of the system if requirements or goals change (i.e., adaptability of the system)? and how to reduce the costs of development and deployment of the overall system. Then, technically, the questions of how to ensure interoperability at conceptual and operational levels of the data warehousing architecture layers and how to be independent from target platforms and better manage the heterogeneity of the system. Several standards are available, UML and CWM for modelling and mainly the *Query-View-Transformation (QVT)* as transformation language. However, the use as such of these standards by experts is difficult, because no detailed architecture showing the possible transformations for each layer (and components) of the system is proposed. Also, the use as such of these standards makes learning difficult or impossible because the number of elements in these specifications is significant which enlarges the assumptions search space (we propose a structure bias). So, a splitting by layer and components of these specifications are required to reduce the search space.

The *model-driven data warehouse* framework provides several advantages (reusability, interoperability, etc.) resulting from the use of the *model-driven architecture* and the *query-view-transformation*. For example, In the case of actual approaches [27, 28] the Y process of the multidimensional layer derivation is defined as the schemas of figure 3.1. The task of *"designing and applying transformation rule"* is the core step of this model-driven warehousing process. The data-source model, denoted as *DSPIM (Data-Source Platform Independent Model)* and resulting from data sources analysis step defines the first input of the task (it represents a conceptual view of a data-source repository). Then, the business requirements model,

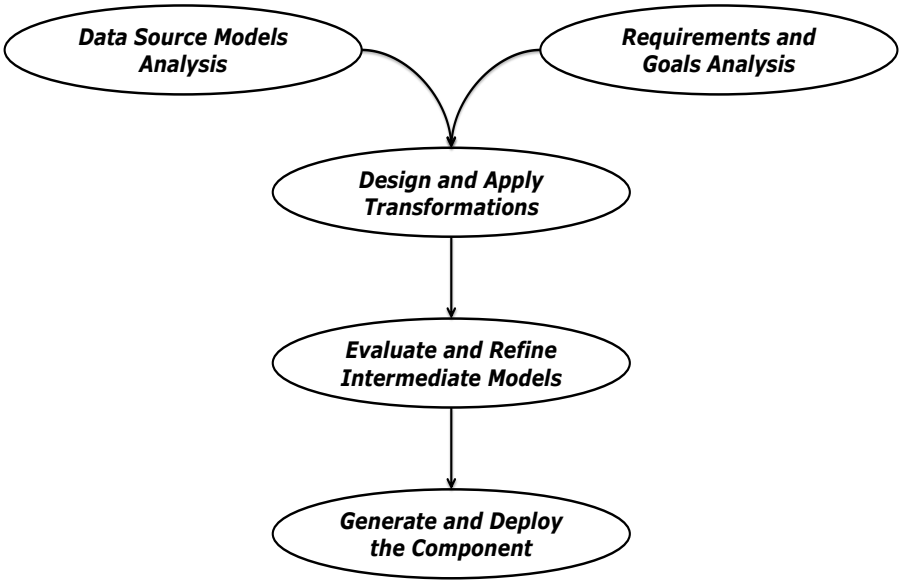

FIGURE 3.1: An Y Cycle for Multidimentional Layer Derivation.

denoted as *CIM (for Computation Independent Model)* in the MDA specification and resulting from the study of organization requirements and goals represents the second input. The results of applying transformation is a multidimensional data-model (i.e., the target-model), usually denoted as *MDPIM (Multidimensional PIM)*. The MDPIM represents a conceptual view of a data warehouse repository. The tasks of *"evaluate and refine intermediate models" and "generate and deploy the component"* is concerned by the validation of the generated models/codes and their refinement (e.g., conceptual-to-logical and logical-to-physical) by applying the corresponding transformation rules.

The development of model transformations is a very hard task. Almost certainly, designers or programmers must have serious skills with the corresponding meta-models and the transformation languages [64, 129]. This creates many risks and challenges during design of the transformations which makes the approach more complex and entails additional costs. Thus, in this situation the automation problem requires addressing many other questions: for example, which are the specifications used to provide an integrated framework that automatically learns transformations from previous data warehouse projects experiences? Then, what is the common conceptual architecture, and what are the tools and techniques used to define such framework and how it can be used within the proposed *model-driven data warehouse* approach? Finally, the problem of which transformations we need

to learn and in what order (i.e., the question of the search bias); which knowledge representation language could offer the most benefits in this context and which specific learning problems can we find in the case of *model-driven engineering*? This helps in improving learning results and the quality of transformation rules. In chapter 5, we answer these questions by proposing a framework and an approach that respond to how we conduct the search for a solution for a given concept.

Decision support databases are usually very large and the input data for model design or for decision making are often substantial. In these areas, the scalability is a key requirement. Furthermore, response times must be very short for such processes that support transformation rules modelling and learning. Indeed, several alterations/ modifications, upgrades or iterations can be considered during the development process. When we look at the problem of automation using machine learning techniques, current and recent approaches are not directly applicable in context of the data warehouse architecture. For example, different approaches are proposed in [20, 63, 66, 68]. But they use simplifications of rules design or templates-based examples representations. Some methods are not applicable on datasets with a large number of models or a large number of components (which is the case of the data warehousing architecture). Mainly, they propose general frameworks (and processes) for components design (and for components engineering). And, no structuring of an architecture based on partitions is defined. These approaches do not explain how the problem of scaling will be able to be resolved and, so, their application remains partial. Thus, the proposed designs are not made under duress of scaling. The reduction of data warehouse models transformation problem and the issue of scalability are at the heart of this thesis.

The *model-driven data warehouse* is a new and a complex application whose context is different from usual applications. In this context, the learning process will use data schemas (named also data patterns) to set examples and background knowledge. These schemas are in the form of *class diagram* in the case of source-models and *multidimensional diagram* or the *On-Line Analytical Processing (OLAP)* for the target-models. Thus, these diagrams are defined in terms of relations between several concepts: e.g., *class, attribute, association* in the case of source-diagram and e.g., *cube, measure, dimension* for the target-diagram. Hence, it is essential to choose a language to express such relational descriptions between objects. In addition, as part of data warehousing systems, multiple business rules can be used

as domain theory. Therefore, this domain theory can enrich the learning context and thus improve performance and transformation rules quality. Consequently, to ensure evolution and adaptability of the learning process, it is necessary to choose a language that allows extending the background knowledge by domain theory.

The *attribute-value* learning is a setting for machine learning where the data to be analyzed are given in a tabular format: each row represents an individual, each column represents a property of the individual (also referred to as attribute or variable), and the entries in the table are the values of these attributes for each individual. The *attribute-value* setting is widely used in statistics and data mining problems. However, attribute-based learning has the following strong limitations: (i) background knowledge can be expressed in rather limited form; (ii) the lack of relations makes the concept description language inappropriate for some domains. On the other hand, the relational learning (as the ILP approach) provides the appropriate approach to answer this problem. With the relational learning framework, it can easily express the relational information between concepts. This is an advantage, because the defined relational information (the context) plays an important role in the resulted transformation rules. The advantage in ILP systems is that the expert has the ability to define the language bias. In our implementation, the main task now is to define relations and predicates to define appropriate hypothesis. We propose a language that improves the quality of rules to simplify the validation task by the expert but also performance of the learning process. In our approach, we keep an invariable number of predicates extracted from metamodels, which are already very well optimized and especially designed to meet this problem.

Within the evolution of web architectures, we need to explore several aspects that may influence the next generation of data warehousing platforms: First, we deal with the problem of what is the functional and the technical architectural requirements for *business intelligence-as-a-service* deployment? Then, we focus on the question of how the *model-driven data warehouse-as-a-service* architecture is defined and the challenges towards model-driven data warehousing and components development in the cloud[1]. Finally, we make proposals to address the issue of which open industry standards and technologies are recommended for the implementation of such architectures. This helps to extend the current architectures of data

[1] *The software-as-a-service applications are part of the cloud computing stack.*

warehousing systems and allows the company to have their own and new business intelligence development model. Our proposals and contributions in this part are present in chapter 7.

3.3 Solution Overview

We start by answering the following question: *which standard approach is used to automatically derive the data warehouse components from existing models?* This is to be based within a general framework providing basic concepts for the automation of development. We address also the question of *which standards are used to design the data warehouse components and for models representation* and the question of *which engineering process is used to provide an integrated method for data warehouse development.* This allows us to provide a complete architecture, considered as modelling/structure bias for the learning process. This architecture allows the definition of a hierarchy of transformations (conceptual, logical and physical). It allows the knowledge of useful transformation to learn; it also allows the knowledge for which transformations, machine learning could add value. Indeed, the proposed architecture allows identifying transformations that may be treated differently (e.g., using existing tools). The engineering process defines the input and output models (or schemas) for each transformation to learn. So, it simplifies the design of the learning process and the definition of predicates used for background knowledge and examples.

We propose to solve the scalability problem on several levels: In a first step, we propose an optimal *model-driven data warehouse* architecture based on industry frameworks (the modelling specifications) and a standard process. This helps creating a partition in the overall architecture and offers a simple and effective way to integrate the learning methods. Partitions allow reducing the number of predicates used in the learning of each concept. Therefore, reduce the search space that remains invariant by increasing the number of data models. This is a significant bias in the framework that we propose. It is defined as a modelling bias (or also as architecture bias) . In fact, the architecture of model-driven data warehouse is a complex architecture that uses multiple components and presents several risks. So, this modelling step is important to manage these risks and makes effective the task

of transformations learning. In our problem, we will first tackle the identification of transformations useful to learn. Indeed, we identify transformations when learning can bring more value and transformations for which existing techniques or approaches can be used. As part of this task (modelling bias definition), we propose for transformations to learn: (i) the input and output models of each learning task, and (ii) the appropriate metamodels and their detailed specification. This modelling bias is fundamental for understanding how to integrate the ILP approach, and how to make learning and how to deploy the application effectively.

We define an approach to derive a set of rules that can assist the developer to identify contextual metamodel elements. These contextual metamodels information is used to design the final transformation rules. The automated approach for transformations learning and its application in context of *model-driven data warehouse* can be summarized in a three-stage process. Figure 3.2 shows these steps of the automated data warehouse development approach. This part of the work will provide answers to questions on the conceptual framework to learn transformations from projects trace and in identifying specific learning problem in the context of *model-driven warehousing*. We contribute to the definition of the modelling bias, the language bias and we propose a solution that best meets the problem of transformations learning in a model-driven setting.

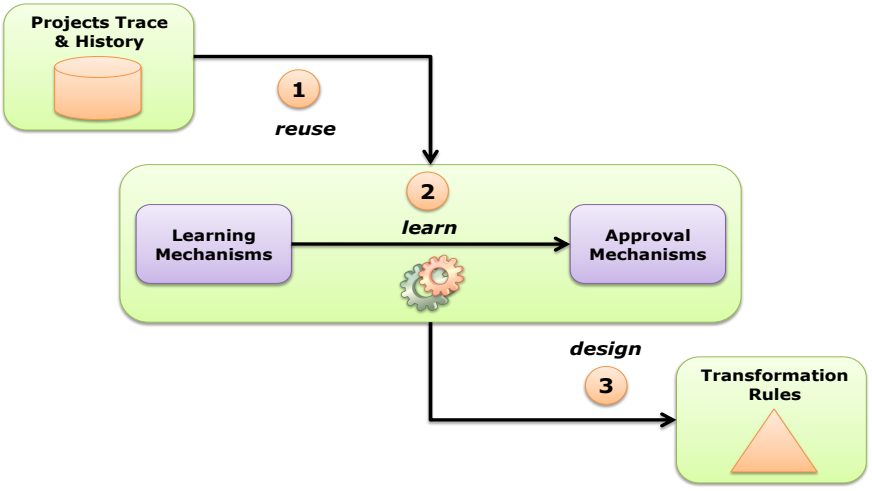

FIGURE 3.2: Automated Transformations Learning Environment.

First, projects trace (or the knowledge base) defines the input of the transformations learning engine. Then, a *model-driven data warehouse* workbench (an environment for models design and derivation) stores the generated rules. So, rules can be

validated, adapted or redesigned by the expert, to comply with the supported transformation languages (e.g., *query-view-transformation*). The QVT is a standard set of languages for model transformation defined by the *Object Management Group (OMG)*. It addresses the need for standardizing the way mappings are achieved between models whose languages are defined using the *meta-object-facility* (the standard metadata interface). Please refer to Appendix B for details concerning the *query-view-transformation* standard. Finally, transformations are applied to refine the designed models and to generate data warehousing components. We used as learning engine the well known Aleph ILP system [97], because of its ability to handle rich background knowledge, made of both facts and rules. Aleph follows a top-down generate-and-test approach. It takes as input a set of examples, represented as a set of Prolog facts and background knowledge as a Datalog program.

We define the transformation learning problem as a concept-search one. Concept learning can be viewed as search of the space of concept descriptions. The hypothesis language determines the search space. The input of the *inductive logic programming* comprises two types of information, examples and background knowledge. While *background knowledge*, denoted by B, formalises existing knowledge on a problem domain, *examples*, denoted by E, describe some underlying concepts not present in background knowledge. Besides these conditions, the outcome of an ILP system depends on several additional factors, the so called bias. For example, the language bias aims at excluding unsuitable hypotheses from the hypotheses language $\mathcal{L}_h$. This allows reducing the search space and so, performances enhancement in general. Therefore, one of the problems on which we focus in this thesis is to find an optimized representation for the language bias in the *model-driven data warehouse* framework. This language bias (also known as declarative bias) aims to restrict the representation to clauses that define best the transformation rules. The proposed representation allows for an improved learning process and also for a good quality of the obtained rules. This will facilitate the task of understanding and validation of the learned rules.

We have proposed the UML CORE and the CWM OLAP metamodels to design respectively the source and the target models. The objective is (1) to provide a set of model transformation examples at the model level (the model layer of MDA, referred to as $M1$, defines elements that describe semantic domains); then

(2) to run a learning engine that will build rules at the metamodel level (the metamodel layer of the *Model-Driven Architecture (MDA)*, referred to as $M2$, is responsible of defining a language for specifying models). We mainly focus on how to find an optimized representation of assumptions, and so the language bias in the *model-driven data warehouse* framework. This language bias aims to restrict the representation to clauses that define best the transformation rules. The proposed representation allows for an improved learning process and also for a good quality of the obtained rules. This will facilitate the task of understanding and validation of the learned rules. This task of defining the knowledge representation language is one of the contributions of this work. The contribution concerns the issues of translating a CWM-UML transformation problem into a *relational learning* problem. Thus, the definition of predicates is made following the study of metamodels, part of the *model-driven architecture*. MDA is a standard recognized by the software engineering community. Our work consists in studying and identifying the metamodels that are useful and appropriate for data warehousing systems. The choice of appropriate metamodels allows obtaining transformation rules similar to transformation designed manually by developers using standard languages (e.g., QVT [29], ATL [64], Eclipse M2M [125]). This helps to assist the expert in the validation and the integration process of these rules.

We define the language bias using the metamodel level ($M2$ level) of the metamodelling architecture. This gives the advantage to define a clear set of predicates with an optimal level of abstraction. Predicates obtained from the $M2$ level will ensure obtaining understandable transformation rules, equivalent to transformation designed manually. In the proposed *model-driven data warehouse* framework, this process will extract predicates from UML CORE and CWM OLAP metamodels. UML CORE defines the predicates used for the representation source-models examples (denoted as DSPIMs). CWM OLAP defines the predicates used for representing target-models examples (denoted as MDPIMs). For example, the following are part of the defined predicates to describe source models (DSPIMs): *class(Name); property(Name, Type, Lower, Upper); association(Name, Source, Target)*. As part of the defined predicates to describe target models (MDPIMs): *cube(Name); measure(Name, Type, Cube); dimension(Name, isTime, isMeasure)* . And *type(Name) and multiplicity(Bound)* are defined as common predicates used for all models definition. As an example of trace model, consider a data schema used to manage customers and invoices (figure 4.15). With respect to

the defined UML CORE and CWM OLAP predicates the following code is example of the generated background knowledge program of DSPIM: *type(integer); type(float); class(invoice); class(customer); property(amount, float, 1, 1).* And, *cube(invoiceFact); measure(amount, float, invoiceFact); dimension(customerDim, false, false)* as part of the generated background knowledge program of MDPIM.

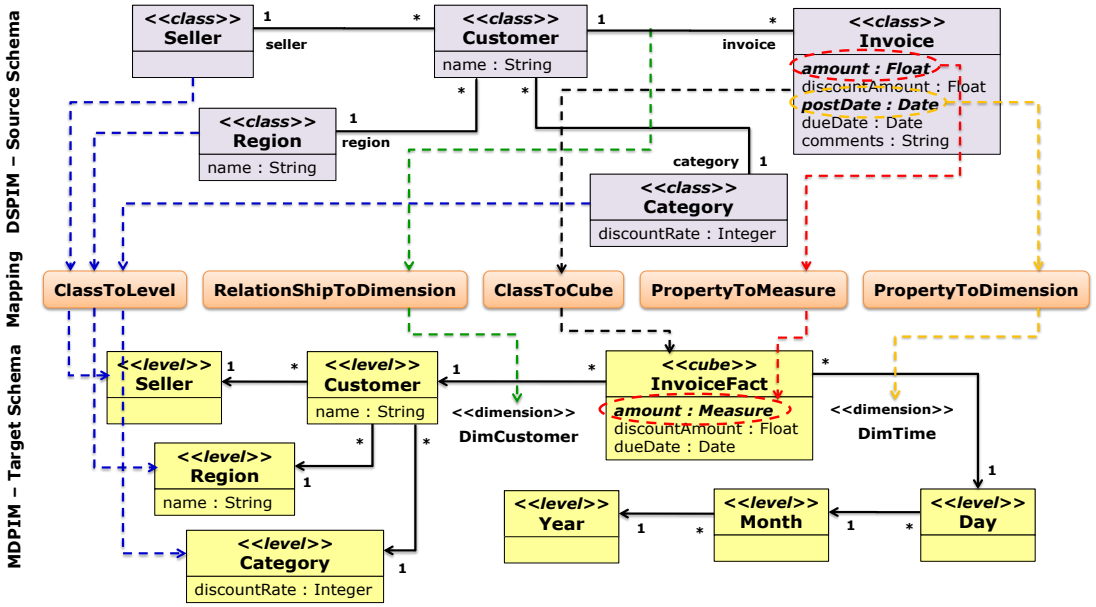

FIGURE 3.3: Mapping UML CORE Instance to CWM OLAP Instance.

The *mapping model* describes all transformations of source elements to target elements. The transformation predicates are of the form: *transformation(SourceElement, TargetElement)* where SourceElement and TargetElement represent, respectively, input and output of the transformation rule. A transformation rule expresses in which conditions or in which context an element SourceElement in the source model is transformed into a TargetElement in the target model. For instance, the question is: in which context a *class* in the UML CORE model can be translated to a *cube* in the CWM OLAP model. Given project traces, we extract the situation where the Invoice class is translated into a cube InvoiceFact, each such situation defines a positive example. Similarly, the situation where a *class* is not transformed into a *cube* defines a negative example. Then, Aleph is run on this training set. In the example of figure 4.15, *classtocube(invoice,invoiceFact)* is a positive example and *classtocube(customer,invoiceFact)*, *classtocube(seller,invoiceFact)*, *classtocube(region,invoiceFact)* are negative examples. This reduction of the UML-CWM problem to ILP aims at inferring general rules that can assist a

designer in the transformation development. For example, *classtocube(A,B) :- class(A), cube(B), associationOwnedAttribute(A,C), property(C,float,1,1), association(D,A,E).* and *classtocube(A,B) :- class(A), cube(B), associationOwnedAttribute(A,C), property(C,integer,1,1), association(D,A,E).* are inferred rules for the ClassToCube concept.

In a *model-driven data warehouse* application, *dependencies* exist between transformations. We investigate a new machine learning methodology stemming from the application needs: *learning dependent- concepts* [130, 131]. This part of the work will answer the questions about the *search bias* problem and the best way to find concepts definition. Based on similar approaches as *layered learning* [21, 74], *context learning* [23, 75], *predicate invention* [76, 77] and *cascade learning* [78, 79], we propose a *dependent-concept learning* approach where the objective is to build a pre-order set of concepts on this dependency relationship. This pre-order is determined by the application, which is then used by the learning process: first learn non dependent concepts, then, at each step, add the learned concept as background knowledge for next concepts to be learned according to the pre-order. This DCL methodology is implemented and applied to our transformation learning problem. Experimental evaluation shows that the DCL system gives significantly better results. So, the second step is the *dependent-concepts learning* approach. It defines the search order of concepts, but also defined a way to extend concepts language. Hence, it improves search performance in terms of accuracy and rules quality. We thus offer an integrated approach covering several aspects (framework, process and research) forming a bias to support the scaling. In the proposed solution, the process part is considered as a modelling bias and the framework part as a language bias.

Chapter 4

Modelling

Model-driven data warehouse represents an approach that aligns the development of data warehousing systems with a general model-driven development paradigm. Such approaches have several advantages (adaptability, integrity, extensibility and standard development) and seem more promising. However, the architecture of model-driven warehousing is a complex architecture that uses multiple components and presents several risks. This modelling step is important to manage these risks and makes effective the task of transformations learning.

This part of work will provide answers to questions on data warehouse components derivation, standards used for design and the engineering process. But at the same time it represents a significant bias in the framework that we propose (as modelling bias or also as architecture bias). We focus, also, to find an optimized representation for the language bias in the model-driven data warehouse context. The problem of model transformation is so expressed with inductive logic programming framework. This language bias (also known as declarative bias) aims to restrict the representation to clauses that define best the transformation rules.

4.1 Motivation

The data warehousing systems are defined through a complex architecture. Current data warehouse development approaches fail to provide a unified method for standard components design and automatic derivation from conceptual models. In the light of this situation, the first goal of our work is to propose a unified approach to develop data warehouses. This will identify the elements (design metamodels, transformations, etc.) that are necessary to automate the process with machine learning. Our unified method is based on the *Model-Driven Architecture (MDA)* and the *Two Track Unified Process (2TUP)* standards. We propose a *data warehouse design framework* as a first part of the method. The *data warehouse design framework* shows the projection of the MDA viewpoints and meta-levels on the data warehousing architecture to tackle the design of every data warehouse component in an integrated and standard manner. Then, as a second part of the method, we propose a *data warehouse engineering process*. The proposed process shows the alignment of the 2TUP disciplines with the MDA transformation process. This alignment ensure the coherence between the *data warehouse design framework* and the *data warehouse engineering process* and to make our approach more integrated. It also enables an iterative and an incremental development of data warehouse layers, and to consider, at the same time, the business and the technical aspects of the data warehouse.

The main advantages of the proposed approach are: (i) it addresses, at the same time, the data warehousing design and engineering problems, indeed it integrates the *data warehouse design framework* and the *data warehouse engineering process*; (ii) it uses open industry standards (i.e., MDA and 2TUP), which facilitate its implementation and extension; (iii) it offers a mix-driven approach, in which, the data-driven, the user-driven, and the goal-driven approaches are combined with the model-driven, the semantic-driven, and the risk-driven approaches; (iv) it offers an innovative architecture to deliver *on-demand model-driven data warehouse design services* that reduce time and costs of data warehouse components implementation.

The automation of the proposed *model-driven data warehouse* approach using advisable and appropriate machine learning techniques represents our second goal. Indeed, one of the main challenges is to automatically learn these transformations from existing project traces. In this context, *model transformation by-example*

(introduced in [30]) is an active research area in model-driven software engineering that uses artificial intelligence techniques and proposes to automatically derive transformation rules. Other related research efforts on *model transformation by-example* are presented with more details [132]. The by-example approach provides assistance to designers in order to simplify the development of model transformations and it reduces complexity, costs and time of development. Thus, following this idea, the proposed model-driven method is extended by machine learning (we illustrate this with a conceptual architecture) in order to support the expert in the transformation process. We propose to express the model transformation problem as an *inductive logic programming* one [31] and to use existing project traces to find the best transformation rules.

The field of *inductive logic programming* had been under active investigation for a long time before that and several successful applications have been proposed. An overview of rule-learning (of which ILP is an instance) can be found in [83], and a substantial amount of ILP theory is presented in [94]. Those applications are chosen that specifically benefit from relational descriptions generated by ILP programs, and from ILP's ability to accommodate background knowledge in the learning process. Applications include mainly: *engineering, data mining, bioinformatics, natural language processing* and so on. Like other machine learning approaches, ILP methods take as input some examples and then construct hypotheses that best represents the underlying concepts. What distinguishes ILP from other methods is that both examples and hypotheses are represented as clause programs. The input of ILP comprises two types of logic programs, examples and background knowledge. While *background knowledge*, denoted by B, formalises existing knowledge on a problem domain, *examples*, denoted by E, describe some underlying concepts not present in background knowledge. To the best of our knowledge, this work is the only one effort that has been developed for automating *model-driven data warehouse* with *relational learning* and it is the first effort that provides experimentations in this context.

The remainder of the chapter is organized as follows. Section 4.2 presents the proposed method for *model-driven data warehouse* engineering. The *data warehouse design framework* and the *data warehouse engineering process* are explained. A *running example* is given in Section 4.3 to detail some aspects of the proposed

method. Section 4.4 presents the proposed conceptual framework for transformations learning in context of model-driven data warehousing. The transformations learning architecture is provided and an example of reduction of the UML-CWM problem to ILP is given. Finally, Section 7.4 summarizes the chapter.

4.2 Method for Model-Driven Data Warehouse

Actually, in a *model-driven data warehouse* context, several steps are needed to automatically learn the transformation rules. The first step consists in isolating steps where it is necessary to induce transformation rules and in identifying the metamodels used to define the input/output models of these transformations. This is the purpose of the designed method for data warehouse development. The approach that we propose uses several development standards. A thorough study of these standards is necessary to understand how to integrate them into the overall approach. Appendix B, provides the results of our review and analysis of standards and practices that are applicable for data warehouse development. It gives a detailed description of the *unified modelling language*, the *common warehouse metamodel*, the *query-view-transformation* and the *two track unified process*. Thus, the content of this section describes our integration efforts, our proposals and contributions to automate the development chain of data warehouses.

4.2.1 Data Warehouse Design Framework

The *data warehouse design framework* is structured on three main layers (as shown in figure 4.1) showing the use of MDA meta-levels for data warehousing components and services development. The *data warehouse design framework* is part of the proposed technical architecture (defined in next section). The *Meta-Object Facility (MOF)*, is the $M3$ meta-level of *model-driven architecture* metamodeling architecture and represents the first layer. The standards metamodels (CWM, UML, etc.) which corresponds to $M2$ meta-level, are used to define the second layer of the design framework. Then, the domain objects (or business objects) and the data warehousing services, are the $M1$ meta-level and form the third layer. The third layer contains also the MDA viewpoints (CIM, PIM, PDM, PSM, and CODE)

which are derived by transformation services using the *query-view-transformation*. Please refer to chapter 2 for a detailed description of MDA viewpoints.

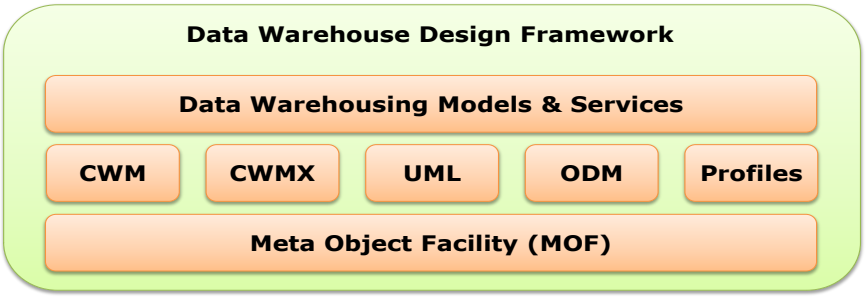

FIGURE 4.1: Data Warehouse Design Framework.

Each data warehouse component has all viewpoints of the *model-driven architecture*, and each component model in a specific viewpoint is defined by the MDA four-level architecture (i.e., the meta-levels). For example, to design the multidimensional layer we define the *multidimensional computation independent model* (Multidimensional CIM), the *multidimensional platform independent model* (Multidimensional PIM), the *multidimensional platform dependent model* (Multidimensional PDM), and the *multidimensional platform specific model* (Multidimensional PSM) viewpoints. These viewpoints correspond also to the $M1$ meta-level (multidimensional models), and are instantiated using a set of data warehouse design metamodels ($M2$ meta-level) conform to *meta-object facility* meta- metamodel ($M3$ meta-level) as discussed before. Note that the CODE viewpoint and the instance (i.e., $M0$) meta- level are not detailed because CODE represents another form of the *platform specific model* and $M0$ is specific to every execution. In the following paragraphs, we describe in details which metamodels (or packages) are used to design the data warehousing layers (i.e., *data source, integration, multidimensional, analysis, and data access*).

The data source layer is obtained by reverse engineering and by the integration of data sources models. The *data source platform specific model* (Data Source PSM) diagram represent the logical view of data sources. So, the suppliers platforms metamodels, and the CWM Relational packages are used to design relational databases. The CWM XML package can be used to represent semi-structured and unstructured data. The Data Source PDM (*platform dependent model*) describes the platforms where the sources are deployed; therefore platforms metamodels (provided by platforms suppliers) are used to define Data Source PDM in order to

facilitate the reverse engineering process. Then, the data source PIM represents the conceptual view of the data sources; it is instantiated using CWM ObjectModel (i.e., UML). In data source layer, we need a unified schema that gives a standard representation of the data, thus offering a way to deal with the heterogeneity in the sources. Thus, the *data source computation independent model* (Data Source CIM) defines information for the semantic integration of schemas. This Data Source CIM represents the domain ontology containing matching concepts to solve heterogeneity during the generation of the integrated schemas (intentional level of integration). We propose the *Ontology Definition Metamodel (ODM)* an MDA-compliant metamodel to define this ontology.

Concerning the integration layer, we propose also the *ontology definition metamodel* to design the integration computation independent model (Integration CIM) model. The Integration CIM defines a semantic mapping between data sources and data warehouse repository to solve heterogeneity during data integration (extensional level of integration). Thus, the role of Integration CIM is to improve the transformation process and solve data quality issues. The Integration PIM represents the conceptual view of integration process. We propose the CWM Transformation package to design data transformation activities, the *CWM WarehouseProcess* package to design maintenance tasks and events. However, these metamodels are too generic to represent explicit extraction, transformation, and loading operations. Therefore, we recommend the use of CWM-profiles to simplify the *Extraction, Transformation, and Loading (ETL)* model design and to ensure meta-data interchange at the same time. The metamodels of ETL platforms such as *SQL Server Integration Services (SSIS)* and *DB2 Warehouse Manager* (with CWMX DB2WarehouseManager package) are used to design Integration PDM and to generate the Integration PSM model.

The Multidimensional CIM level represents the enterprise goals and business needs in our approach. In [133] UML profiles for i* modelling in the data warehousing domain have been presented. This i* extension is used to define a CIM and it can be easily integrated in a *model-driven architecture* approach through a set of QVT transformations in order to derive the conceptual multidimensional models. In our framework, this conceptual multidimensional model is defined through the Multidimensional PIM viewpoint, and we use CWM OLAP metamodel to instantiate it. The Multidimensional PDM is used to personalize the multidimensional

PSM and to adapt it to the context of the target platform. Platforms metamodels such as Oracle and Microsoft SQL Server are used to design the Multidimensional PDM model. The multidimensional logical view is given by Multidimensional PSM model, which is developed using the CWM Relational package.

Concerning the analysis layer, we recommend also i* framework to design Analysis CIM (goals and requirements for analysis). This layer contains customs data structures such as *Online Analytical Processing (OLAP)* data cubes and data mining models used for analysis by end-users tools. Regarding the Analysis PIM, we recommend CWM OLAP and CWM DataMining metamodels to design, respectively, data cubes and data mining components. Concerning Analysis PDM, some platforms are described in CWMX metamodel; as examples CWMX Express for the Oracle Express server, CWMX Hyperion for Hyperion OLAP platform, etc. Other platforms metamodels such as Mondrian for OLAP, SAS Data Mining for data mining can also be used to define specific platforms representations. For the Analysis PSM of the OLAP data cube, we have the choice between a relational implementation (ROLAP) and a multidimensional implementation (MOLAP). So, we propose the CWM Relational package to implement ROLAP case; and for MOLAP case we propose the CWM Multidimensional package.

The goal of data access layer is to generate specific user interfaces in order to develop reports and for information visualization. At the computation independent model level (Data Access CIM), UML metamodel (use cases package part) or other adapted profiles are used to define users requirements. Several CWM metamodels can be used to design the Data Access PIM. In our approach, we propose the CWMX InformationReporting package for reports construction and CWM InformationVisualization package to support information visualization. UML class diagram metamodel is also used with these metamodels to design applicative and Graphical User Interface layers at Data Access PIM level. *Java Enterprise Edition (JEE) and Microsoft .NET* environments are, in general, used to generate these tools, so we adopt them to define the Data Access PDM and the Data Access PSM levels.

The main advantages of the proposed *data warehouse design framework* are: (i) Completeness: since, our framework covers the design of all data warehousing components, and integrates all MDA viewpoints and meta-levels. (ii) Extensibility: in our approach the metamodeling architecture using MOF simplifies the extension of the framework through metamodeling techniques. (iii) Standard development:

indeed, all data warehousing layers design diagrams are identified using a standard metamodels for data warehouse development such as CWM, CWMX, ODM, etc.

4.2.2 Data Warehouse Engineering Process

The MDA transformation process does not include all engineering disciplines such as preliminary study and tests. Moreover, result of *model-driven architecture* process is a semi-complete system code. Thus, it is necessary to define a code completion activity in the global development process. Also, the transformation process of MDA is not an iterative and incremental process. To overcome these limits, we propose to use the *two tracks unified process* process in order to develop data warehouse components. We keep also the MDA approach in order to ensure coherence between the *data warehouse design framework* and the *data warehouse engineering process*. Therefore, in our *data warehouse engineering process*, the MDA transformation process is a sub-process. Figure 4.2 shows the disciplines and the iterations applied to develop the components of a one layer of the data warehousing architecture. It represents, also, the mapping between the 2TUP activities and the MDA transformation process steps. Then, we will adapt this process for the development of all data warehousing layers in order to define the *data warehouse engineering process*. Thus, in our process, the 2TUP activities are adapted to enable the transformation process of MDA using QVT and to take into account the data warehouse development constraints.

The *data warehouse engineering process* starts by the preliminary study activity. The preliminary study contains a study of the enterprise business process to collect business information's, identifying preliminary requirements, and a study of market platforms to prepare the technical requirements modelling. Each layer of the data warehouse is developed using a MDA transformation process starting by the definition of the layer BCIM (Business CIM), using the common TCIM (Technical CIM) and ends with components code generation. So, the MDA process is repeated for the construction of each data warehouse layer. If several components are defined in a layer, then several 2TUP iterations may be applied (for example, runs iteration per component). The final *data warehouse engineering process* showing the development of all data warehousing layers is given by figure 4.3. In the

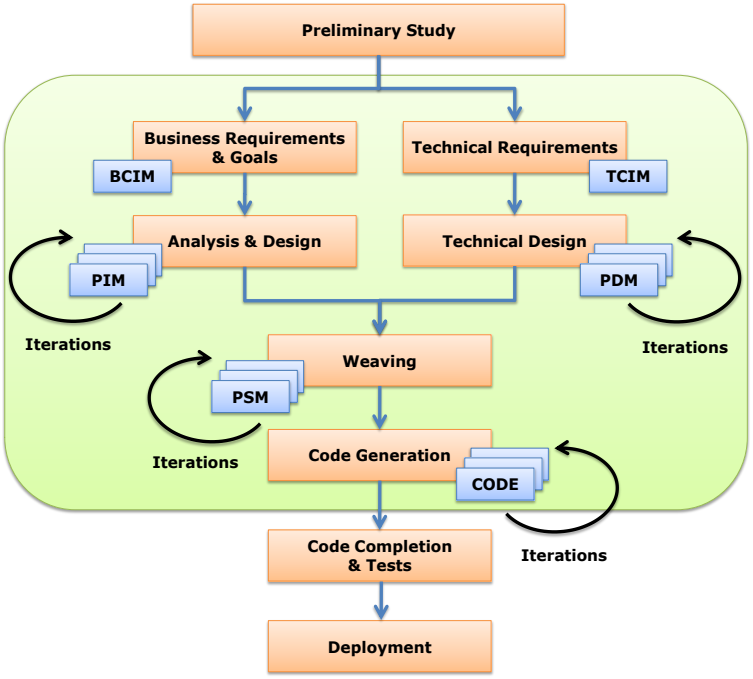

FIGURE 4.2: The MDA/2TUP Integration.

following paragraphs, we describe in more details the execution of the proposed *data warehouse engineering process*.

The *model-driven warehousing architecture* process starts by the development of the data source layer. A reverse engineering process is applied since data source code to obtain the data source logical model (Data Source PSM). Then the reverse engineering process is applied on Data Source PSM using information's given by data source PDM and Data Source CIM, to obtain the data source conceptual model (Data Source PIM). The Data Source PDM is used to solve the technical errors related to reverse process and the Data Source CIM is used to solve the problems related to the semantic heterogeneity of sources. The next step of the proposed *data warehouse engineering process* is the derivation of the multidimensional components (data warehouse repository and/or data marts). The Data Source PIM and the Multidimensional CIM are merged using a QVT transformation to generate the multidimensional conceptual model (Multidimensional PIM). The Multidimensional PIM is instance of CWM OLAP metamodel already defined in our *data warehouse design framework*. Then, the multidimensional logical model (Multidimensional PSM) and CODE (i.e., code of the multidimensional schemas) are generated for a specific platform using the description given by the multidimensional platform

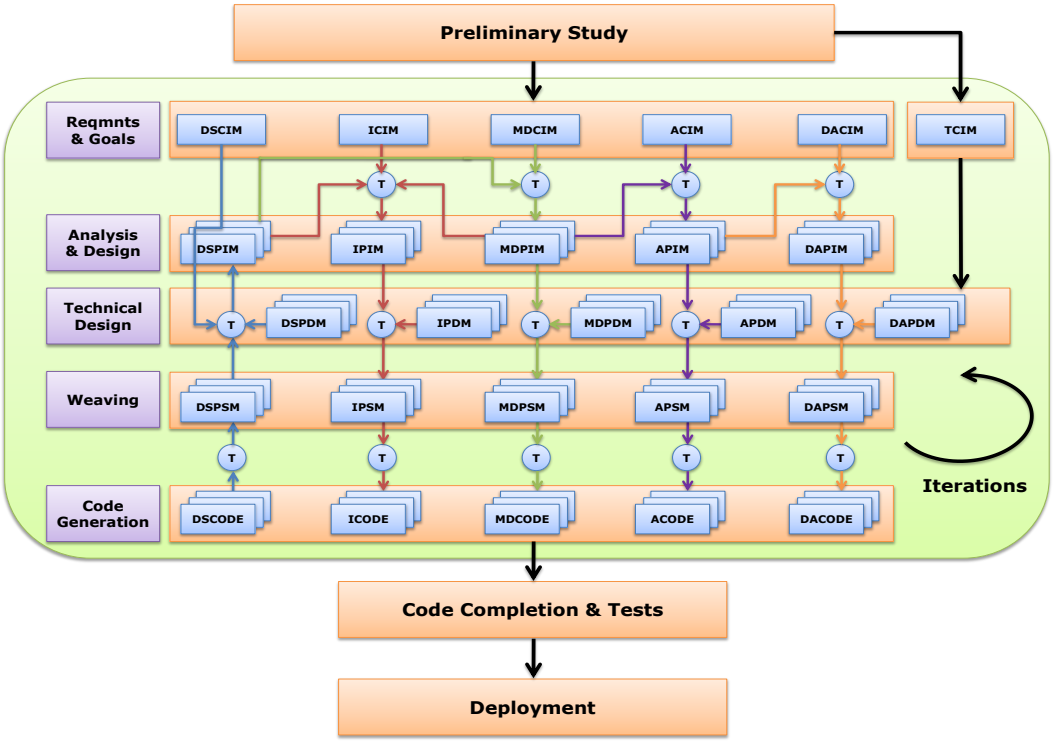

FIGURE 4.3: Data Warehouse Engineering Process.

dependent model (Multidimensional PDM).

The development of the conceptual ETL jobs (Integration PIM) requires three models: the Integration CIM (semantic transformation rules), the Data Source PIM (i.e., the source-model), and the Multidimensional PIM (i.e., the target-model). The Data Source PIM and the Multidimensional PIM are used to define mapping between sources and the data warehouse repository (or data marts). The Integration CIM is used to solve problems related on heterogeneous data and to make semantic data cleansing, correction, and integration. The Integration PSM and Integration CODE (i.e., XML files in general) describing the mapping and transformations are finally generated for a specific ETL tool. The fourth step of the *data warehouse engineering process* is the customization of analysis models. So, a set of OLAP cubes and data mining models (Analysis PIM) can then derived from the multidimensional layer according to business requirements and goals defined in the Analysis CIM model. The *data warehouse engineering process* ends with the derivation of the data access layer (the fifth step). The data access tools and reporting interfaces are developed using the analysis layer models (Analysis PIM) and user requirements provided by the Data Access CIM. The MDA transformation

process using QVT is also applied to generate the Data Access PIM and Data Access PSM models. End-users applications are generated for a specific framework (JEE or .NET) to obtain the Data Access CODE.

The main advantages of the proposed *data warehouse engineering process* are: (i) the whole data warehouse development is tackled in an iterative and incremental way; (ii) it is component oriented, offering flexibility to the model and supporting the re-use; (iii) it allows a better technical risk management and thus constitutes the deadlines and the costs control; (iv) the MDA transformation process is fully integrated in the global engineering process, which includes additional disciplines to improve quality.

4.3 Example of Application

In this section, we present some results of the proposed method, extracted from a first experimental prototype. We study the generation process of the integration layer, and we focus on the *Integration Platform Specific Model (IPSM)* derivation. As it is already defined in our *data warehouse engineering process*, the IPSM is derived during the weaving activity using the *Integration Platform Independent Model (IPIM)* and the *Integration Platform Dependent Model (IPDM)*. We choose *Microsoft SQL Server Integration Services (SSIS)* [134] as target ETL platform. The SSIS is a new version of *Data Transformation Services (DTS)* in SQL Server 2000. Thus, one of our contributions presented in this section is the extension of the CWM Transformation package with a *Platform Dependent Model (PDM)* for SSIS. As a transformation process example for SSIS, we use the tutorial given in [135].

The objective of this case study is to develop an ETL package that extracts and transforms currency data contained in a flat file source. These data are then integrated into a fact table called FactCurrecyRate. The source is a set of historical currency data, which has the following four columns: the average rate of the currency, a currency key, a date key, and the end-of-day rate. The FactCurrencyRate table (fact table) has also four columns: the average rate, the currency key, the time key, and the end-of-day rate. The fact table has relationships to two dimension tables (Currency and Time dimensions). These relationships

indicate that lookups will be necessary to integrate the currency key and time key values. This case study constitutes a first step to verify the approach feasibility and to define which model is generated in each step of the transformation process.

```
<CWM:Classifier xmi.id="_1.1" name="CurrencyDataFile" visibility="public" namespace="_1">
    <CWM:Classifier.feature>
        <CWM:Attribute xmi.id="_1.1.1" name="AverageRate" owner="_1.1" type="_3" />
        <CWM:Attribute xmi.id="_1.1.2" name="CurrencyID" owner="_1.1" type="_5" />
        <CWM:Attribute xmi.id="_1.1.3" name="CurrencyDate" owner="_1.1" type="_6" />
        <CWM:Attribute xmi.id="_1.1.4" name="EndOfDayRate" owner="_1.1" type="_3" />
    </CWM:Classifier.feature>
</CWM:Classifier>
```

FIGURE 4.4: Part of the Data Source Platform Independent Model.

For model interchange during the transformation process, we use the *XML Metadata Interchange (XMI)*. XMI is a standard language for models description and metadata serialization. The XMI is an *object management group* standard mechanism for the stream-based interchange of MOF-compliant metamodels. XMI is essentially a mapping of the *W3C's eXtensible Markup Language (XML)* to the *meta-object-facility*. Figures 4.4 and 4.5 show a part of the conceptual model of the data source (DSPIM) and the conceptual multidimensional target (MDPIM). These models are instantiated using the corresponding CWM packages (i.e., CWM ObjectModel and CWM OLAP) defined in Section 4.2.1. Note that to express some semantic aspects of the models during transformation, we use the TaggedValue element (defined in CWM ObjectModel metamodel). For example, it used to define table type (case of the FactCurrencyRate in figure 4.5) or to define transformation type in the conceptual ETL model. This allows for more comprehensive models and reduces transformations errors. A part of the derived *Integration Platform Independent Model (IPIM)* using CWM Transformation package is given in figure 4.6.

The *SQL Server Integration Services* tasks are organized on two categories: (i) the control flow components containing tasks for integration process control, precedence constraints and events definition; (ii) the data flow components responsible for data source and destination adapters, and data transformation elements. The *Integration Platform Dependent Model (IPDM)* for SSIS is defined by an extension of CWM Transformation package. This IPDM is made according to the definition

```
<CWMOLAP:Cube xmi.id="_7.3" name="FactCurrencyRate" visibility="public" schema="_7">
    <CWM:ModelElement.taggedValue>
        <CWM:TaggedValue tag="Type" value="FactTable" />
    </CWM:ModelElement.taggedValue>
    <CWM:Classifier.feature>
        <CWM:Attribute xmi.id="_7.3.1" name="CurrencyKey" owner="_7.3" type="_4" />
        <CWM:Attribute xmi.id="_7.3.2" name="TimeKey" owner="_7.3" type="_4" />
        <CWM:Attribute xmi.id="_7.3.3" name="AverageRate" owner="_7.3" type="_3" />
        <CWM:Attribute xmi.id="_7.3.4" name="EndOfDayRate" owner="_7.3" type="_3" />
    </CWM:Classifier.feature>
</CWMOLAP:Cube>
```

FIGURE 4.5: Part of the Multidimensional Platform Independent Model.

of *SQL Server Integration Services* components and the specification of CWM Transformation metamodel [136].

```
<CWMTFM:TransformationActivity xmi.id="_8.1" name="ExtractSampleCurrencyData">
    <CWM:Namespace.ownedElement>
        <CWMTFM:TransformationStep xmi.id="_8.1.1" name="ExtractSampleCurrencyData" task="_8.2"
            succeedingStep="_8.1.2" />
        <CWMTFM:TransformationStep xmi.id="_8.1.2" name="LookupCurrencyKey" task="_8.3"
            precedingStep="_8.1.1" succeedingStep="_8.1.3" />
        <CWMTFM:TransformationStep xmi.id="_8.1.3" name="LookupDateKey" task="_8.4"
            precedingStep="_8.1.2" succeedingStep="_8.1.4" />
        <CWMTFM:TransformationStep xmi.id="_8.1.4" name="SampleOLEDBDestination" task="_8.5"
            precedingStep="_8.1.3" />
    </CWM:Namespace.ownedElement>
</CWMTFM:TransformationActivity>
```

FIGURE 4.6: Part of the Integration Platform Independent Model.

The Control Flow components extend the TransformationActivity element (figures 4.7 and 4.8); the PrecedenceConstraint from *SQL Server Integration Services* extends PrecedenceConstraint class from CWM Transformation with custom attributes (figure 4.9); and the data flow components (such as FlatFileSource and Lookup) extend the TransformationTask element.

During the *weaving* operation, the *Integration Platform Specific Model (IPSM)* for SSIS is generated. The platform independent transformations defined in the IPIM are derived into the IPSM using the target platform description (i.e., the IPDM). The resulting ETL code files (i.e., the integration code - ICODE) represents another level of the IPSM model. For SSIS case, the ICODE corresponds to a ".dtsx"

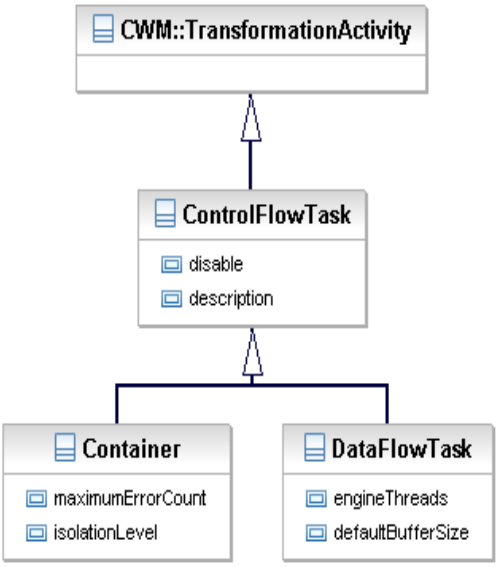

FIGURE 4.7: SSIS Control Flow Components Extension.

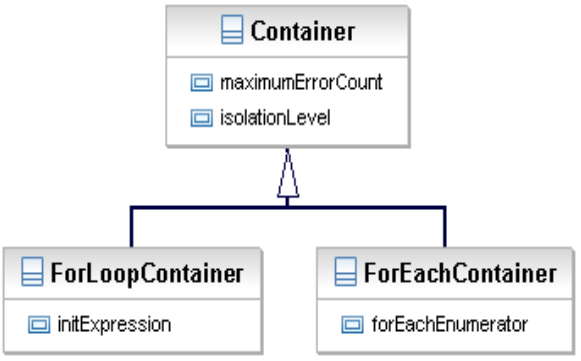

FIGURE 4.8: The Container Component Specialization.

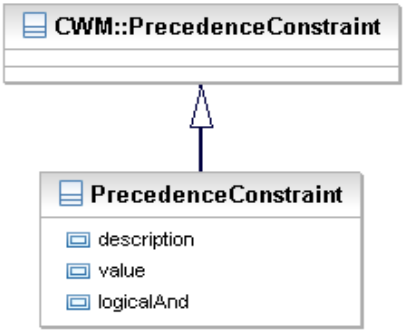

FIGURE 4.9: PrecedenceConstraint Extension.

file (SSIS package extension) which can be imported into *SQL Server Business Intelligence Development Studio* for code correction and completion. Figure 4.10 shows the control flow part (Data Flow task) of the IPSM package. The SSIS Data Flow task provides a data flow engine to extract data from sources, transform and load data into destinations. Thus, figure 4.11 shows the corresponding data flow transformations part of the derived IPSM.

FIGURE 4.10: Data Flow Task of IPSM.

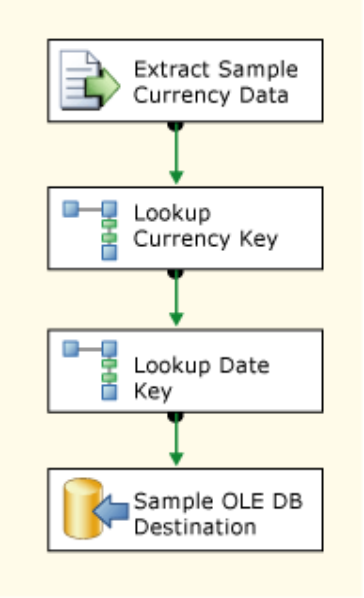

FIGURE 4.11: Data Flow Transformations of IPSM.

4.4 Framework for Transformations Learning

Transformations development is a very hard task, because programmers must have serious skills with the corresponding metamodels and the transformation languages. This makes the approach more complex and entails additional costs, especially

in the case of a data warehousing project. The aim of our research is to align model transformation by-example approach to automate the *model-driven data warehouse* development. The data warehouse design metamodels present a different context from UML/ER metamodels alignment widely studied by related works. So, this makes our work more interesting and challenging. In this section, we start by presenting the context of model transformation and the architecture of a conceptual framework for transformations learning. Then, we show how the UML-CWM transformation problem is defined in the *inductive logic programming* language.

4.4.1 Transformations Learning Architecture

The *query-view-transformation* is proposed by the *object management group* [29] as standard language for model transformation. Figure 4.12 shows the operational context of the QVT. A source-model Ma is transformed into a target-model Mb according to a transformation definition T. The transformation definition T written in any QVT implementation or language [64, 129]. The source Ma and the target Mb models are conforms to their respective MMa and MMb metamodels. The transformation definition is a model conforming to the *query-view-transformation* metamodel. All metamodels conform to the *meta- object facility* meta-metamodel.

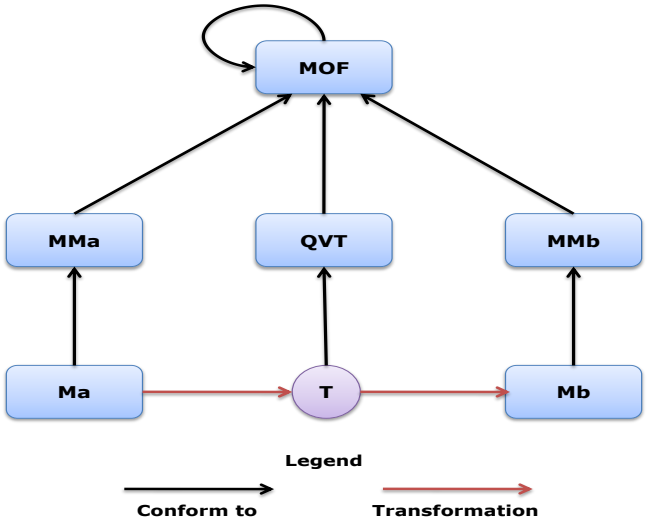

FIGURE 4.12: The Query-View-Transformation Setting.

The *QVT Relations (QVT-R)* specification is a declarative language that we use for model-to-model transformation and it is an example of relational approach. Figure 4.13 shows the ClassToCube relation as example. The left part defines the source-model context (or source meta-elements), denote $\mathbb{P}(MM)$, where MM is the source meta-model. The rigth part defines the target-model context (or resulted target meta-elements), denote $\mathbb{P}(MN)$, where MN is the target meta-model. $\mathbb{P}(MM)$ and $\mathbb{P}(MN)$ define respectively input and output patterns of the transformation rules specification and they are used in the next chapter for the formal definition of a model transformation. Following the QVT specification, a relation in QVT-R framework is defined by the following three elements:

- ***Two or more domains:*** each domain is a distinguished set of elements of a candidate model (source or target model). This set of elements must be matched in that model by means of patterns.
- ***When clause:*** it defines the conditions under which the relation applies (i.e., a precondition) and it usually corresponds to the execution of a parent-relation.
- ***Where clause:*** it specifies additional constraint and conditions that must be satisfied by all model elements participating in the relation (i.e., a post-condition).

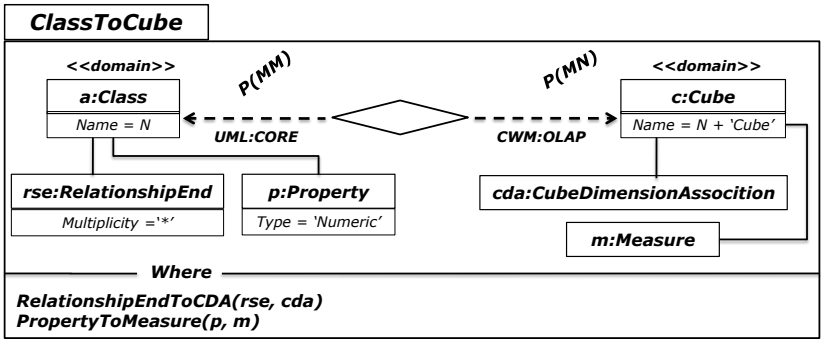

FIGURE 4.13: Graphical Notation of ClassToCube Relation.

The question of *how to provide an integrated framework that automatically learns transformations from previous data warehousing projects (or experiences)* is addressed in our previous work. We illustrate this by a conceptual architecture

(figure 4.14). We focus on the *DSPIM (Data-Source PIM)* to *MDPIM (Multidimensional PIM)* transformations learning. The DSPIM (source model) represents a conceptual view of a data-source repository and the MDPIM (target model) represents a conceptual view of a data warehouse repository. We have proposed the UML CORE and the CWM OLAP metamodels to design respectively the source and the target models. The objective is (1) to provide a set of model transformation examples at the model level (the model layer of MDA, referred to as $M1$, defines elements that describe semantic domains); then (2) to run a learning engine that will build rules at the metamodel level (the metamodel layer of the *model-driven architecture*, referred to as $M2$, is responsible of defining a language for specifying models).

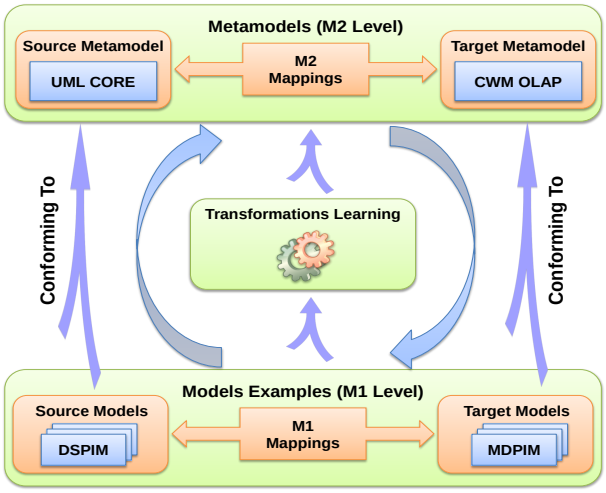

FIGURE 4.14: Conceptual Architecture for Transformations Learning.

The *transformations learning* component is the core module of the framework. We will see later that represents the *inductive logic programming* engine. The input for transformation learning is a set of interconnected source and target model pairs (learning examples). These examples of models represent the company projects trace. So, they are designed in the $M1$ level of the metamodeling architecture of the *model-driven architecture* and they are conforming to their respective metamodels. In general, a pre-treatment step is necessary to ensure that project traces are full conforming to the *unified modelling language* and the *common warehouse metamodel* metamodels. The aim of the framework is to generate transformation rules between the concepts of metamodels. Consequently, these transformation rules define mapping at the metamodel level (i.e., the $M2$) mapping). As the transformation design is an essential task during the model-driven engineering

process, our work is focused on this task. We will focus our application on the single transformation from UML metamodel to CWM metamodel. First, we introduce the two main models (UML CORE and CWM OLAP) concerned with this transformation (please refer to Appendix C for this). In the next sub-section, we give a project trace transformation showing a mapping example between the source and target instances. Then, we will show how to reduce this problem as an inductive logic program.

4.4.2 Reduction of the UML-CWM Problem to ILP

The above discussion of ILP (i.e., chapter 2) assumes the existence of a hypothesis language. Hypothesis language limits the set of clauses to be considered for a learning task and therefore reduces the complexity of the learning process. Most ILP systems allow users to explicitly specify the hypothesis language using some form of *declarative bias*. Declarative bias, also known as *language bias*, often applies a set of syntactic constraints on the hypothesis language. In this section, we mainly focus on how to find an optimized representation of assumptions, and so the language bias in the *model-driven data warehouse* framework. This language bias aims to restrict the representation to clauses that define best the transformation rules. The proposed representation allows for an improved learning process and also for a good quality of the obtained rules. This will facilitate the task of understanding and validation of the learned rules.

We define the language bias using the metamodel level ($M2$ level) of the metamodelling architecture. This gives the advantage to define a clear set of predicates with an optimal level of abstraction. Predicates obtained from the $M2$ level will ensure obtaining understandable transformation rules, equivalent to transformation designed manually. In the proposed *model-driven data warehouse* framework, this process will extract predicates from UML CORE and CWM OLAP metamodels. UML CORE defines the predicates used for the representation source-models examples (denoted as DSPIMs in figure 4.15). CWM OLAP defines the predicates used for representing target-models examples (denoted as MDPIMs in figure 4.15). So, according to UML CORE and CWM OLAP metamodels, we define the following predicates to describe, respectively, source models (i.e., the DSPIMs) and target models (i.e., the MDPIMs).

Firstly, a few common predicates required for instantiating source and target models are defined as:

```
type(Name). multiplicity(Bound).
isTime(Boolean). isMeasure(Boolean).
```

The defined predicates from UML CORE are:

```
class(Name).  property(Name, Type, Lower, Upper).
association(Name, Source, Target).
associationOwnedAttribute(Class, Property).
associationMemberEnds(Association, Property).
```

Then, the defined predicates from the CWM OLAP are:

```
cube(Name).  measure(Name, Type, Cube).
dimension(Name, isTime, isMeasure).
cubeDimensionAssociation(Cube, Dimension).
level(Name).  levelBasedHierarchy(Name, Dimension).
hierarchyLevelAssociation(LevelBasedHierarchy, Level).
```

As an example of trace model, let us consider a data schema used to manage customers and invoices (figure 4.15). With respect to the defined UML CORE and CWM OLAP predicates, the following code is part of the generated background knowledge program of DSPIM *(Data Source Platform Independent Model)* and MDPIM *(Multidimensional Platform Independent Model)*. First, we need a set of common type definitions, as the following, that will be used in the source and target predicates:

```
type(integer). type(float). type(string). type(date).
multiplicity(0). multiplicity(1). multiplicity(*).
isTime(true). isTime(false). isMeasure(true). isMeasure(false).
```

Then, a part of the DSPIM definition is as the following:

```
class(invoice). class(customer). class(category).
property(amount, float, 1, 1).
associationOwnedAttribute(invoice, amount).
association(invoice-customer, invoice, customer).
property(cInv, null, 0, *). property(iCust, null, 1, 1).
associationMemberEnds(invoice-customer, cInv).
associationMemberEnds(invoice-customer, iCust).
```

Finally, a part of the MDPIM definition will be:

```
cube(invoiceFact). measure(amount, float, invoiceFact).
dimension(customerDim,false,false). dimension(timeDim,true,false).
cubeDimensionAssociation(invoiceFact, customerDim).
level(customerLev). level(sellerLev).
levelBasedHierarchy(customerDimH1, customerDim).
hierarchyLevelAssociation(customerDimH1, customerLev).
hierarchyLevelAssociation(customerDimH1, sellerLev).
```

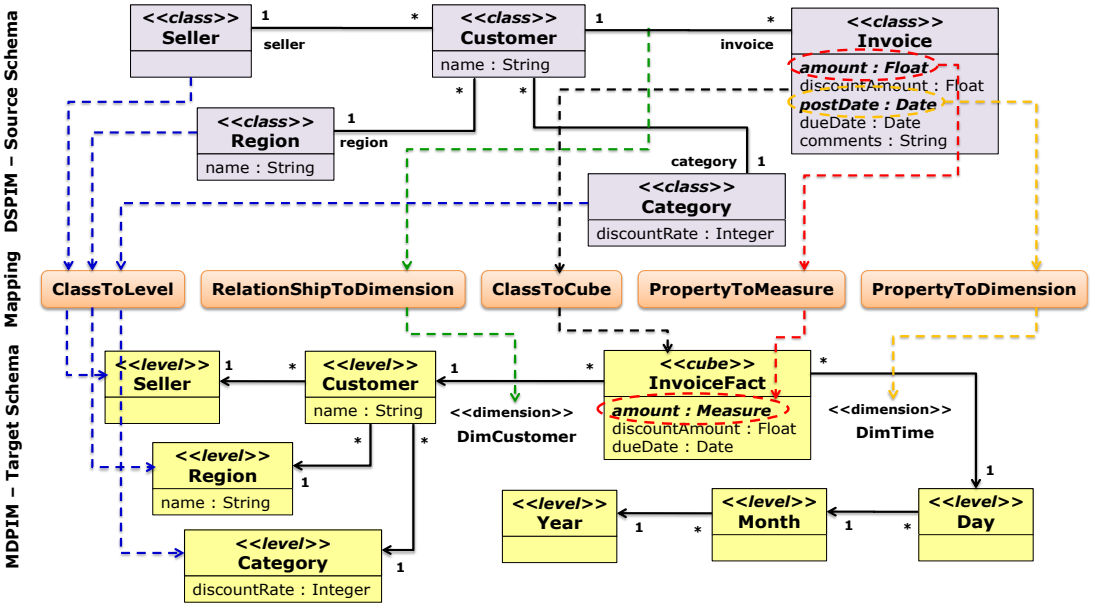

FIGURE 4.15: Mapping UML CORE Instance to CWM OLAP Instance.

We introduce the two widely-used forms of language bias, namely *mode declarations* (that are part of the background knowledge). Mode declarations were first

introduced as a form of declarative bias in [137]. ILP systems that implement mode declarations include Warmr [138], Foil [98], Tilde [101], Progol [100] and Aleph [97]. As the actual syntax varies from one system to another, we give in this section the mode language supported by Aleph. These declare the mode of call for predicates that can appear in any clause hypothesised by Aleph. The Modes indicates the predicate format, and can be described as: *predicate(ModeType1, ModeType2, ... , ModeTypen)*. The declaration *modeh* indicates the predicate that will compose the head of the rules. The *modeb* declaration indicates that the predicate can be part of the body of the generated rules. We refer readers to Appendix D for a brief review of Aleph syntax. For example, modes declaration of ClassToCube and RelationShipToDimension concepts takes the form:

```
:- modeh(1,classtocube(+class,+cube)).
:- modeb(*,class(+class)).
:- modeb(*,cube(+cube)).
:- modeb(*,property(+property,#type,#multiplicity,#multiplicity)).
:- modeb(*,association(-association,+class,-class)).
:- modeb(*,associationOwnedAttribute(+class,-property)).
:- modeb(*,associationMemberEnds(+association,-property)).

:- modeh(1,relationshiptodimension(+association,+dimension)).
:- modeb(*,class(+class)).
:- modeb(*,cube(+cube)).
:- modeb(*,dimension(+dimension,#isTime,#isMeasure)).
:- modeb(*,classtocube(+class,+cube)).
:- modeb(*,association(+association,-class,-class)).
:- modeb(*,property(+property,#type,#multiplicity,#multiplicity)).
:- modeb(*,associationMemberEnds(+association,-property)).
```

The *determinations* are also used by Aleph as another form of search directives. The *determination* statements declare the predicated that can be used to construct hypotheses. They are part of the language and search restrictions for Aleph. Determination statements declare the predicated that can be used to construct a hypothesis. For ClassToCube and RelationShipToDimension concepts, they take the form:

```
:- determination(classtocube/2,class/1).
:- determination(classtocube/2,cube/1).
:- determination(classtocube/2,associationOwnedAttribute/2).
:- determination(classtocube/2,associationMemberEnds/2).
:- determination(classtocube/2,association/3).
:- determination(classtocube/2,property/4).

:- determination(relationshiptodimension/2,class/1).
:- determination(relationshiptodimension/2,cube/1).
:- determination(relationshiptodimension/2,cube/1).
:- determination(relationshiptodimension/2,classtocube/2).
:- determination(relationshiptodimension/2,associationMemberEnds/2).
:- determination(relationshiptodimension/2,association/3).
:- determination(relationshiptodimension/2,dimension/3).
:- determination(relationshiptodimension/2,property/4).
```

The *mapping model* describes all transformations of the source elements to the target elements. The transformation predicates are of the form: *transformation(SourceElement, TargetElement)* where SourceElement and TargetElement represent, respectively, input and output of the transformation rule. A transformation rule expresses in which conditions or in which context an element SourceElement in the source model is transformed into a TargetElement in the target model. For instance, the question is: in which context a *class* in the UML CORE model can be translated to a *cube* in the CWM OLAP model. Given project traces, we extract the situation where the Invoice class is translated into a cube InvoiceFact, each such situation defines a positive example. Similarly, the situation where a *class* is not transformed into a *cube* defines a negative example. Then, Aleph is run on this training set. In the example of figure 4.15, *classtocube(invoice,invoiceFact)* is a positive example and *classtocube(customer,invoiceFact)*, *classtocube(seller,invoiceFact)*, *classtocube(region,invoiceFact)* are negative examples. This reduction of the UML-CWM problem to ILP aims at inferring general rules that can assist a designer in the transformation development. For example, *classtocube(A,B) :- class(A), cube(B), associationOwnedAttribute(A,C), property(C,float,1,1), association(D,A,E).* and *classtocube(A,B) :- class(A), cube(B), associationOwnedAttribute(A,C), property(C,integer,1,1), association(D,A,E).* are inferred rules for the ClassToCube concept.

4.5 Summary

In this chapter, we have contributed to improve current *model-driven data warehouse* approaches, by providing a complete and an integrated method. Our unified method for data warehouses development integrates at the same time the *data warehouse design framework* and the *data warehouse engineering process*. We started by presenting the *data warehouse design framework* based on the *model-driven architecture* concepts. Then, the *data warehouse engineering process* based on the *2 tracks unified process* is described. The proposed method for data warehouses development is a *unified and mix-driven approach*, which includes, at the same time: the *data-driven*, the *user-driven*, the *goal-driven*, the *model-driven*, the *semantic-driven*, and the *risk-driven* approaches. Indeed, the reverse engineering and the schemas integration of data sources make our approach data-driven at this layer. The enterprise goals and business needs are defined through the MDCIM *(Multidimensional CIM)* and the ACIM *(Analysis CIM)*, thus they provide a goal-driven development for these layers. End-users requirements (i.e., within the DACIM) are considered to developing the Data Access layer, so we consider the development process of this layer as a user- driven. The DSCIM *(Data Source CIM)* and the ICIM *(Integration CIM)* models define two ontologies, which are modelled using the *ontology definition metamodel* in order to solve schemas integration and data integration problems. Therefore, the development of the Data Source and the Integration layers follow also a semantic -driven process.

The main contributions of the work presented in this chapter are: (i) a definition of current data warehousing and business intelligence problem statement including data warehouses components construction and business intelligence applications deployment; (ii) a proposal for a unified and a mix-driven data warehouse development approach; (iii) introduction of a machine learning architecture based on the *inductive logic programming* to support the proposed *model-driven data warehouse* approach by addressing the problem of UML-CWM transformation problem.

The *model transformation by-example* [65, 66, 139–141] constitutes a solution to derive transformations in model-driven software engineering environments. We rely primarily on the conceptual *model transformation by-example* framework presented by [63, 142]. Then, we propose an adapted version for *model-driven data warehouse* context. This architecture will help us to apply learning and define

what specific learning approach to use. The learned hypotheses can be easily incorporated into later learning tasks. Noting that hypotheses take the same form as background knowledge in the *inductive logic programming* , it is straightforward to use previous learned hypotheses as background knowledge in the same fashion as human knowledge for related learning tasks. So, the learned hypotheses can also enrich our knowledge on the domain or they can be used in a second stage of the learning process. This idea is applied in the proposed *dependent-concept learning* approach. This problem is addressed and formalized in the next chapter. Then, the evaluation of the *dependent-concept learning* method is provided in Chapter 6. Finally, in Chapter 7, we will introduce a multi-layered architecture of a tool supporting the proposed *model-driven warehousing* approach.

Chapter 5

Learning Approach

Model transformation in the context of model-driven data warehouse is ensured by human experts. It generates an exorbitant cost and requires high proficiency. We have proposed an optimized model-driven approach and a machine learning framework to reduce the expert contribution in the transformation process. We propose to express the model transformation problem as an inductive logic programming one and to use existing project traces to find the best business transformation rules. We used the Aleph ILP system to learn such rules. Obtained results show that found rules are close to expert ones.

Within our metamodeling context, we need to deal with several dependent concepts. In fact, in a model-driven data warehouse application, dependencies exist between transformations. We investigate a new machine learning methodology stemming from the application needs: learning dependent-concepts. Following work about layered learning [21, 74], context learning [23, 75], predicate invention [76, 77] and cascade learning [78, 79], we propose a new methodology that automatically updates the background knowledge of the concepts to be learned. The evaluation shows that the dependent-concepts learning approach is better to deal with the model-driven problem.

5.1 Motivation

The *model-driven data warehouse* gathers approaches that align the development of the data warehouse with a general model-driven engineering paradigm [17]. The *model-driven engineering* is mainly based on models, meta-models and transformation design. Indeed, model-driven strategy encourages the use of models as a central element of development. The models are conforming to metamodels and the transformation rules are applied to refine them. Therefore, transformations are the central components of the each model-driven process. However, transformation development is a very hard task that makes the model-driven approach more complex and entails additional costs. So, designers or programmers must have high skills in the corresponding metamodels and the transformation languages, e.g., the *Query-View-Transformation (QVT)*. In addition, data warehousing projects require more knowledge about the underlying business domain and requirements. This raises many risks and challenges during the transformations design. One of the main challenges is to automatically learn these transformations from existing project traces. In this context, *model transformation by-example* (introduced by [30]) is an active research area in model-driven software engineering that uses artificial intelligence techniques and proposes to automatically derive transformation rules. It provides assistance to designers in order to simplify the development of model transformations and it reduces complexity, costs and time of development.

In the context of *model-driven data warehouse*, several steps are needed to automatically learn the transformation rules. The first step, which has been addressed in the previous chapter, consists in isolating steps where it is necessary to induce transformation rules and in identifying the metamodels used to define the input/ output models of these transformations. The proposed method is extended by machine learning in order to support the expert in the transformation process. We propose to express the model transformation problem as an *Inductive Logic Programming (ILP)* one [31] and to use existing project traces as a source of examples to learn the best transformation rules. To the best of our knowledge, this work is the only one effort that has been developed for automating *model-driven data warehousing* with relational learning and it is the first effort that provides experimentations in this context.

We use the Srinivasan's *Aleph (A Learning Engine for Proposing Hypotheses)* [97] as ILP system. Aleph uses the Progol algorithm, proposed by [143], to learn rules described as Prolog programs. The Aleph has a powerful representation language that allows representing complex expressions and simultaneously incorporating new background knowledge easily. Aleph also let choose the order of generation of the rules, change the evaluation function and the search order. More information about the most common settings allowed for the Aleph system can be found in the Appendix D. The basic algorithm of Aleph follows procedure that can be described in 4 steps:

1. ***Select:*** Select a positive example $e+ \in E^+$ to be generalised. If none exist ($E^+ = \emptyset$), stop.
2. ***Saturation:*** Construct the most-specific-clause (i.e., the bottom clause) $\perp$ that entails the selected example, and is within language restrictions provided.
3. ***Reduction:*** Search for a more general clause than the bottom clause. This is done by searching for some subset of the literals in the bottom clause that has the "best" score.
4. ***Cover Removal:*** The clause with the best score is added to the current theory, remove all redundant examples from E^+. Return to Step 1.

The problem of learning more than one predicate together is known as *multi-predicate learning* [144]. It deal with the case in which the definition of a certain predicate p is function of another predicate q that has to be learned (the case extends naturally to more than two predicates). In the model-driven data warehousing problem, a predicate is defined for each model-element (e.g., class, attribute, etc.) to transform *dependencies* exist between transformations. The question is, what is the best order to choose to learn these rules of transformation? So, we investigate a new machine learning methodology stemming from the application needs: learning dependent-concepts. Following work about *layered learning* [21, 74], *context learning* [23, 75], *predicate invention* [76, 77] and *cascade learning* [78, 79], we propose a *Dependent-Concept Learning (DCL)* approach where the objective is to build a pre-order set of concepts on this dependency relationship: first learn non dependent concepts, then, at each step, add the learned concept as background

knowledge for next concepts to be learned according to the pre-order. This DCL methodology is implemented and applied to our transformation learning problem. Experimental evaluation shows that the DCL system gives significantly better results.

This chapter is organised as follows: Section 5.2 provide a formalisation of main background concepts of the model-driven domain. Section 5.3 details the used machine learning algorithms and introduces the *dependent-concept learning* approach. Section 7.4 summarizes the chapter and gives our conclusions.

5.2 Key Concepts Formalisation

In this section we provide background definitions and concepts useful for understanding the application domain. A formal definition is also provided in order to be used next in the definition of the transformation learning approach. First, we define the notion of *model*. Then, we recall the definition of a *metamodel* and the relation between models and metamodels. Finally, the definition of a *model transformation* is given.

Definition 5.1. (**Model**) A model $M = (G, MM, \mu)$ is a tuple where: $G = (N_G, E_G, \Gamma_G)$ is a directed multi-graph[1], MM is itself a model called the reference model of M (i.e., its metamodel) associated to a graph $G_{MM} = (N_{MM}, E_{MM}, \Gamma_{MM})$, and $\mu : N_G \cup E_G \rightarrow N_{MM}$ is a function associating elements (nodes and edges) of G to nodes of G_{MM}.

The relation between a model and its reference model (metamodel) is called *conformance* and is noted *conformsTo*. Elements of MM are called meta-elements (or meta-concepts). μ is neither injective (several model elements may be associated to the same meta-element) nor surjective (not all meta-elements need to be associated to a model element). The relation between elements and meta-elements is an instantiation relation. For example, the Invoice (respectively InvoiceFact) element in a DSPIM (MDPIM) is an instance of Class (respectively Cube) meta-class in the

[1] A directed multi-graph $G = (N_G, E_G, \Gamma_G)$ consists of a finite set of nodes N_G, a finite set of edges E_G, and a function $\Gamma_G : E_G \rightarrow N_G \times N_G$ mapping edges to their source and target nodes [145].

UML CORE (CWM OLAP) metamodel. So, regarding this relation, the structure of Invoice (InvoiceFact) is *conformsTo* Class (Cube) specification.

Definition 5.2. (**Metamodel and Meta-Metamodel**) A meta-metamodel is a model that is its own reference model (i.e., it conforms to itself). A metamodel is a model such that its reference model is a meta-metamodel [145].

The metamodeling architecture (part of the *model-driven architecture* international standard) is based on meta-levels: M_3, M_2, M_1 and M_0. M_3 is the meta-metamodel level and it forms the foundation of the metamodeling hierarchy (the *meta-object-facility* is an example of meta-metamodel). M_2 consists of the metamodel level (the *unified modelling language* and the *common warehouse metamodel* are examples of metamodels). M_1 regroups all user-defined models and M_0 represents the runtime instances of models.

Authors in [146] provide a classification of models transformation approaches (template-based, graph-based, relational and so on). In our case, we are interested in *relational approaches* that can be seen as a form of constraint solving. The basic idea is to specify the relations among source and target element types using constraints. Declarative constraints can be given executable semantics, such as in logic programming. In fact, logic programming with its unification-based matching, search, and backtracking seems a natural choice to implement the relational approach, where predicates can be used to describe the relations [146]. For example, in [147], authors explore the application of logic programming. In particular *Mercury*, a typed dialect of *Prolog*, and *F-logic*, an object-oriented logic paradigm, to implement transformations. In [148] authors discuss a formalization of modeling and model transformation using a generic formalism, the *Diagrammatic Predicate Logic (DPL)*. The DPL [149, 150] is a graph-based specification format that takes its main ideas from both categorical and first-order logic, and adapts them to software engineering needs.

Definition 5.3. (**Model Transformation**) A model transformation is defined as the generation of a target model from a source model (a general definition). Formally, a model transformation consists of a set of transformation rules which are defined by input and output patterns (denoted by $\mathbb{P}$) specified at the M_2 level (the metamodel level) and are applied to instances of these meta-models. Thus, a model transformation is associated to a relation $R(MM, MN) \subseteq \mathbb{P}(MM) \times$

$\mathbb{P}(MN)$ defined between two metamodels which allows to obtain a target model N conforming to MN from a source model M that conforms to metamodel MM [151].

5.3 Relational Learning of Dependent-Concepts

The data warehouse is a database used for reporting; therefore a candidate language used to describe data is a relational database language. This language is close to Datalog language used in relational learning. In addition, the conceptual models are defined in term of relations between elements of different types (properties, classes and associations). Therefore, it is natural to use supervised learning techniques handling concept languages with the same expressive level as manipulated data in order to exploit all information provided by the relationships between data. Even if there are quite a number of efficient machine learning algorithms that deal with attribute-value representations, relational languages allows encoding structural information fundamental for the transformation process.

The attribute-based approaches are limited to non-relational descriptions of objects. In fact, the learned descriptions do not specify relations among the objects' parts. The background knowledge is expressed in a rather limited form and the concept description language is usually inappropriate for some domains. The *mode-driven data warehouse* is a new and a complex application and it is different from usual applications. In the proposed framework, the learning process will use data-models (or data schemas) to set examples and background knowledge. So, the definition of relations between model elements (e.g., class, attribute, association, etc.), is required to set-up the learning process. Relational learning provides the appropriate approach to answer this problem. This framework provides several advantages, because the defined relational information plays an important role in the resulted transformation rules. This is why ILP algorithms [31, 95] have been selected to deal with this learning problem. As ILP suffers from a scaling-up problem, the proposed architecture [132, 152] is designed in order to take into account this limitation. Thus, it is organised as a set of elementary transformations such that each one concerns a few number of predicates only, to reduce the search space. This section reminder the relational learning theory, introduces the *dependent-concept learning* approach and compares it related concept-search approaches.

5.3.1 Relational Learning Setting

We consider the machine learning problem as defined in [153]. A (single) concept learning problem is defined as follows. Given i) a training set $E = E^+ \cup E^-$ of positive and negative examples drawn from an example language $\mathcal{L}_e$ ii) a hypothesis language $\mathcal{L}_h$, iii) background knowledge B described in a relational language $\mathcal{L}_b$, iv) a generality relation $\geq$ relating formulas of $\mathcal{L}_e$ and $\mathcal{L}_h$, learning is defined as search in $\mathcal{L}_h$ for a hypothesis h such that h is consistent with E. A hypothesis h is consistent with a training set E if and only if it is both complete ($\forall e^+ \in E^+, h, B \geq e^+$) and correct ($\forall e^- \in E^-, h, B \not\geq e^-$). We refer the reader to definitions (i.e., cover relation) and (i.e., completeness and consistency) for more detail. In an ILP setting, $\mathcal{L}_e$, $\mathcal{L}_b$ and $\mathcal{L}_h$ are Datalog languages, and most often, examples are ground facts or clauses, background knowledge is a set of ground facts or clauses and the generality relation is a restriction of deduction.

In the *inductive logic programming* framework, regarding the background knowledge B and examples E, a model M_i is characterized by its description MD_i (i.e., a set of predicates that correspond to the involved elements). The predicates used to represent M_i as logic programs are extracted from its metamodel MM_i. For example, consider a data model used to manage customers and invoices. The classes *Customer* and *Invoice* are defined respectively by *class(customer)* and *class(invoice)*. As example the *one-to-many* association that relates Customer to Invoice is mainly defined by *association(customer-invoice, customer, invoice)*. Then, the logic description of models from project's traces constitutes the generated background knowledge program in ILP.

Definition 5.4. (**Transformation Example**) A transformation example (or trace model) $R(M, N) = \{r_1, \ldots, r_k\} \subseteq \mathbb{P}(M) \times \mathbb{P}(N)$ specifies how the elements of M and N are consistently related by R. A training set is a set of transformation examples.

The transformation examples are project's traces or they can be collected from different experts [154]. For instance, we are interested in the transformation of the *Data-Source PIM* (DSPIM) to the *Multidimensional PIM* (MDPIM). The DSPIM represents a conceptual view of a data-source repository and its *conformsTo* the UML CORE metamodel (part of the *unified-modelling-language*). The MDPIM

represents a conceptual view of a target data warehouse repository and its *conformsTo* the CWM OLAP metamodel (part of the *common-warehouse-metamodel*). The predicates extracted from the UML CORE metamodel to translate source models into logic program are: *type(name)*, *multiplicity(bound)*, *class(name)*, *property(name, type, lower, upper)*, *association(name, source, target)*, *associationOwnedAttribute(class, property)*, and *associationMemberEnds(association, property)*. Then, according to the CWM OLAP metamodel, the predicates defined to describe target models are: *cube(Name)*, *measure(Name, Type, Cube)*, *dimension(Name, isTime, isMeasure)*, *cubeDimensionAssociation(Cube, Dimension)*, *level(Name)*, *levelBasedHierarchy(Name, Dimension)*, and *hierarchyLevelAssociation(LevelBasedHierarchy, Level)*.

By analysing the source and target models, we observe that structural relationships (like aggregation and composition relations, semantic dependency, etc.) define a restrictive context for some transformations. For instance, let us consider the concept PropertyToMeasure. For instance, we know that there is a composition relation between Class and Property and there is also a composition relation between Cube and Measure in the metamodels. This implies that the concept PropertyToMeasure must be considered only when the transformation ClassToCube is learned. Therefore, the ClassToCube concept must be added as background knowledge in order to learn the PropertyToMeasure concept. This domain specificity induces a pre-order on the concept to be learned and defines a dependent-concept learning problem. Therefore, in our approach, concepts are organized to define a structure called *dependency-graph*. In [155], Esposito et al. use the notion of dependency graph to deal with hierarchical theories. Authors define the dependency graph as a directed acyclic graph of concepts, in which parent nodes are assumed to be dependent on their offspring.

Definition 5.5. (**Dependency Graph** after [155]) A dependency-graph is a directed acyclic graph of predicate symbols, where an edge (p, q) indicates that atoms of predicate symbol q are allowed to occur in the hypotheses defining the concept denoted by p. So, the concept p depends on the concept q (and denoted $p \preceq q$).

5.3.2 Dependent-Concept Learning Problem

Let $\{c_1, c_2, \ldots, c_n\}$ be a set of concepts to be learned in our problem. If we consider all the concepts independently, each concept c_i defines an independent ILP problem, i.e., all concepts have independent training sets E_i and share the same hypothesis language L_h and the same background knowledge B. We refer to this framework as the *Independent-Concept Learning (ICL)*. The second framework, *Dependent-Concept Learning (DCL)*, takes into account a pre-order relation[2] $\preceq$ between concepts to be learned such that $c_i \preceq c_j$ if the concept c_j depends on the concept c_i or in other term, if c_i is used to define c_j (as definition 5.5 of dependency-graph). More formally, a concept c_j is called *parent* of the concept c_i (or c_i is the *child* of c_j) if and only if $c_i \preceq c_j$ and there exists no concept c_k such that $c_i \preceq c_k \preceq c_j$. $c_i \preceq c_j$ denotes that c_j depends on c_i for its definition. A concept c_i is called *root* concept iff there exists no concept c_k such that $c_k \preceq c_i$ (in other words, a root concept c_i does not depend on any concept c_k, for $k \neq i$).

The DCL framework uses the idea of decomposing a complex learning problem into a number of simpler ones. Then, it adapts this idea to the context of ILP multi-predicate learning. A dependent-concept ILP learning algorithm accepts a pre-ordered set of concepts, starts with learning root concepts, then children concepts and propagates the learned rules to the background knowledge of their parent concepts and continues recursively the learning process until all dependent-concepts have been learned. Within this approach, we benchmark two settings: (i) the background knowledge B_j of a dependent-concept (parent) c_j is extended with the child concept *instances* (as a set of facts, and this framework is referred to as DCLI) and (ii) B_j is extended with child concept intentional definitions: all children concepts are learned as sets of rules and are added to B_j, and this frameworks is referred to as DCLR in the following sections. In both cases, DCLI or DCLR, all predicates representing child of c_j can be used in the body of c_j's definition. Our claim here is that the quality of the c_j's theory substantially improves if all its children concepts are known in B_j, extensionally or intentionally. In the next chapter (i.e., Evaluation), we provide results concerning the impact of child concepts' representation (extensional vs. intentional) on the quality of the c_j.

[2] A pre-order is a binary relationship reflexive and transitive.

The task of empirical *dependent-concept learning* of model-driven context in ILP can be formulated as follows:

Given a dependency graph $G_d = (C_d, E_d)$ where $Cd = \{c_1, c_2, \ldots, c_n\}$ the set of concepts to learn such that $\forall c_i \in C_d$ a set of examples $E = \{E_1, E_2, \ldots, E_k\}$ is given; and defined as (where m is the number of training models): $E_i = \{R_i^j(M^j, N^j) \mid R^j(M^j, N^j) \subseteq \mathbb{P}(M^j) \times \mathbb{P}(N^j),\ j \leq m\}$ and a background knowledge B which provides additional information about the examples and defined as: $B = \{\mathbb{P}(M^j) \cup \mathbb{P}(N^j) \mid M^j\ conformsTo\ MM,\ N^j\ conformsTo\ MN)\}$

Find: $\forall c_i \in C_d$, given E_d (and following a BFS strategy[3]), learn a transformation rule $R_i(MM, MN) \subseteq \mathbb{P}(MM) \times \mathbb{P}(MN)$; where MM is the reference source-metamodel and MN is the reference target-metamodel.

Compared to *layered learning*, the *dependent-concept learning* approach aims at learning a concept, using the concepts definition on which it depends. Then, while the *layered learning* approach exploits a bottom-up, hierarchical task decomposition, the DCL algorithm exploits the dependency relationships between specific concepts of the given dependency-graph. The dependency structure in [21] is a hierarchy, whereas our dependency structure is a *directed acyclic graph*. The DCL and *predicate invention* approaches share the fact they correspond to the process of introducing new theoretical relationships. However, in the case of *predicate invention*, the approach is usually based on decomposition of the theory to learn on simple sub-theories and the DCL approach is based on the composition of a theory from the learned theories. Then, as the *cascade generalization*, the DCL approach extends the background knowledge at each level with concepts of the sub-level (according to the dependency-graph). But, within the proposed *dependent-concept learning*, we use the same classifiers for all iterations.

In our experiments (provided in next chapter), we report the results of the extension of the background knowledge by instances (i.e., a first setting named DCLI) and the learned theory (i.e., a second setting named DCLR). Finally, the *model transformation by-example* approach aims to find contextual patterns in the source model that map contextual patterns in target model. This task is defined as *context*

[3]Start by an offspring and non- dependent concept (i.e., a root concept), then follow its parents dependent-concepts

analysis in [30]. However, the notion of context is different in the *dependent-concept learning*. In fact, in the DCL, contextual information is the result of the learning process (which will form the transformation rule); while within the *contextual learning* strategy the context is part of input information's that improve the performance of the learner.

5.4 Summary

Machine learning techniques have been successfully applied to various problems and applications [18, 19, 156]. Most of these applications rely on attribute-values learning [85, 90, 157]. Attribute-based learning is limited to non-relational descriptions of objects in the sense that the learned descriptions do not specify relations among the objects' parts. Examples representing relational information are difficult to be encoded into propositional representation, i.e., using attribute-values. In fact, the data warehouse system is defined as a database with relations between different concepts: *cube*, *measure*, *dimension*, and so on. These relations are necessary in the definition of the transformation (notion of transformation context). Traditional machine learning, which only operates on propositional examples, relies on propositionalisation [158, 159] to transform relational information into attribute-values. However such transformation is not always possible and often causes loss of information.

The term *inductive logic programming* was coined [160] in order to highlight the emerging area of research on the intersection of machine learning and logic programming. Both input and output of ILP take the form of first-order logic. The representation of first-order logic gives ILP many advantages over attribute-based machine learning techniques. It is remarkably easier to incorporate domain knowledge with ILP than other machine learning techniques. In fact, background knowledge can easily be amended by adding or removing clauses. However, in complex systems the ILP approach takes long time to get the results. Therefore, the search space is reduced because only part of the modelling specification is used for each layer and each component. This is very well identified and described in our architecture. The search space is also invariant, since research takes place at M2 (metamodel). The metamodeling architecture is defined as models abstraction with

several levels (M0, M1, M2 and M3). The M2 specification is the generalization of all models at level M1 (which defines projects trace). So, by using M2, our search space remains invariant for the data-models (which are at M1). We choose recognized standards (i.e., UML and CWM) to facilitate the exchange with the experts. This also allows understanding of the obtained rules and for easy integration of our framework in existing systems.

The main goal is to automatically derive the transformation rules to be applied in the *model-driven data warehouse* process. This aims to reduce the contribution of transformations designer and thereby reducing the time and the cost of datawarehouses development. We use the *inductive logic programming* framework to express the model transformation and we find a new methodology (the *dependent-concept learning*) that is suitable to solve this kind of problem. In this chapter, a formalisation of model-driven concepts is provided and the problem statement in a relational learning setting is expressed. The DCL problem is defined and the learning approach of model transformations is provided. For the evaluation (presented and discussed in the next chapter), we propose to compare the following approaches: (1) The *independent-concept learning* approach, which proposes to learn the set of considered concepts independently; and (2) The *dependent-concept learning* approach, which consider a dependency graph to learn the concepts. Within this second approach, we benchmark two settings: (i) the background knowledge B of dependent-concepts (i.e., parent-concepts) is updated with their child instances (which is denoted as DCLI) and (ii) with their child intentional definitions (which is denoted as DCLR). The obtained results support the conclusion that the *dependent-concept learning* approach is suitable to solve this kind of problem.

Chapter 6

Evaluation

The proposed model-driven data warehousing architecture defines the input and output data-models (or schemas) for each transformation to learn. So, it simplifies the design of the learning process and the definition of predicates used for background knowledge and examples. Then, one of the problems on which we focus is to find an adequate representation for the language bias in the model-driven data warehouse framework. The proposed representation allows for an improved learning process and also for a good quality of the obtained rules. We define the language bias using the metamodel level (denoted M2 level) of the meta-modelling architecture. This gives the advantage to define a clear set of predicates with an optimal level of abstraction. The experiments presented in this chapter allow also validating the proposed architecture and language.

The evaluation is performed in two steps with two different datasets (i.e., by two experiments). In, the first experiments use a set of real-world data models provided by our industrial partner (i.e., Intelligence Power). Then, in the second experiments, we use the Microsoft AdventureWorks 2008R2 sample database [161]. We use Aleph [162], an inductive logic programming engine, to learn first-order transformation rules.

6.1 Materials

The first experimental setup uses a dataset of real-world data models provided by our industrial partner (i.e., Intelligence Power). It mainly compares the results of the tested settings: ICL, DCLI and DCLR. The *Independent Concept Learning (ICL)* approach proposes to learn the set of considered concepts independently. Then, the *Dependent Concept Learning (DCL)* approach as it is described in the previous section explores a dependency-graph to learn concepts. Within this approach, we benchmark two settings: (i) the background knowledge B of dependent-concepts is updated with parent concept instances (this setting is denoted as DCLI) and (ii) with parent-concepts intentional definitions (and is denoted as DCLR). The used set of real-world data models represent projects' traces such as the example presented in figure 4.15. In each project trace, we find the *source-model(s)*, the *target-model(s)* and the *transformations*. For these experiments, we have selected 10 model instances (of database schemas) describing several application domains *(invoices, sales, e-commerce, banking- investment, and so on)*. The *source-models* description mainly includes the definition of *classes, associations, and properties* elements. In the *target-models*, we find elements like *cubes, measures, dimensions and levels*. From each model, we extract a set of positive and negative examples that define respectively positive and negative transformations as explained before.

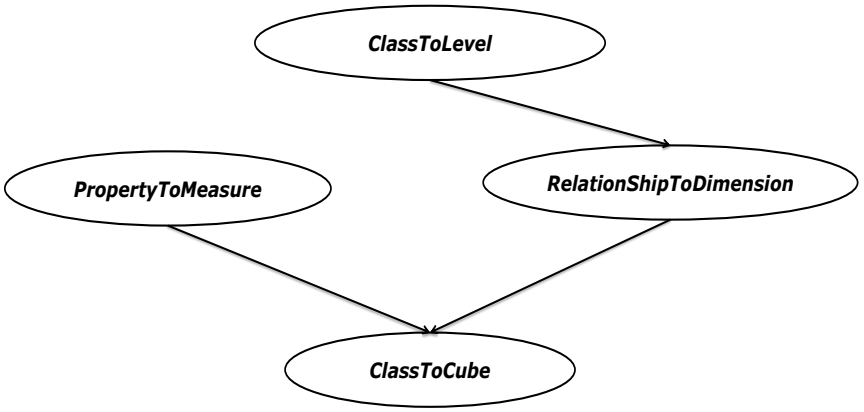

FIGURE 6.1: Pre-order Dependency Relation Between Considered Concepts.

From the analysis of this first dataset, we identify the following concepts dependencies (see figure 6.1):

TABLE 6.1: Number of examples (E^+ and E^-) per-concept used for learning.

Concept	Positive Examples	Negative Examples	Total Number
ClassToCube	27	44	71
PropertyToMeasure	47	202	249
PropertyToDimension	38	207	245
RelationShipToDimension	33	60	93
ElementToHierarchyPath	115	223	338
ElementToDimensionLevel	109	229	338

- ClassToCube $\preceq$ PropertyToMeasure: The PropertyToMeasure concept depends on the concept ClassToCube. In general, transformation of *properties* depends on contextual information of transformed *classes* and the context of obtaining measures is part of the context of obtaining *cubes*. In fact, *properties* that become *measures* are numeric properties of classes that become cubes. So, we need information about the context of ClassToCube transformation in order to find the context of PropertyToMeasure.

- ClassToCube $\preceq$ RelationShipToDimension: Indeed, dimensions are obtained from relationships of the class that is transformed into cube. This is the case of the *invoice* class and *invoice-customer* association. The CubeDimensionAssociation meta-class relates a Cube to its defining dimensions as showed by the CWM OLAP metamodel in [130]. These relationships define the axes of analysis in the target multidimensional schema [163]. The experiments presented in the next show significant performances improvement when we extend the background knowledge of RelationShipToDimension by ClassToCube instances.

- RelationShipToDimension $\preceq$ ClassToLevel: There is a two-level dependency between ClassToLevel and ClassToCube concepts. As the example in figure 4.15, *classes* that participate in a transformed *association* into a *dimension* are the levels of the resulting dimension. We will also experiment learning the ClassToLevel concept using the knowledge provided by the ClassToCube and RelationShipToDimension concepts.

Concerning the second set of experiments, we use the Microsoft AdventureWorks 2008R2 reference databases [161]. The Microsoft AdventureWorks reference databases are AdventureWorks Sample OLTP Database (AdventureWorksOLTP) and AdventureWorks Sample Data Warehouse (AdventureWorksDW). The AdventureWorksOLTP is a sample operational database used to define the source-model

(i.e., the data- source schema – DSPIM). The AdventureWorksDW is a sample data warehouse schema used as target-model (i.e., the multidimensional schema – MDPIM). The AdventureWorksOLTP, AdventureWorksDW and the mapping between them (evaluated by the expert) are considered as a reference project-trace. This will allow us to benchmark our approach on a new extended schema (that generate more examples) and a new dependency-graph. The databases elements (i.e., classes, properties and associations) are encoded as background knowledge (B) and the mapping instances between their elements allows to define positive (E^+) and negative (E^-) examples. The number of examples, available pour each concept are provided by table 6.1. Then, the average accuracy results from the 10x10-fold cross validation are then reported by figures below.

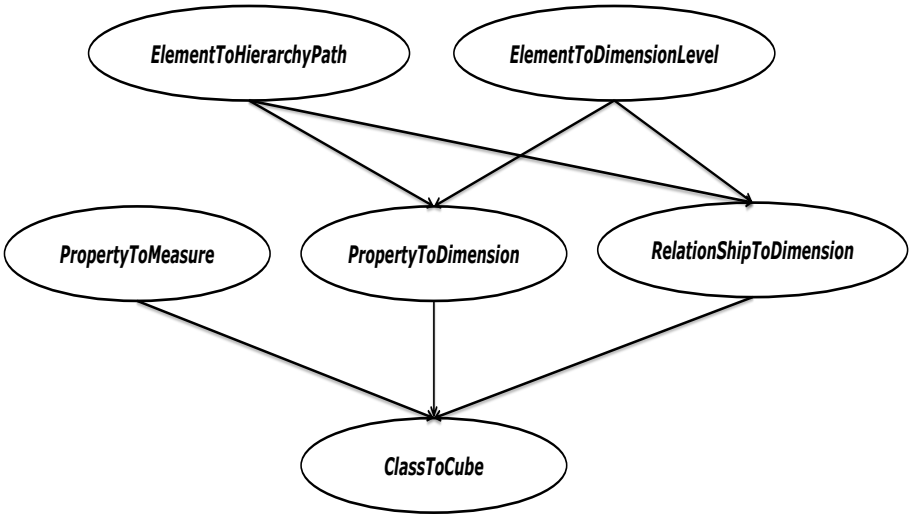

FIGURE 6.2: Considered Dependency-Graph of Second Experiments.

We run Aleph in the default mode, except for the *minpos* and *noise* parameters: *:- set(minpos, p)* establishes as p the minimum number of positive examples covered by each rule in the theory (for all experiments we fix $p = 2$); and *:- set(noise, n)* is used to report learning performance by varying the number of negative examples allowed to be covered by an acceptable clause (we use two setting $n = 5$ and $n = 10$). We propose also to compare the *Independent-Concept Learning (ICL)* and the *Dependent-Concept Learning (with more different settings, DCLI, DCLR, etc.)* approaches. We use the concept dependencies illustrated by the graph in figure 6.2. It integrates the ClassToCube $\preceq$ PropertyToMeasure and ClassToCube $\preceq$ RelationShipToDimension dependencies; and adds the following dependencies compared to the first graph:

- ClassToCube $\preceq$ PropertyToDimension: This defines dependency between *classes* transformed into *cubes* and their *properties* that can be transformed into *dimensions*. Regarding the UML CORE metamodel, we find a structural dependency between Class and Property elements (a Class includes attributes, represented by the ownedAttribute role that defines a set of *properties*). Then, regarding the CWM OLAP metamodel, we have a structural dependency between Cube and Dimension elements. Current experiments confirm that structural dependencies in the metamodel act on the ways to perform learning.

- *(PropertyToDimension, RelationShipToDimension)* $\preceq$ RelationShipToDimension: A Dimension has zero or more hierarchies. A Hierarchy is an organizational structure that describes a traversal pattern through a Dimension, based on parent/child relationships between members of a Dimension. Then, elements that are transformed into dimensions (*properties* and *relationships*) extend the background knowledge used to find hierarchy paths.

- *(PropertyToDimension, RelationShipToDimension)* $\preceq$ ElementToDimensionLevel: A LevelBasedHierarchy describes hierarchical relationships between specific levels of a Dimension (e.g., Day, Month, Quarter and Year levels for the Time dimension). So, rules of transforming elements into Dimension are used to find rules of obtaining the levels.

The number of model-elements used for experiments forms the size of the background knowledge. In the first experiment we use a dataset consisting of 10 models with 475 model-elements. In the second set of experiments, the sample consists of 6 schemas (of AdventureWorks) containing 1028 elements (has doubled). The results of running show almost equivalent performance between the first and second experiment. The proposed architecture and the partitioning allow the use of a small number and specific predicates for each concept. The search space remains invariant as it always uses the same number of elements of metamodels that describe the background knowledge and examples. This structure is used to define invariants of transformation from the definition of metamodels. We conclude that this type of *model-driven architecture* with a two-dimensional partitioning (layers and design-levels) is the best to support and scaling-up machine learning integration.

6.2 Evaluation Methods

The first goal of this analysis is to examine if the choice of the number of training models and examples influence the performances. We report the obtained test accuracy curves of learning ClassToCube and PropertyToDimension concepts. They represent concepts of the first-level (level 1 of learning, or child-concepts). The second goal of the analysis is to study the performances of the DCL approach (with the two settings DCLI and DCLR) compared to the ICL approach. We report the *Receiver Operating Characteristics (ROC)* curves of the tested approaches (ICL, DCLI and DCLR) for learning PropertyToMeasure, RelationShipToDimension and ClassToLevel concepts (concepts of level 2 and 3). For all experiments, we use the repeated random sub-sampling validation strategy (the average of 10 iterations is reported).

The ROC graphs are a useful technique for visualizing, organizing and selecting classifiers based on their performance [164]. The following metrics are used to report the ROC graphs. The *true-positive-rate* (also called *hit rate* and *recall*) and the *false-positive-rate* (also called false alarm rate) of a classifier are estimated as $tp\ rate = \frac{TP}{P}$; $fp\ rate = \frac{FP}{N}$. Additional terms associated with ROC curves are sensitivity and specificity: $sensitivity = recall = \frac{TP}{P}$; $specificity = 1 - fp\ rate = \frac{TN}{N}$. ROC graphs are two-dimensional graphs in which *tp rate (sensitivity)* is plotted on the Y axis and *fp rate (1 - specificity)* is plotted on the X axis. For the second goal of our experiments, we build ROC graphs for PropertyToMeasure, RelationShipToDimension and ClassToLevel to compare performances of the tested approaches (ICL, DCLI and DCLR). Figure 6.5 shows results obtained for PropertyToMeasure, figure 6.6 reports the result curves of learning RelationShipToDimension and figure 6.7 gives performances of learning ClassToLevel.

We use the *accuracy* measure for the first goal of this benchmark (i.e., to examine how the number of training model instances and examples influence the performances). In machine learning accuracy is commonly used for comparing the performance of algorithms. Thus, many researchers report their results in terms of accuracy. The accuracy of a model can be interpreted as the expectation of correctly classifying a randomly selected example. We examine the accuracy of the learned rules to show the impact of the number of training models and examples

on the training performances. Accuracy is defined, based on the *contingency table (also known as confusion matrix)*, as $Accuracy = \frac{TP+TN}{P+N}$, where P (N) is the number of examples classified as positive (negative), TP (TN) is the number of examples classified as positive (negative) that are indeed positive (negative). Then, the *Area Under the ROC Curve*, abbreviated AUC, is the common measure to compare the tested methods. The AUC represents also a measure of accuracy.

6.3 Experiments

We performed experiments on two different datasets, and we note that in both cases, the transformations (their number and the dependencies between them) are not the same. Moreover, through these experiments, we conclude that more the number of models (or data) from the knowledge base is important, more the dependency graph is complex (increased number of nodes and relationships). Regarding the increased number of transformations in the second graph, we show that the application keeps acceptable performance in terms of execution time and quality rules. This is explained by the fact that the proposed architecture has several advantages in terms of division of the problem. The overall design is organized through several levels of modelling (and so learning levels). The representation language is reduced for each transformation and then offers a support for scalability.

We use Aleph [162], an inductive logic programming engine, to learn first-order transformation rules. As input, Aleph takes: (i) background information in the form of predicates, (ii) a list of modes declaring how these predicates can be chained together, (iii) a designation of one predicate as the "head" predicate to be learned, and (iv) a lists of positive and negative facts of the head predicate are also required. The learned logical clauses give the relationship between the transformations and the contextual information (elements) in the models. The contextual information in model-driven methodologies corresponds to the model elements and the relationships between them (i.e., the concepts of the model) in different abstraction levels. This context can be associated to a sub-model that matches the required information to apply a transformation on a concept.

We run Aleph in the default mode, except for the *minpos*, parameter, *:- set(minpos, 2)* establishes as 2 the minimum number of positive examples covered by each rule

in the theory. The mode definitions in Aleph are required to produce a theory. We give below the Aleph modes declaration for ClassToCube and PropertyToMeasure as examples:

```
:- modeh(1,classtocube(+class,+cube)).
:- modeb(*,class(+class)).
:- modeb(*,cube(+cube)).
:- modeb(*,property(+property,#type,#multiplicity,#multiplicity)).
:- modeb(*,association(-association,+class,-class)).
:- modeb(*,associationOwnedAttribute(+class,-property)).
:- modeb(*,associationMemberEnds(+association,-property)).

:- modeh(1,propertytomeasure(+property,+measure)).
:- modeb(*,measure(+measure,#type,+cube)).
:- modeb(*,class(+class)).
:- modeb(*,cube(+cube)).
:- modeb(*,classtocube(+class,+cube)).
:- modeb(*,property(+property,#type,#multiplicity,#multiplicity)).
:- modeb(*,associationOwnedAttribute(-class,+property)).
```

The accuracy of the second set of experiments based on the new dataset (of AdventureWorks) confirm the first results reported also in [130]. Then, considering the second dependency-graph, we study also the performances of the DCL approach (with the two settings DCLI and DCLR) compared to the ICL approach. We report in this section the ROC curves of the tested approaches (ICL, DCLI and DCLR) based on the new dataset and the new enhanced dependency-graph. In order to assess the impact of a child-concept rules quality on the learning performances of a parent concept, we experiment the case where the child concept is noisy. This experiment is made within the DCL approach, we add noise to the non-dependent concept (i.e., ClassToCube) and we observe results of learning dependent-concepts with different acceptable noise setting ($n = 5$ and $n = 10$). We report the cases where 10% (denoted N-DCLI and N- DCLR) and 20% (denoted N2-DCLI and N2-DCLR) of the examples are noisy. To add noise, we swap positives and negatives examples.

6.4 Results and Discussion

In figures 6.3 and 6.4, we report the obtained results for ClassToCube and PropertyToDimension concepts (concepts of level 1). Figure 6.3 gives the average test accuracy as a function of the number of training models. Figure 6.4 shows the test accuracy, averaged over 10 models, now as a function of the number of training examples. The obtained results indicate, as expected, that accuracy increases with the number of models or examples. Results also give the minimum number of model instances or examples required to learn these concepts. For instance, with 10 models, the accuracy is for $96,5\%$ for the concept ClassToCube and about $81,3\%$ for concept PropertyToDimension. The main explanation, as shown by figure 6.4, comes from the difference between number of examples describing the two concepts. Then, from an expert point-of-view, we expect a potential dependency between these two concepts which does not appear in the concerned data.

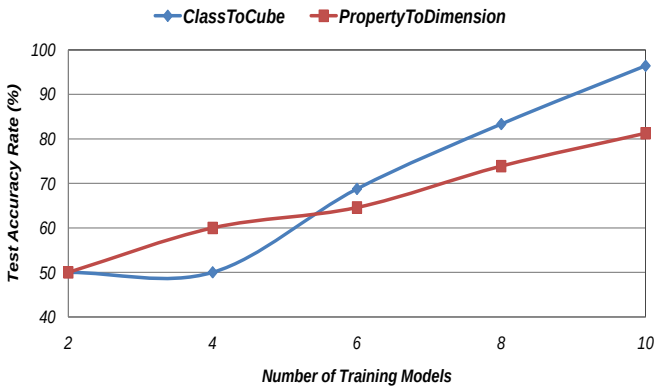

FIGURE 6.3: ClassToCube and PropertyToDimension Accuracy as a Function of the Number of Models (ICL Setting).

Aleph induces the following rules with the best score for ClassToCube:

```
classtocube(A,B) :- class(A), cube(B), property(C,float,1,1),
associationOwnedAttribute(A,C), association(D,A,E).
```

```
classtocube(A,B) :- class(A), cube(B), property(C,integer,1,1),
associationOwnedAttribute(A,C), association(D,A,E).
```

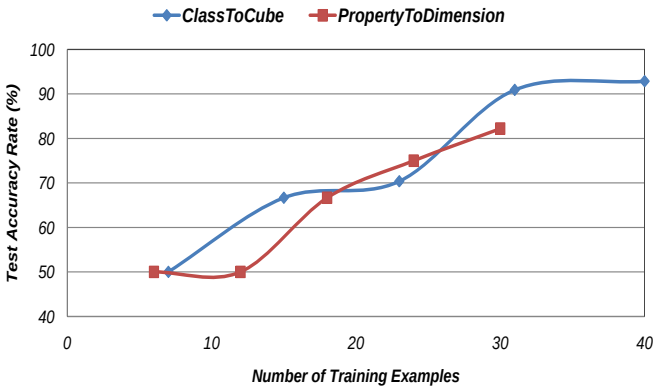

FIGURE 6.4: ClassToCube and PropertyToDimension Accuracy as a Function of the Number of Examples (ICL Setting).

A class A with property C of type either integer or float (from the source-model) is transformed to a cube B (in the generated target-model) i.e., *associationOwnedAttribute(A, B), property(B, float/integer, 1, 1).*). This result defines the source-context of ClassToCube transformation. The *association(C, A, E).* indicates that the transformed class A participates is an association that can define a hierarchy-path of a dimension (where A as a second parameter represent the source-element of the association). This will defines contextual information for the transformation that generates dimensions and associates them to the cube. Concerning the semantic of each predicate, please refer to Chapter 4 for details.

We note that the learned rules for ClassToCube are close to the rules designed manually. We compare the resulted rules with those provided by related work. For example in [28], the source-model context of the proposed EntityToCube is formed by Entity, RelationShipEnd (with multiplicity = '*') and Attribute (with numeric types) relations. In this format, the numeric type represents float and integer numbers. Regarding the resulting rules, *associationOwnedAttribute(A, C) and association(D, A, E)* atoms are associated with RelationShipEnd relation; then *property(C, float, 1, 1) and property(C, integer, 1, 1)* to Attribute relation. In fact, we have proposed a good language bias and a good modelling bias based on the same domain metamodel used by experts (UML and CWM) and this explains the obtained rules. Also, all of the resulting rules are found in a reasonable time: 30 (and 35) seconds on average for the first (for the second) experiments.

Concerning concepts PropertyToMeasure, RelationShipToDimension and ClassToLevel, Aleph induces also the following rules with the best score. For each concept, the

obtained rules include predicates of children-concepts in the dependency-graph, when learned in the dependent-concept learning framework:

```
% PropertyToMeasure
propertytomeasure(P,M) :- associationOwnedAttribute(A,P),
measure(M,float,C), classtocube(A,C), property(P,float,1,1).
propertytomeasure(P,M) :- associationOwnedAttribute(A,P),
measure(M,integer,C), classtocube(A,C), property(P,integer,1,1).

% RelationshipToDimension
relationshiptodimension(C,D) :- association(C,A,E),
classtocube(A,B), dimension(D,false,false).
relationshiptodimension(C,D) :- association(C,A,E),
classtocube(A,B), dimension(D,true,false).

% ClassToLevel
classtolevel(D,L) :- level(L), association(C,A,E), classtocube(A,B),
relationshiptodimension(C,D), levelBasedHierarchy(H,D).
```

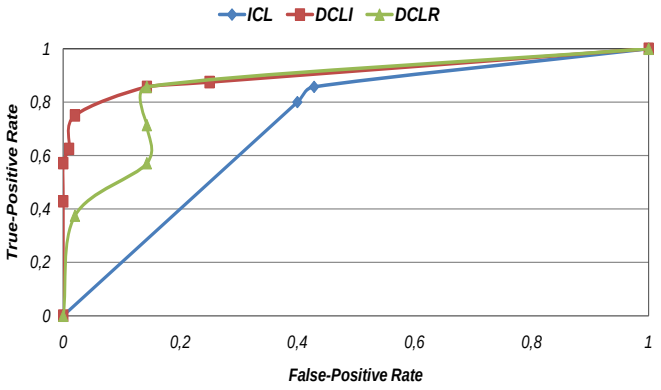

FIGURE 6.5: ROC Curves of Learning PropertyToMeasure.

The first results on figures 6.5, 6.6 and 6.7 show that the DCLI has greater AUC than other tested methods (i.e., ICL and DCLR). The DCLI curves follow almost the upper-left border of the ROC space. Therefore, it has better average performance compared to the DCLR and ICL ($AUC_{DCLI} > AUC_{DCLR} > AUC_{ICL}$). The ICL curves almost follow to the 45-degree diagonal of the ROC space, which represents

a random classifier (so poor performance). The DCLR setting exhibits good results with respect to the ICL approach, which are nevertheless slightly worse than the results of the DCLI setting.

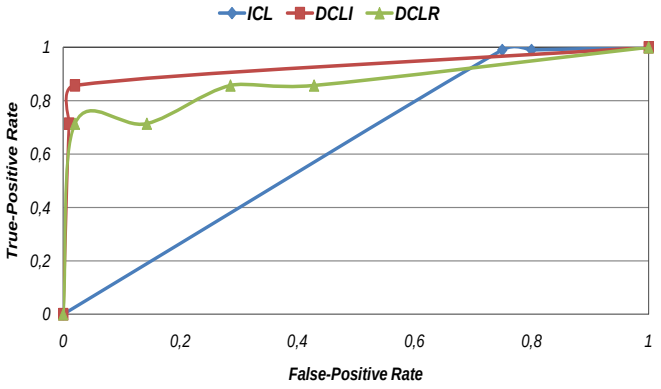

FIGURE 6.6: ROC Curves for Learning RelationShipToDimension.

Within DCLI configuration, when learning a concept of level $i \geq 2$, uses in its background knowledge concepts of level $j < i$ as set of facts (extensional definition), as opposed to DCLR, which previously learns as sets of rules definitions for concepts of level $j < i$. If lower-level concepts (i.e., children-concepts) are not perfectly identified, the learning errors propagate to higher- level concepts (i.e., parent-concepts). We assume here that examples are noise-free, which explains why DCLI has a better behaviour than DCLR. Thus, for PropertyToMeasure and RelationShipToDimension (concepts of level 2), results integrate the error rate from ClassToCube learned rules (concept in level 1). For ClassToLevel that depends on RelationShipToDimension, results are influenced by the error rate propagation from learning ClassToCube and then RelationShipToDimension.

Then, as example of obtained graphs within the second set of experiments, the learning results for PropertyToMeasure and RelationshipToDimension are reported by figures 6.8 and 6.9. The curves show that $n = 10$ setting (right part of each figure) gives best performances compared to $n = 5$. Indeed, *data quality* and *conceptual models quality* [165, 166] play an important role in the design of information systems, and in particular *decision support systems*. Then, comparing ICL, DCLI and DCLR approaches, results show that the DCLI has greater AUC than other tested methods. The DCLI curves follow almost the upper-left border of the ROC space. Therefore, it has better average performance compared to the DCLR and ICL ($AUC_{DCLI} > AUC_{DCLR} > AUC_{ICL}$). The ICL curves almost follow to the

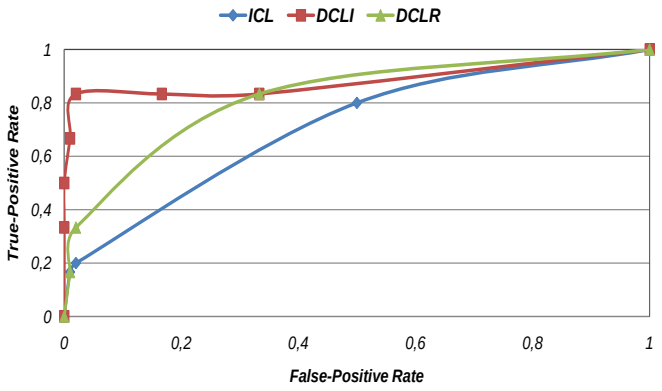

FIGURE 6.7: ROC Curves of Learning ClassToLevel.

45-degree diagonal of the ROC space, which represents a random classifier. The DCLR setting exhibits good results with respect to the ICL approach, which are nevertheless slightly worse than results of the DCLI setting.

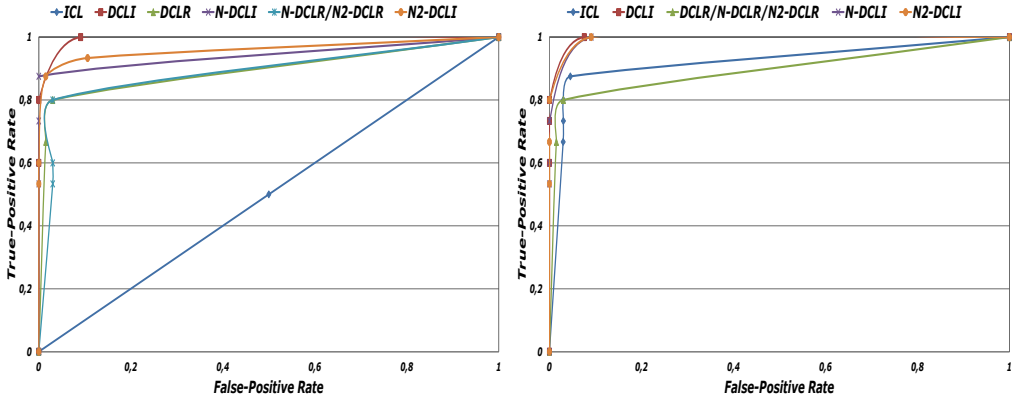

FIGURE 6.8: Learning PropertyToMeasure, n = 5 (left) and n = 10 (right).

In the actual setting, $AUC_{DCLI} > AUC_{DCLR} > AUC_{ICL}$ result is expected, because the DCLI configuration, when learning a parent concept, uses in its background knowledge child-concepts as set of facts (extensional definition), as opposed to DCLR, which previously learns as sets of rules definition for offspring concepts. In case lower level concepts (i.e., child-concepts) are not perfectly identified, the errors for offspring concepts propagate to parent concepts. We assume here that examples are noise-free, which explains why DCLI has a better behaviour than DCLR. Thus, for PropertyToMeasure, PropertyToDimension and RelationShipToDimension, results integrate the error rate from ClassToCube learned rules. We observe also that for the parent-concepts ElementToHierarchyPath and ElementToDimensionLevel

that depend on PropertyToDimension and RelationShipToDimension, results are influenced by the error rate propagation from learning ClassToCube and then PropertyToDimension and RelationShipToDimension. Another remarkable point concerning curves is that the gap between ICL and DCL becomes important for top-level concepts (i.e., parent-concepts) in the dependency-graph. So in this case, the contribution of DCL becomes more significant. For example that the gap is most important for RelationShipToDimension concept (in figure 6.9) than PropertyToMeasure concept (in figure 6.8). This is explained by the fact that when finding top-level concepts using ICL, the learning configuration will be deprived of much more information on all intermediate concepts.

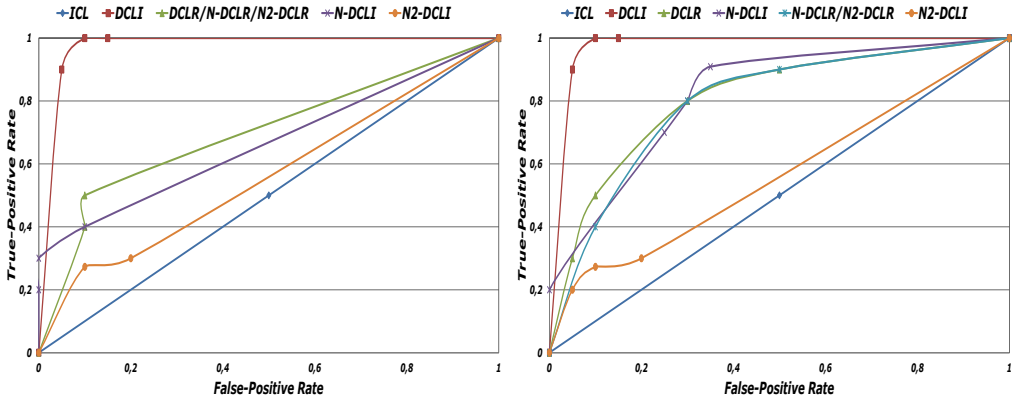

FIGURE 6.9: ROC Curves for Learning RelationshipToDimension.

Considering the N-DCLI, N2-DCLI, N-DCLR and N2-DCLR settings, we have mainly: $AUC_{N-DCLI} > AUC_{N2-DCLI}$ and $AUC_{N-DCLR} > AUC_{N2-DCLR}$. Curves show that the obtained performances depend on the concept to learn and its *degree-of-dependence* on ClassToCube (the noisy non-dependent concept of this configuration). For instance, in figure 6.9, RelationShipToDimension is most impacted than PropertyToMeasure in figure 6.8. The PropertyToDimension and RelationShipToDimension concepts are highly dependent on ClassToCube. This can be observed on most schemas (remarks provided in first experiments) and it is confirmed by the expert point-of-view. For example, in the case of RelationShipToDimension, the N2-DCLI curve seems to reach the 45-degree diagonal. This gives us an idea of the noise that we can accept when learning specific dependency relationships.

The model-driven process, based on *two track unified process* is one of the components of the architecture that we propose. This process offers a comprehensive

TABLE 6.2: Test Accuracy for Learning RelationShipToDimension (First Setting).

Approach	***Test Accuracy Average (%)***
	(PropertyToDimension,RelationshiptoDimension), i.e., information on the two concepts is given
DCLI (n=5)	*87*
DCLR (n=5)	*71,9*
DCLI (n=10)	*87*
DCLR (n=10)	*77,6*

partitioning by layer (or component) and a local partitioning (by design-level). The proposed frameworks, largely based on *unified modelling language* and *common warehouse metamodel* define the second component of the architecture. These frameworks are used to define the representation language of transformations to learn. The choice of industry standards (e.g., UML, CWM), recognized by experts, allows ensuring a good level of system integrity and also provides an optimal representation language (for understandable rules). The transformations whose context is explained in Chapter 4, are bound by the execution-dependency (the "where" relationship, or the post-condition). The proposed approach, based on the dependency-graph is consistent with this definition of transformations. Indeed, execution-dependencies are transformed (or reduced) into search- dependencies. This reduction problem creates the best environment for defining parent-concepts and improves the quality of the obtained rules; thus ensuring an effective assistance to experts. When there are more changes, there are more execution-dependencies (problem of scaling-up). If these dependencies are not considered (this is the ICL case), the error rate, the lack of information and consistency will be more important in scaling. The DCL approach addresses this problem by an analysis and definition of a dependency-graph taking into account the number of possible transformations.

TABLE 6.3: Test Accuracy for Learning RelationShipToDimension (Second Setting).

Approach	***Test Accuracy Average (%)***	
	(-,RelationshiptoDimension), i.e., losing PropertyToDimension	***(PropertyToDimension,-),*** i.e., losing *RelationshiptoDimension*
DCLI (n=5)	*76,7*	*76,7*
DCLR (n=5)	*70,8*	*70,1*
DCLI (n=10)	*77,2*	*76,7*
DCLR (n=10)	*74,6*	*69,9*

The third goal of the analysis is to experiment the case when we lost a node in the dependency-graph. Tables 6.4 and 6.5 give results of learning ElementToHierarchyPath and RelationShipToDimension when (i) we consider all parent-concepts

(i.e., PropertyToDimension and RelationShipToDimension), (ii) when we lost PropertyToDimension, denoted (-, RelationShipToDimension) and (iii) when we lost RelationShipToDimension, denoted (PropertyToDimension, -). We report the test accuracy average of 10-fold random sampling method. Results confirm that taking into account the two dependency relationship, i.e., setting (PropertyToDimension, RelationShipToDimension), is necessary to ensure good performances. For example, we note a decrease of $\simeq 10\%$ when we lost PropertyToDimension or RelationShipToDimension. This experiment allows us to verify dependency relationships and their degree, then to validate the dependency-graph.

TABLE 6.4: Test Accuracy for Learning ElementToDimensionLevel (First Setting).

Approach	***Test Accuracy Average (%)***
	(PropertyToDimension,RelationshiptoDimension) i.e., information on the two concepts is given
DCLI (n=5)	*89,4*
DCLR (n=5)	*73,4*
DCLI (n=10)	*89,4*
DCLR (n=10)	*77,7*

This also allows for defining a certain *degree-of-dependence* of a concept against another. For example test accuracy results of learning RelationShipToDimension (table 6.3), in case of DCLR ($n = 10$) setting are: $74,6\%$ (for losing PropertyToDimension) and $69,9\%$ (for losing RelationShipToDimension). Thus, losing RelationShipToDimension gives a lesser accuracy result than losing PropertyToDimension. We have the same observation in table 6.5 (learning ElementToDimensionLevel). In case of DCLR ($n = 10$) setting, the accuracy results are: $74,9\%$ (for losing PropertyToDimension) and $72,6\%$ (for losing RelationShipToDimension). This joins our remark in the first experiments concerning the gain in accuracy. Finally, advanced analysis of gain in accuracy and degree of dependence will be part of a future work.

TABLE 6.5: Test Accuracy for Learning ElementToDimensionLevel (Second Setting).

Approach	***Test Accuracy Average (%)***	
	(-,RelationshiptoDimension) i.e., losing PropertyToDimension	***(PropertyToDimension,-)*** i.e., losing *RelationshiptoDimension*
DCLI (n = 5)	*79,7*	*79,6*
DCLR (n = 5)	*72,5*	*71,6*
DCLI (n = 10)	*79,6*	*79,6*
DCLR (n = 10)	*74,9*	*72,6*

6.5 Summary

We note that the obtained transformation rules are similar to the transformation manually designed using the *query-view-transformation* language presented in [28]. Results show acceptable accuracy values for ClassToCube, the error rate is around 4% with 10 models and around 7% with $E = 40$. This shows that learning is possible with a small number of models (examples). This is achieved through a strong domain theory and the proposed efficient modelling. Indeed, in our model-driven approach, the models used for learning are represented at a conceptual level (resulting from the reverse engineering of logical schemas). This reduces numeric attributes of type primary-key and/or foreign-key that we find in the database logical representation and therefore it reduces the number of negative examples covered by the learned rules. We consider this as an important language bias that improves learning results and performances. This means that properties of type *date* are transformed into dimensions, which covers many examples. However, for PropertyToDimension, the error rate remains high: around 19% with 10 models and around 18% with $E = 30$. Indeed, there is a high number of attributes transformed into dimensions which are not of type *date*. This is the case for the numeric degenerate dimensions [167] and the categorical dimensions [168] that can have a string type. Such examples are obviously missing from our training set.

The analysis of project traces allows identifying specific business goals that must be addressed by the generated data warehousing system. The structure of the data warehouse will be generated by the learned transformations. Therefore the context of these transformations must be consistent with the stated goals in order to generate the expected data warehouse specification. It is concluded that a step of data analysis project is required to identify the necessary transformations and relationships between them (dependencies). A careful definition of the transformations context and a faithful design of the dependency-graph can improve learning performance thereafter. This step allows defining a language bias (i.e., the transformations context) of and a search bias (i.e., the dependency-graph). We show that the DCLR improves transformation learning and allows for more accurate results compared to the ICL approach. Then, we examine the *gain in accuracy (GA)* of the DCLR and ICL. Let $GA^{C}_{DCLR-ICL} = AUC^{C}_{DCLR} - AUC^{C}_{ICL}$ be the gain in accuracy (of DCLR compared to ICL) when learning the concept C.

The *gain in accuracy* information allows us to define a *degree-of-dependence* of a concept with respect to another. We introduce and we experiment the *degree-of-dependence* within the second experiment. Within the actual configuration and the example of PropertyToMeasure and RelationShipToDimension, we observe that $GA_{DCLR-ICL}^{PropertyToMeasure} < GA_{DCLR-ICL}^{RelationShipToDimension}$ (this analysis will be part of a future work).

The *dependent-concept learning* is better because adding a child-concept description allows the definition of new information (new context) to consider when learning the parent-concept. This adds dependency information considered as an information that enriches the search space of the parent-concept and helps finding expected relations. This plays as an additional language bias, but also a search bias, allowing for good learning performances and good rules quality. The learned theory of the child-concept extends the background knowledge with a specific theory simple to learn, but at the same time it defines a sub-context of the parent-concept theory. So, this learning strategy first finds relations that are simple to learn with a minimum number of model elements. Then, the resulted sub-theories are used to set-up the learning context of parents-concepts. This approach is suitable to solve model-driven based problem because: (i) in metamodels definitions, we find dependencies between model elements (class, attribute, cube, measure, etc.), and (ii) in the manually designed rules, the "where" part of the transformation defines rules that must be activated (or executed) as post-condition. This post- condition information is a form of dependency.

In this evaluation, a sensitivity analysis of classifiers is performed. Indeed, we can tell how robust a classifier is, by noting the classification accuracy of learning approaches using noisy data of different noise levels. We use two different percentages of noise, 10% of the original data set (approaches working on the obtained noisy dataset are denoted N-DCLI and N-DCLR) and 20% (the obtained datasets are denoted N2-DCLI and N2-DCLR). Similar experiments on all datasets are performed and the resulted behaviours are compared to the ICL and DCL results using original datasets. Figures show that, in the presence of noise, and for most concepts, the classification accuracy of the DCLs settings drops less than those of the ICL approach. The performance degradation measures effect of noise on the classifiers. A classifier is more tolerant and resistant to the noise when it shows a smaller performance deviation. Regarding child-concepts (e.g., ClassToCube,

PropertyToMeasure), the behaviour of approaches that learn on noisy-datasets remains close to the DCLs settings those work on the standard datasets (noise-free datasets). Nevertheless, the more we advance in levels of the dependency-graph, the less resistant are these approaches to noise. Their accuracy will be much lower than the non-noisy approaches and gets closer to the performance of ICL setting. We conclude that child-concepts are more tolerant to noise, because the learning environment is easier (none or few dependencies exist) and the search space is reduced. Also, noisy-data do not support obtaining good performance in the case of large dependency graphs. Thus, a larger hierarchy of concepts can significantly reduce the quality of the rules at the parent-concepts, because their learning environment is noisy by error propagation of learning child-concepts. This part of work of learning using noisy data will be studied and evaluated with more details in our future work.

Chapter 7

Implementation

In recent years, the data warehousing infrastructures have undergone many changes in various aspects. This is usually due to many factors: the emergence of software-as-a-service architecture model; the success of agile and iterative data warehouse development approaches; the introduction of new approaches based on the model-driven architecture; the changing needs of organizations and the extension of the data warehouse into new application areas; and the evolving of standards and open-source technologies.

This chapter explores several aspects that may influence the next generation of data warehousing platforms: (i) the architectural aspects for business intelligence-as-a-service deployment; (ii) the promising open industry standards and technologies recommended for use; and (iii) the emerging methodological aspects for data warehouse components engineering.

7.1 Motivation

The *Software-as-a-Service (SaaS)* is a model for software delivery that allows lower total cost of ownership and better return on investment for subscribers [126]. Recently, several business intelligence platforms including some well known names have embraced this new model of architecture. These solutions present many advantages derived in general from the advantages of the SaaS concept. However, not much information on the architecture of these *on-demand* solutions is provided. In addition, few solutions offer integrated platforms that cover all functional and technical aspects of the data warehousing architecture. Indeed, these tools focus only on the problem of the business intelligence services deployment and they provide a partial on-demand solution to the problem of data warehousing design. Furthermore, none of these solutions offers an integrated *model-driven data warehouse* development approach and a web- based environment that supports this approach. Finally, there are few studies on open standards and open-source business intelligence tools integration in this context.

This chapter deals with two important aspects of *business intelligence-as-a-service* architectures: (i) the functional aspects to deliver business intelligence services covering the *model-driven data warehousing* services; and (ii) the technical aspects covering the recommended open industry standards and open-source tools. This chapter provides, also, several recommendations for data warehousing standards and development technologies. It, mainly, helps project managers and organizations involved in developing web-based business intelligence solutions.

We study the advantages of SaaS deployment model and some industry experiences (BusinessObjects, MicroStrategy, etc.) are presented. Then, a proposal for a common functional *business intelligence-as-a-service* architecture is described. We focus on the data warehousing projects management and components design services and we discuss the *model-driven data warehouse* approaches integration. Indeed, such approaches of data warehouse engineering provide several advantages and their integration in a SaaS environment seem more promising. So, we introduce the *model-driven warehousing* in the cloud using our approach [169] based on the *Model-Driven Architecture (MDA)* and the *2 Track Unified Process (2TUP)*. The MDA/2TUP based process for data warehouse engineering and its advantages are thus presented.

We study also the data warehousing standards and open-source business intelligence and web-applications development tools integration. Indeed, only a few solutions propose a standard and integrated data warehouse design framework like the *Common Warehouse Metamodel (CWM)*. Other MDA-compliant metamodels are useful, but they are not yet integrated as the *Ontology Definition Metamodel (ODM)*. In addition, little information is available on the technical architecture of data warehousing platforms. So, we give several recommendations concerning the standard industry tools, languages, and business intelligence APIs which can be integrated. Then, a technical architecture for the *business intelligence-as-a-service* based on the most popular open-source frameworks is described. The proposed technical architecture is based on *Java Enterprise Edition (JEE)* technologies using spring framework.

The chapter is organized as follows: the next Section gives a review of the *software-as-a-service* architecture. Section 7.3 describes the proposed architectures (i.e., functional and technical) for *business intelligence-as-a-service* including our approach for the *model-driven data warehousing-as-a-service*. Finally, Section 7.4 gives our conclusions and future work.

7.2 Software-as-a-Service Overview

The *software-as-a-service* is a model of architecture for software delivery where a software company publishes one copy of their software on the Internet. It allows individuals and companies (multi-tenant architecture) to "rent" it through a subscription model (*pay-as-you-go* model). The software company centrally operates, maintains and supports all its customers using this centralized service. The on-demand and *pay-as-you-go* models mean that in a SaaS architecture, costs are directly aligned with usage. The cost may increase as the usage of the application increases. Multi-tenant architecture model means that the physical backend hardware infrastructure is shared among many different customers but logically is unique for each customer [126]. Gartner[1] defines SaaS as software that's owned, delivered and managed remotely by one or more providers. The provider delivers an application based on a single set of common code and data definitions,

[1] http://www.gartner.com

which are consumed in a one-to-many model by all contracted customers anytime on a *pay-for-use* basis, or as a subscription based on usage metrics.

Lower *Total Cost of Ownership (TCO)* and better *Return On Investment (ROI)* constitute the major benefits of SaaS. Indeed, several factors contribute to making it considerably less expensive to implement a SaaS application than a traditional on-premises application. These factors include: (i) lower IT costs; since when you subscribe to a SaaS application, you avoid the overhead associated with implementing conventional software (installing and maintaining servers, etc.). (ii) Economies of scale; subscription costs for SaaS applications reflect the economies of scale achieved by multi-tenancy; as example on database is used to store all customers data, so, this makes the overall system scalable at a far lower cost. (iii) *Pay-as-you-go*; companies who subscribe to a SaaS application pay a monthly or annual subscription fee, sometimes depending also on the number of users or transactions. Others key variables such as simplicity, flexibility and accessibility constitute the advantages of SaaS deployment model.

In recent years, SaaS is gaining momentum with more and more successful adoptions. Several companies including some well known names have embraced this new model for software distribution. As SaaS providers, we cite Salesforce.com for on-demand *Customer Relationship Management (CRM)* services; PeopleSoft On-Demand from Oracle provides SaaS infrastructure for enterprise applications; ShareMinds that offers an on-demand *Enterprise Content Management (ECM)*. Google maps and apps (mail, docs, sites, etc.); and recently Microsoft, that announces *office web-apps*. Also, several business intelligence software solutions are exposed in a SaaS model. SAP BusinessObjects, the world's leading business intelligence software company launched a hosted on-demand platform [170] to deliver analytic and reporting functionality. PivotLink is one of the major actors of the on-demand business intelligence [171]. PivotLink offers a SaaS business intelligence solution covering data analysis, reporting and dashboards. MicroStrategy [172] is a leading business intelligence applications provider. MicroStrategy introduces in [173] the MicroStrategy platform architecture for hosted reporting, analysis and monitoring applications.

LogiXML [174]provides a web-based ad-hoc reporting, analysis, dashboard and *data integration (i.e., ETL)* applications. LogiXML solution is one of the few solutions that offer a fully web-based data integration environment. Talend one of

the most known open-source data integration solutions, starts recently the Talend on-demand project [175]. Talend on-demand is a centralized and shared repository hosted by Talend in order to consolidate Talend Open Studio metadata and project information in online and to facilitate collaboration, object and code reuse. However, it consists of a partial SaaS solution since the design of ETL package is made with Talend Open Studio, the on-demand service covers only uploading, sharing and collaboration functions. Finally, Pentaho Ad-Hoc Reporting [176] and OpenI [177] are examples of open-source solutions for web-based business intelligence deployment that allow build and publish reports, analyses, and dashboards. These solutions present many advantages derived in general from the advantages of the SaaS concept. But, not much information on the architecture of these on-demand solutions is provided. In addition, few solutions offer integrated platforms that cover all functional and technical aspects of the data warehousing architecture. Indeed, these tools focus only on the problem of the business intelligence services deployment and they provide a partial on-demand solution to the problem of data warehousing design. Furthermore, none of these solutions offer an integrated *model-driven data warehouse* development approach and a web-based environment that supports this approach.

The goal of our work is to provide an integrated *business intelligence-as-a-service* architecture covering the main data warehousing aspects (integration, analysis, reporting, design, etc.). In addition, we give a description of the role of each module in the architecture and recommendation for enterprise-class technologies to implement it. The study of a web-based data warehouse components design and derivation architecture using a model-driven approach (see Section 7.4) represents the key contribution of this chapter.

7.3 Architecture for Business Intelligence-as-a-Service

This section is divided into two parts: In Section 7.3.1 we propose a unified and complete *functional architecture* for *business intelligence-as-a-service*; and then an open-source based technical architecture for the corresponding functional architecture is explained. In Section we focus on the *model-driven data warehouse*

design component and we introduce the *model-driven data warehouse-as-a-service* architecture.

7.3.1 Functional and Technical Architectures

The proposed *business intelligence-as-a-service* architecture is defined through five main layers (figure 7.1): (i) the *technical resources* layer (data, applications & deployment tools); (ii) the *data warehouse design and management* layer; (iii) the *infrastructure administration and configuration* layer; (iv) the *core business intelligence services* layer (meta-data, integration, analysis, reporting, and delivery services); (v) and the *end-users access tools* layer (web portal, desktop, mobile). The *operating system and hardware* layer is not discussed in this chapter. In fact, it corresponds to the *Platform-as-a-Service (PaaS)* and the *Infrastructure-as-a-Service (IaaS)* concepts of the cloud computing which are out of scope of this work.

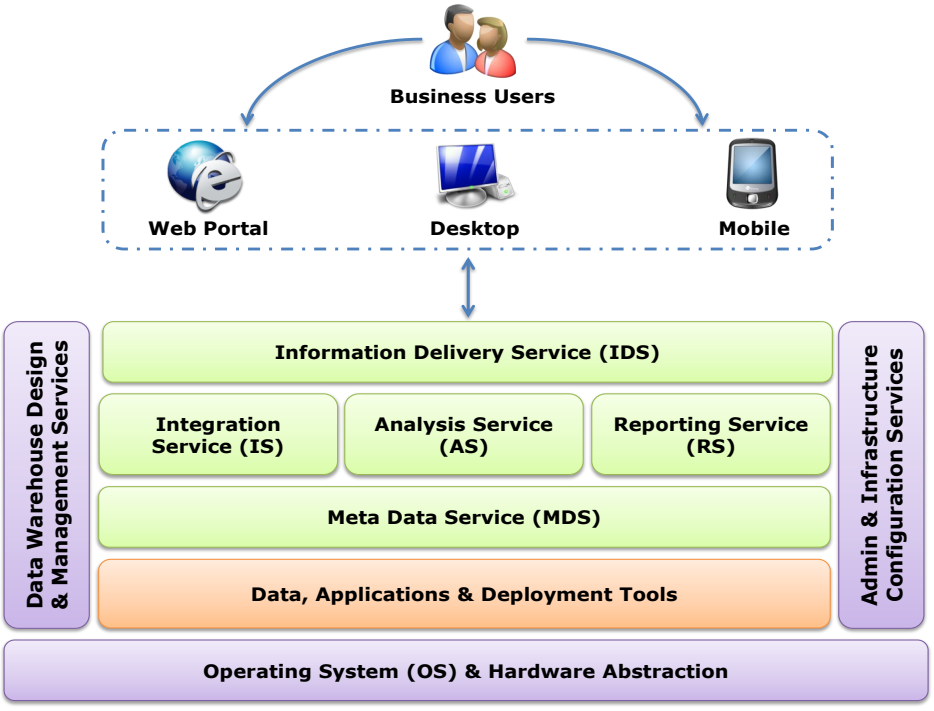

FIGURE 7.1: Business Intelligence-as-a-service Functional Aspects.

The *Technical Resources* layer (data, applications & deployment tools) contains the data warehousing components (database servers, ETL engine, analysis server,

reporting engines, etc.) used to deploy and to execute the designed data warehousing models. This layer corresponds also to data stored in the cloud, web data, and accessible web services useful for the platform. The integration of these data and the interoperability between all of these tools and APIs can be ensured using an *Enterprise Service Bus (ESB)* like apache ServiceMix [178] or JBossESB [179]. A detailed technical analysis is given in the next Section (technical architecture).

The *Design and Management* layer contains services to design data warehousing models (ETL jobs, multidimensional models, reports, etc.) and services to manage data warehouse development projects. It represents model-driven data warehouse services part of a web-based environment to design and manage data warehousing projects using our model-driven approach for data warehouse development presented in [169]. This layer offers an on-demand data warehouse design in order to ensure platform integrity (same technologies for all layers). It also simplifies the deployment and the access to the development environment for developers whom want to subscribe to this service. This reduces the installation time of development infrastructure and its costs.

The *Core Business Intelligence Services* layer represents the basic business intelligence applications used by business users. We identify five essential business intelligence services: (i) The *Meta-Data Service (MDS)*, which allows meta-data and business information definition to facilitate information sharing and exchange between all services. (ii) The *Integration Service (IS)*, which offers an ad-hoc way to define data integration jobs, jobs scheduling, etc. (iii) The *Analysis Service (AS)*, which allows definition of analysis data models (OLAP data cube), data cube visualization and navigation. (iv) The *Reporting Service (RS)* can be defined using existing reporting APIs like BIRT web viewer [180], or it can present same ad-hoc reporting functionalities; the current version of the RS implementation supports BIRT reporting and ad-hoc reporting. (v) The *Information Delivery Service (IDS)* is an abstraction level to support many client interfaces and technologies (web browser, mobile, office tools). It can be also presented as a web service for more flexibility to access the platform.

The infrastructure administration and configuration layer offers a web-based tool for administrators to manage user's accounts, to customize services configuration and to report same information on platform usage and performance. Finally, the end-users access tools layer contains client applications used to access the platform

and use its services. The web browser, web services, desktop/office tools, and mobile applications are examples of client-side technologies to support.

The proposed technical architecture (figure 7.2) is based on *Java Enterprise Edition (JEE)* technologies using Spring framework[2]. Spring is a very popular framework for enterprise Java applications development and integration. It allows easy platform configuration and extension using several reusable modules (JavaConfig, Security, Web, Integration, etc.). Spring framework and Spring development tools are provided by the SpringSource company. Since 2009, SpringSource is a division of VMware, a leader in virtualization and cloud solutions.

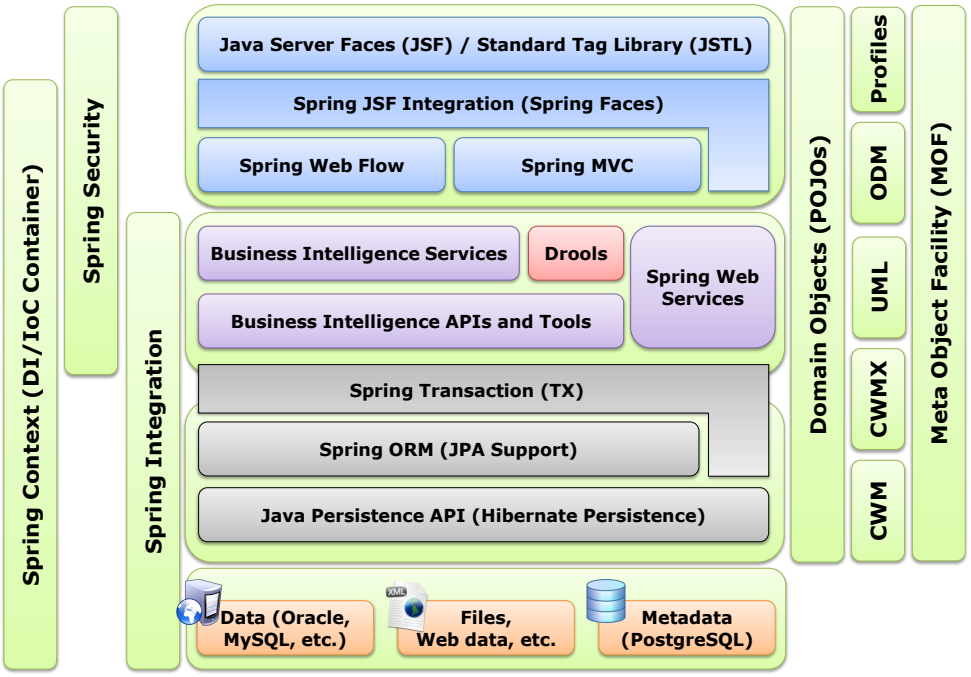

FIGURE 7.2: The Proposed Technical Architecture.

At the data layer, we use PostgreSQL to store metadata. PostgreSQL is one of the most mature and advanced open- source database management systems. For the customer data (used for reporting, analysis, etc.) the platform supports many databases such as Oracle, MSSQL, MySQL using configuration capabilities of Spring. To facilitate access to customer data (in general is stored in a cloud-based database), the integration of web data and the interoperability between remote database management systems and local data warehousing tools we need an ESB. For this purpose, we plan to use Spring Integration which provides a simple model

[2]http://www.springsource.org

for building enterprise integration solutions via many supported ESB-features. The Data Access layer allows a simplified way to access metadata and offers an abstraction level for the services layers to manipulate much heterogeneous persistent storage. The *Object-Relational Mapping (ORM)* tools, called also persistence APIs such as Hibernate or iBatis, are in general used to support the implementation of this layer. So, for persistence layer, we use the *Java Persistence API (JPA)* to define the ORM using Java metadata annotations and Hibernate is used as persistence provider for JPA.

The domain model layer contains domain objects that represent the business concepts of the information system. The domain objects are used by all layers can represent a large proportion of meta-data that are serialized into the data repository. Current domain model implements the *common warehouse metamodel* and CWMX *(a CWM eXtension)* metamodels. For the future, we plan to integrate other metamodels and profiles as the *ontology definition metamodel*. The implementation of these meta-models is based on the *meta-object facility* meta-metamodel. More details about this layer (also called the framework layer) and the packages used are provided in next section.

Spring Integration, *Spring Security*, *Spring Web Services* and *Spring Context* represent the main configuration layers of the platform. Spring Integration supports the well-known *Enterprise Integration Patterns* and offers many ESB-features. So, it simplifies integration of existing business intelligence tools, the access to remote customer databases and interoperability through java messaging services. *Spring Security* is a highly customizable, extensible authentication and authorization framework for securing Spring-based applications. *Spring Web Services* is used to expose business intelligence services on the web. Finally, *Spring Context* offers automatic mechanisms to create objects and configuration (defines the ApplicationContext) via *dependency injection*.

The *business rules* brick plays an important role in a service oriented infrastructure and any business intelligence system (essentially performance management). Indeed, a SaaS platform is shared by several customers that have different business processes, the definition of a business rules engine is essential for the orchestration of services. The *Business Process Management (BPM)* defines the process logic while the *Business Rules Management (BRM)* implements the decision logic. Thus, we recommend *Drools* [181] an open-source business rules management tools that

can be easily integrated with *Spring Context*. For the presentation layer, we use *Java Server Faces (JSF)* technology. Sun Mojarra, and Spring Faces are the JSF libraries used to defines user interfaces. Finally, all services are configured with spring context and run under Apache Tomcat web server, a standard container for Java server technologies. An interesting survey of others open-source tools for business intelligence (including databases, ETL, analysis, etc.) is presented in [182].

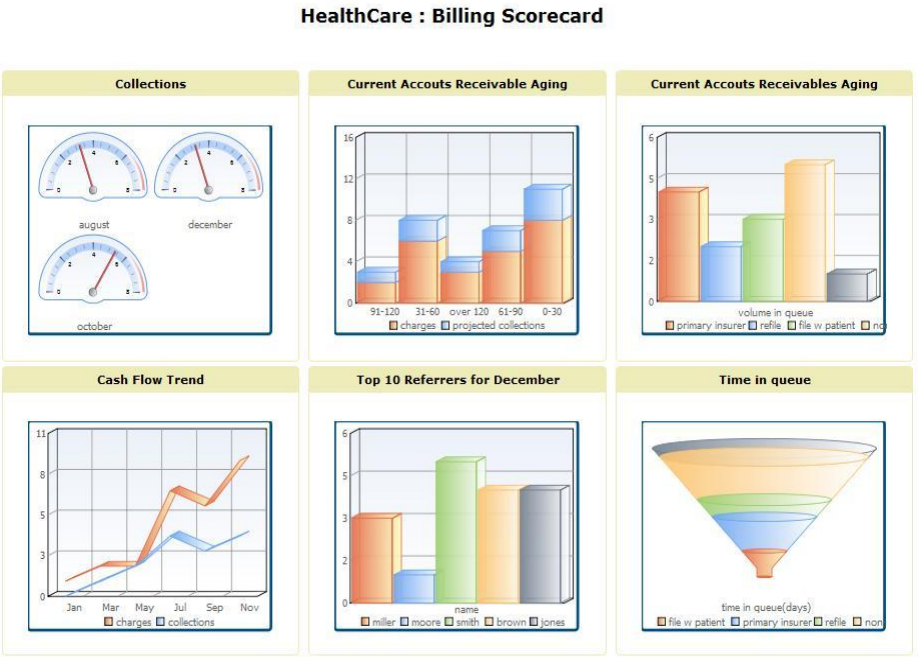

FIGURE 7.3: Dashboard Example for Healthcare Case.

We are worked on a prototype of a platform integrating the concepts discussed in this chapter. The project is named *ODBIS (On-Demand Business Intelligence Services)* [127]. A first release covering the administration, metadata and reporting services is available at [183]. The administration service provides a secure web-based application to manage authorities (privileges), roles, users, and groups. It offers also some search futures. The reporting services provides: (i) futures to manage report-groups and reports; (ii) a BIRT reporting module that allows upload and execute BIRT reports under the integrated BIRT Viewer; (iii) an ad-hoc reporting module which offers an easy way to defines chart reports, data-table reports and to build dashboards. The meta-data service provides futures to define data-sources and data-sets. DataSource objects provide a set of information (e.g., url, user and password) used to connect to database servers. DataSet objects are a SQL

query abstraction used by charts, data-tables and dashboards. Figure 7.3 shows a dashboard example using the ad-hoc reporting module for Healthcare case.

7.3.2 Towards Intelligent MDDW-as-a-Service

The *model-driven data warehouse services* module is part of the data warehouse management and design layer of figure 7.1. It represents an implementation of the proposed metamodeling architecture (mainly the *data warehousing design framework*). The services set contains: (i) the services for the *data warehousing engineering process* management based on the integration of 2TUP and the transformation process of MDA; (ii) the services for the data warehousing models design based on metamodeling architecture of MDA; and (iii) the services for models deployment, exportation, and importation through meta-data (merged with the meta-data service).

We introduce the architecture of a data warehouse development environment that implements our unified method, in order to provide a complete support for data warehouse engineering approaches and to contribute to improve current data warehouse development tools. This data warehousing design environment is called MDDWS for *model-driven data warehouse services*. MDDWS is a web-based *model-driven data warehouse* development environment and it is developed under the ODBIS platform [127]. The MDDWS application offers an innovative architecture to deliver on-demand model-driven data warehouse design services that reduce time and costs of data warehouse components implementation. The *model-driven data warehouse services* stack (in figure 7.4) includes the data warehousing project management service (methodology layer), the data warehousing design service, and the data warehousing refinement and deployment services. The project management service covers project steps definition and control, and project team roles definition. We study the integration of existing web-based and open-source project management tool like dotProject [184].

The *data warehousing design service* allows web-based diagramming features. Indeed, interactive and rich web solutions (Adobe AIR, Microsoft Silverlight, AJAX, etc.) are the key technologies to provide these features. Currently, we study existing web-based diagramming tools: several products with different technologies options (Ajax, Flex, and Silverlight) are provided by yWorks [185]. Gliffy [186] offers

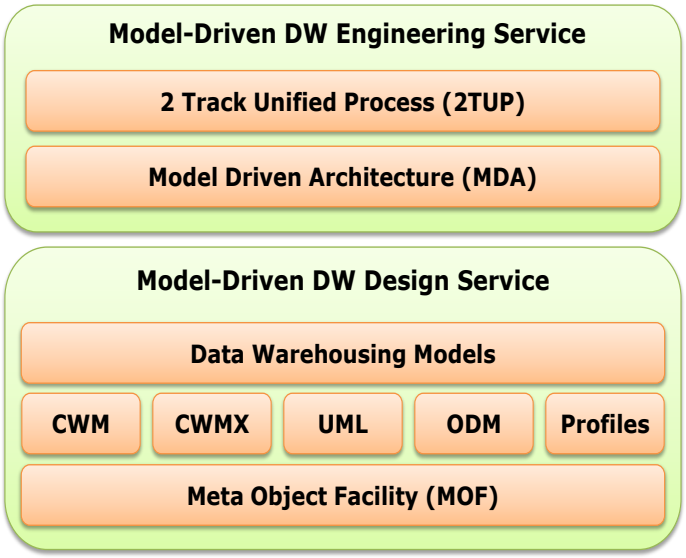

FIGURE 7.4: Model-Driven Data Warehouse Services Stack.

online modelling solution for different diagrams types (Flowchart, Network, UML, Business Process, etc.). Gliffy is based on OpenLaszlo [187] an open source platform for the development and delivery of rich Internet applications. MxGraph [188] is a JavaScript library that uses built-in browser capabilities to provide an interactive drawing and diagramming solution. MxGraph allows several modelling options: Graph, Database, Org Chart, and Workflow. Finally, JS-Graph-IT [189] and Jalava [190] are open-source JavaScript libraries to develop web-based diagram editors.

7.4 Summary

This chapter has provided a study on the benefits of the *software-as-a-service* model for the design of business intelligence architectures. First, we have defined current data warehousing and business intelligence applications deployment issues; we discussed some of the most important related environments. Then, we have described the proposed architecture to support common on-demand business intelligence services (functional architecture); we introduce *model-driven data warehouse-as-a-service* based on our approach to support *model-driven data warehousing* (that uses MDA and 2TUP); and we have defined a flexible multi-layered technical

architecture based on standard Java Enterprise with Spring and the most popular open-source tools.

Concerning core business intelligence services, we work to improve current realized services: administration, reporting and metadata. We will start soon the development of the integration and analysis services, the development of *model-driven data warehouse services* and security improvements. For the future, we plan to study other dimensions of the cloud-computing in order to improve scalability and configurability of data warehousing platforms. Therefore, we explore the *Platform-as-a-Service (PaaS)* and the *Infrastructure-as-a-Service (IaaS)* concepts. For example, we study the *VMware vFabric Cloud Application Platform* [191], a solution for building, running, and managing Spring applications in the cloud. We will explore also an advanced distributed data management platform like GemFire [192].

Chapter 8

Future Research Directions

Regarding *model-driven data warehouse* automation, we will experiments the case when a business goals model is considered during transformations in our future work. For example, the derivation of the MDPIM *(Multidimensional Platform Independent Model)* from the pair (DSPIM, MDCIM), where MDCIM *(Multidimensional Computation Independent Model)* defines the organisation requirements/goals. We plan also to extend the approach to new application domains that provide a large dependency-graph (e.g., the *Extraction, Transformation and Loading (ETL)* process in the data warehousing architecture). Then, we plan for an extension of the proposed *model-driven architecture* and the conceptual transformation learning framework to knowledge engineering seems also an interesting and a challenging future work. For example, a recent work in [193] proposes an MDA approach to knowledge engineering that addresses the problem the mapping between CommonKADS knowledge models and *Production Rule Representation (PRR)*. Below we discuss others important directions in the fields of *Data Warehouse Performance Management* and *Semantic Model-Driven Policy Management* that we consider interesting.

The book entitled *DW 2.0: The Architecture for the Next Generation of Data Warehousing* [194] describes an architecture of the second-generation data warehouses. It presents also the differences between DW 2.0 (introduced as the new generation) and the first generation data warehouses. Authors start by an overall architecture of DW 2.0 and give its key characteristics. Then, they present the DW 2.0 components and the role of each component in the architecture. The proposed

architecture focuses on three key features: (i) the data warehouse repository structure (organization on four sectors: interactive, integrated, near line, and archival); (ii) unstructured-data integration and organization; and (iii) unified meta-data management. We confirm that unstructured data and web-data integration constitutes a future challenge. Thus, we support semantic-based approaches [195] for web-data integration and data warehouses contextualization with documents [196]. This kind of approaches will probably represent the essential part of what we call the "DW 3.0 architecture". The DW 3.0 concept (or content data warehouse) is a unified architecture that includes the data warehouse, the document warehouse and the web warehouse. According to authors [194], DW 2.0 represents the way corporate data needs to be structured in support of web access and *Service Oriented Architecture (SOA)*. For this purpose, an effort is provided by [197]. So, we believe that *business intelligence-as-a-service* platforms need a more efficient, personalized and intelligent web-services discovery and orchestration engines. The perfect marriage of SOA/SaaS infrastructures is a key issue to design future on-demand business intelligence services.

In [198], the author studies the evolving state of data warehousing platforms and gives options available for next generation data warehousing. The *options* include concepts presented in this chapter: SaaS and open-source business intelligence tools. It presents also many important features such as: real-time data warehousing, data management practices and advanced analytics. In [199], author discusses other remaining challenges to extend traditional data warehouse architecture. The focus is mainly given for the data warehouse full-scale problem (world warehouse) and the privacy in data warehousing systems. The metadata management for *business intelligence-as-a-service* infrastructures and cloud-based databases will be also an interesting research direction. Indeed, current standards and models should be extended in this new architectural context. Finally, we believe that our proposal for *model-driven data warehouse-as-a-service* is a key characteristic to provide future data warehouse design in the cloud.

The purpose of the *Common Warehouse Metamodel (CWM)* specification is to define a common interchange specification for metadata in a data warehouse. This definition provides a common language and metamodel definitions for the objects in the data warehouse. CWM describes a format to interchange metadata, but lacks the knowledge to describe any particular type of interchange. The need to define

the context of a CWM interchange was discovered when the CWM co-submitting companies produced the CWM interoperability showcase. In order to make an effective demonstration of CWM technology, the participants needed to agree upon the set of metadata to be interchanged. In this context, the *object management group* propose the *CWM Metadata Interchange Patterns (CWM MIP)* specification in order to address the limitations of the CWM. The purpose of CWM MIP specification is to add a semantic context to the interchange of metadata in terms of recognized sets of objects or object patterns. We will introduce the term *Unit of Interchange (UOI)* to define a valid, recognizable CWM interchange. From this information, a user of CWM, working in conjunction with CWM MIP, should be able to produce truly interoperable tools. CWM MIP augments the current CWM metamodel definitions by adding a new metamodel package. This new metamodel will provide the structural framework to identify both a UOI and an associated model of a pattern, and providing the necessary object definitions to describe both. For future work, we will study in detail the CWM MIP in order to define new features that can improve the proposed architecture.

The *Ontology Definition Metamodel (ODM)* has been used as a basis for ontology development as well as for generation of OWL *(Web Ontology Language)* representations. The specification defines a family of independent metamodels, related profiles, and mappings among the metamodels corresponding to several international standards for ontology and *Topic Maps* definition, as well as capabilities supporting conventional modelling paradigms for capturing conceptual knowledge, such as entity-relationship modelling. The *ontology definition metamodel* is used for ontology development and analysis on research in *context-aware* systems [200–202]. As part of the *object management group* metamodeling architecture (ODM is a MOF-compliant metamodel), the ODM enables using *model-driven architecture* standards in ontological engineering. The ODM is applicable to knowledge representation, conceptual modelling, formal taxonomy development and ontology definition, and enables the use of a variety of enterprise models as starting points for ontology development through mappings to UML and MOF.

The software engineering community is beginning to realize that security is an important requirement for software systems, and that it should be considered from the first stages of its development. Unfortunately, current approaches which take security into consideration from the early stages of software development do

not take advantage of *model-driven development*. Security should definitely be integrated as a further element of the high-level software system models undergoing transformation until the final code generation [203]. Thus, *model-driven development* for secure information systems, *model-driven security* and *dynamic refinement of security policy* are a new promising research direction. Another important aspect related to this, is semantics. In fact, dynamic refinement of security requires applying innovative semantic reasoning techniques to security metrics and contextual information. We consider this as an interesting problem for model- driven approach and its application. Also, our participation in Predykot European project[1] was an opportunity to identify new areas of application of our work and to discuss innovative ideas around several topics.

Predykot (Policies REfined DYnamically and Kept On Track) will provide an innovative, modular and consistent eco- system of software modules to dynamically refine a security policy and to ensure that it remains efficient whatever changes occur to it: administrative, contextual etc. It intends to shift the focus of security policy management from basic operational improvements to critical intelligence for business process improvement. Intelligent mechanisms are indeed necessary to ensure that a policy remains efficient in time, to take contextual information into consideration to dynamically refine the policy, with the objectives of governance, risk management and compliance. Predykot is targeting the markets where security is crucial, such as cloud computing, large and mission- critical systems, identity & access management, professional mobile radio, mobile near field communication equipments and services.

We seek to answer the question of *how to provide intelligent methods and techniques to dynamically refine security policy using the contextual information?* It addresses different new issues such as: advanced semantic reasoning, recent security standards integration and deployment of the approach in several application domains. The project covers also the extension and the improvement of several existing approaches such as: related-policies management, context-aware and smart nodes, and the improvement of policies refinement techniques. Modularity, adaptability and consistency will be the main features of the proposed architectures, methods and standards. Thus, the dynamic policy refinement life-cycle proposed in this document aims to ensure these important aspects. This proposal addresses the

[1] http://www.itea2- predykot.org/

use of the *model-driven development* for security policies derivation and standards recommendation for policies definition and representation. The general research area related to this proposal is called *model-driven security*. The idea is derived from the research challenges discussed in [203, 204], where authors briefly explore some of the important related works [205–210] and standards [211, 212] to this context. So, an adapted architecture using a model-driven approach and that provide our vision of the problem is described.

Regarding the problem description, the "Semantics and Reasoning" component will represent the core workflow element of the proposed architecture. This component is responsible of semantic policy adaptation (or reactions generation) based on policy changes and users activity. The project description also implies that the overall workflow contains a human approval step. Then, when we talk about automatic derivation (or generation), the application of the *model-driven architecture* approach is directly possible. In the case of the proposed framework, the terms are around model-driven security or model-driven policy. The aim of our proposal is to take advantages of the semantic-driven approaches and the model-driven approaches. So, considering the "Semantics & Reasoning" component definition and the model-driven engineering definition, several questions arise: (i) how allow interoperability between the reasoning process (reasoning mechanisms) and the MDA process (policies generation mechanisms)? (ii) Which representation languages are available to define policy in order to ensure the integrity of the entire process? And (iii) in more general, how provide a unified semantic model-driven and reasoning approach? To address this problem, we propose the use of several industry standards covering the semantics, security, and the MDA aspects. In addition, based on our experience on MDA-compliant architectures, a common approach showing the use of these standards is defined.

The ontology is the central concept of any semantic-driven development. The *ontology definition metamodel* specification [212] is an *object management group* standard (MDA-compliant and extensible metamodel) that allows model-driven ontology engineering. It provides standard profiles for ontology development in UML and enables consistency checking, reasoning, and validation of models in general. The *ontology definition metamodel* include five main packages: At the core are two metamodels that represent formal logic languages: *DL (Description Logics)* which, although it is non-normative, is included as informative for those

unfamiliar with description logics and *CL (Common Logic)*, a declarative first-order predicate language. There are three metamodels that represent more structural or descriptive representations that are somewhat less expressive in nature than CL and some DLs. These include metamodels of the abstract syntax for RDFS, the *Ontology Web Language (OWL)* and *Topic Maps (TM)*. Thus, the ODM standard is highly recommended in an MDA-based process because it is conform to MOF metamodeling architecture. In this case, is important to have a generic ontologies representation in order to: (i) facilitate the transformation of semantic models using the MDA-enabled frameworks; and (ii) ensure interoperability between the different components/tools (including the reasoning engine and the transformation engine).

Security policy definition is very important in organization because it should cover many aspects. Several security specifications are proposed. However, the *object management group* security specifications [211] catalog remains the most comprehensive and extensible. The OMG catalog mainly contains: (i) the *Authorization Token Layer Acquisition Service (ATLAS)* specification which describes the service needed to acquire authorization tokens to access a target system using the CSIv2 protocol; (ii) The *Common Secure Interoperability Specification, Version 2 (CSIv2)* which defines the *Security Attribute Service* that enables interoperable authentication, delegation, and privileges; (iii) the *CORBA Security Service* which provides a security architecture that can support a variety of security policies to meet different needs (identification and authentication of principals, authorization and infrastructure based access control, security auditing, etc.); (iv) the *Public Key Interface (PKI)* specification which provides interfaces and operations in CORBA IDL to support the functionality of a PKI (issuance, management, and revocation of digital certificates); and (v) the *Resource Access Decision Facility (RAD)* specification which provides a uniform way for application systems to enforce resource-oriented access control policies. The integration of these standards in the final architecture allows more quality in policy representation and a more unified, interoperable model-driven security approach with the *model-driven architecture*. Note also that the proposed approach is open for integration of others security specifications and profiles.

In the proposed approach we focus on the "Semantics & Reasoning" module. Thus, we discuss some details of the "Semantics & Reasoning" flow based on our main

objectives (i.e., adaptability, consistency) and the model-driven support that we add. Figure 8.1 illustrate the workflow that we explain below.

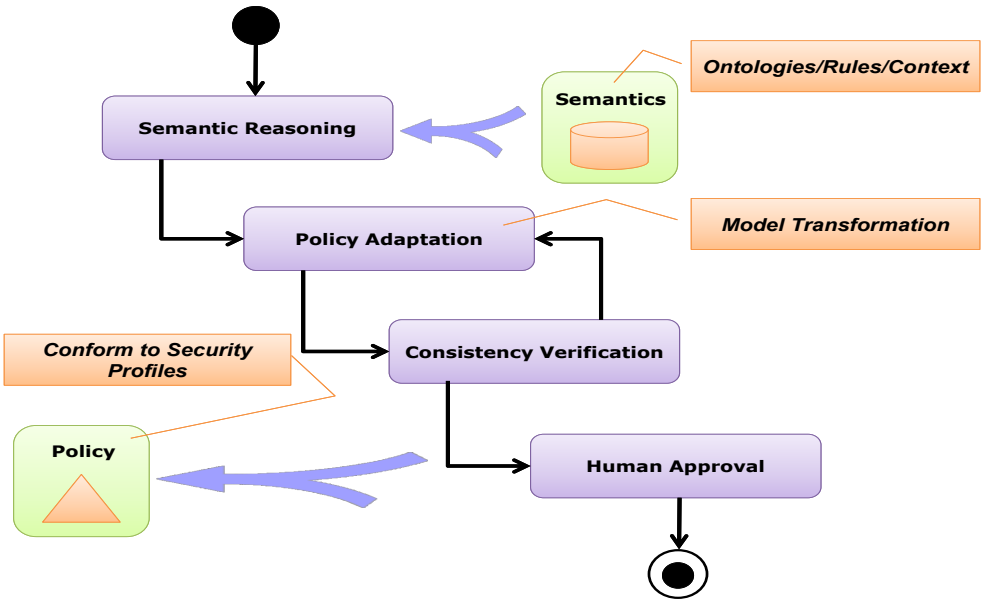

FIGURE 8.1: Semantic Model-Driven Policy Adaptation.

Semantic Reasoning: the policies adaptation flow starts by a semantic analysis step. This step considers at the input several semantic conceptions: context, user's activities/profiles, auditing, etc. This information is represented in general by ontologies and/or rules. Then, as discussed above, the ontologies models are conform to the *ontology definition metamodel* metamodel.

Policy Adaptation: this step corresponds to the *model-driven architecture* transformation process. Based on the results of semantic reasoning step, it selects the appropriate transformation from transformations repository and apply it on current policy model. Thus, information given by the reasoning engine is automatically projected on policy by the transformation engine. In a model-driven context this supposes that policy models are conforming to the specifications cited above (ATLAS, RAD, etc.) or other profiles already recognized by the transformation. This allows for more interoperability between reasoning and transformation engines.

Consistency Verification: it is a machine approval foregoing the human approval. During this step the system checks if the main security constraints are violated or not. If a constraint is violated, the transformation process must be re-executed

with new parameters (information about the violation is also audited), else the system waits for human approval.

Human Approval: it can be a very important step in the approval workflow mechanisms of some organizations. Indeed, in the detailed requirements we stress this sentence: *automatic reactions on a policy might not be accepted by security officers, or even more simply might not be suitable in real-life.*

Finally, in this proposal we discuss briefly the application of the *model-driven engineering* for policy adaptation in the respect of the problem description. So, based on our contributions to improve model-driven methodologies, an adapted *model-driven security* approach is defined. Then, we provide also recommendations for specification (perspectives to integrate new policy languages) to define policy. Future work will provide a detailed analysis, more research and improvements around the proposed approach.

Chapter 9

Conclusion

This thesis studies a complex problem at the crossroad of several research fields. We study the *model-driven data warehouse* engineering and its automation using machine learning techniques. The automation of *information systems* engineering and, in particular, *decision support systems* remains a difficult and a challenging task. Indeed, the *model-driven warehousing* requires a transformation phase of the models that necessitate a high level of expertise and a large time span to develop. The main goal of the proposed framework is to automatically derive the transformation rules to be applied in the *model-driven data warehouse* process. This process of transformation concerns for instance the creation of an OLAP cube in business intelligence from an UML diagram of the considered application. The proposed solution allows the simplicity of *decision support systems* design and the reduction of time and costs of development. First, we use the *model-driven engineering* paradigm for data warehouse components development (as first level of automation). Then, we use *inductive relational learning* techniques for automatic model transformation generation (as second level for automation). Finally, we propose a deployment solution for *business intelligence-as-a-service* architecture based on promising architectural model and open technologies in order to reduce time and costs of the infrastructure installation.

Through the modelling step, we contribute in isolating steps where it is necessary to induce transformation rules; in identifying the metamodels used to define the input/output models of these transformations and in designing a conceptual framework for transformations learning that use adequate representation language. We

have focused on effective modelling of the *model-driven data warehouse* architecture in order to simplify machine learning framework integration. This architecture allows to effectively deploying the application, with respect to standards and data warehousing requirements in organisations. This modelling step, considered as modelling bias (or architecture bias) is important to manage these risks and make efficient the task of transformations learning. In this step, the *model-driven data warehouse* framework is extended by *Inductive Logic Programming (ILP)* capabilities in order to support the expert in the transformation process. The ILP offers a powerful representation language and the given results (i.e., transformation rules) are easy to understand. We have focused on providing an optimized representation of the language bias (or declarative bias) based on *unified modelling language* and the *common warehouse metamodel* standards. This declarative bias addresses the reduction of CWM-UML problem into ILP and aims to restrict the representation to clauses that define best the transformation rules.

Through the learning approach step, we contribute to the definition of the optimal way to learn transformation rules in model-driven frameworks. Indeed, dependencies exist between transformations within the *model-driven data warehouse* architecture. We investigate a new machine learning methodology stemming from the application needs: *Learning Dependent-Concepts (DCL)*. We used in our experiments the well known Aleph ILP system, because of its ability to handle rich background knowledge, made of both facts and rules. Aleph follows a top-down generate-and-test approach. It takes as input a set of examples, represented as a set of *Prolog* facts and background knowledge as a Datalog program. It also enables the user to express additional constraints C on the admissible hypotheses. Aleph tries to find a hypothesis $h \in \mathcal{L}_h$, such that h satisfying the constraints C and which is complete and partially correct. We used Aleph default mode: in this mode, Aleph uses a simple greedy set cover procedure and construct a theory H step by step, one clause at a time. To add a clause to the current target concept, Aleph selects an uncovered example as a seed, builds a most specific clause as the lowest bound of its search space and then performs an admissible search over the space of clauses that subsume this lower bound according the user clause length bound. In the next section, we show the reduction of the source-model, the target-model and the mapping between them into an ILP problem. The DCL method is implemented using Aleph and it is applied to our transformation learning problem. We show

that the DCL approach gives significantly better results and performances across several experimental settings.

Through the implementation step, we contribute in improving and optimising the deployment of data warehousing components and services. We have studied the functional aspects to deliver business intelligence services covering the model-driven data warehousing services; and the technical aspects covering the recommended open industry standards and open-source tools. A common functional and technical *business intelligence-as-a-service* architecture is then described. This proposal extends the data warehousing process automation because such deployment architecture (i.e., web-based services) allows for speed and agile business intelligence delivery.

Appendix A

Context of Work

This work is funded by the *ANRT (National Association for Research and Technology, http://www.anrt.asso.fr)* [1], Grant CIFRE–1341/2008 and *Intelligence Power Company*. The CIFRE is for *Industrial Conventions for Training through Research*. The thesis work is also prepared in the *LIPN (Computer Science Lab of Paris-Nord University)* and *Intelligence Power Company*. The LIPN (http://www-lipn.univ-paris13.fr) is associated to the CNRS (UMR 7030). The LIPN researches deal with the reasoning automation around strong axes of *Combinatorics*, *Combinatorial Optimization*, *Fundamental Computer Science* and *Artificial Intelligence*. These works are especially based on competences in *Algorithmics, Logic, Natural Language, and Machine Learning*. The Lab is structured into five teams (A3, AOC, CALIN, LCR, and RCLN) whose research themes show numerous meeting points. This work is supervised by Céline Rouveirol and Aomar Osmani, members of *Machine Learning and Applications (A3)* team. Intelligence Power (http://www.intelligencepower.com) is created with a vision to provide innovative business intelligence solutions. It mainly focuses on research and development of intelligent decision-making platforms. Thus, *decision-support systems* development and the use of new approaches and applications (e.g., *machine learning*, *model-driven engineering*) is the key activity of the company. This thesis is a very important step to achieve R&D goals of Intelligence Power. The results from this work will represent the key features of future enterprise decision-making tools and will form part of Intelligence Power commercial offer.

[1] *The French ANRT (Association Nationale de la Recherche et de la Technologie.)*

The advent of the *model-driven development* (also known as *model-driven engineering*) paradigm in 2003 [24], was a significant change in the way of developing software and information systems. It represents a promising approach to support software development practices [17, 52, 53]. The approach is mainly based on models, meta-models, and model transformation designs. Indeed, the model-driven strategy encourages the use of models as central element of development. The models are conformed to metamodels and the transformations rules are applied to refine them. In the case of database systems, this approach includes but not limited to automatic generation of the logical model from the conceptual model; and the physical model from the logical by applying transformations. Then, the *model-driven data warehouse* represents approaches [27, 28, 152] that align the development of the data warehouse with a general *model-driven engineering* paradigm.

"Nothing is lost, nothing is created, all is transformed." The transformations are essential for each model-driven process. And, a simple model transformation consists in defining the mapping between elements of a source model (i.e., the input parameter of the transformation) and a target model (i.e., the resulted output of the transformation execution). In this context, the *Query-View-Transformation (QVT)* standard plays a central role, since it allows for the specification of model transformation rules. However, transformations design requires highly skilled experts, which results in additional cost and time for *model-driven warehousing* development. We propose, in the framework of model-driven data warehouse, an approach that transforms a knowledge base (of previous projects experiences) into model transformation rules. The obtained rules are applied to transform schemas of future data warehousing projects (e.g., conceptual-to-logical, logical-to-physical). In the case of Extraction, Transformation, and Loading (ETL) model, the deployed application transforms operational data into multidimensional data.

Appendix B

Study of Standards and Practices

This appendix gives an overview on the *Unified Modelling Language (UML)*, the *Common Warehouse Metamodel (CWM)* and the *Query-View-Transformation (QVT)* standards that are used to define the proposed *data warehousing design framework*. It presents also the *Two Track Unified Process (2TUP)* standard that is used to define the proposed *data warehouse engineering process*. The *unified modelling language*, the *common warehouse metamodel* and the *query-view-transformation* specifications are maintained by the *Object Management Group, Inc. (OMG)*. In the reference metamodeling architecture, the three standards appear in the $M2$ level (metamodels) and are conform to the *Meta-Object Facility (MOF)* meta-metamodel. Figure B.1 gives examples of instances of these metamodels and details about each metamodel are provided in the following paragraphs.

The *object management group* is an international organization supported several members, including information system vendors, software developers and users. Founded in 1989, the OMG promotes the theory and practice of object-oriented technology in software development [213]. OMG's objectives are to foster the growth of object technology and influence its direction by establishing the *Object Management Architecture (OMA)* [214]. The OMA provides the conceptual infrastructure upon which all OMG specifications are based. The *meta-object facility* is an OMG metadata interface standard that can be used to define and manipulate a set of interoperable metamodels and their instances (models). The MOF also defines a simple meta-metamodel (based on the *unified modelling language* with

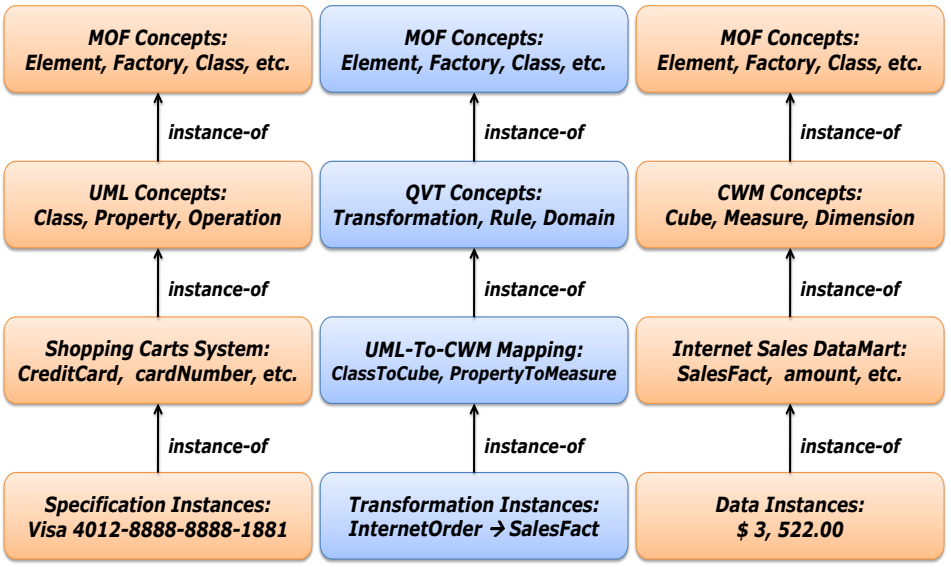

FIGURE B.1: Instances Examples of UML, QVT and CWM Metamodels.

sufficient semantics to describe metamodels in various domains starting with the domain of object analysis and design.

The *unified modelling language* is a general-purpose modelling language for specifying, constructing, and documenting the artefacts of systems. The UML specification is defined by using a metamodeling approach that adapts formal specification techniques. The InfrastructureLibrary for the UML metamodel contains the packages Core and Profiles. The Core package (i.e., UML CORE) is the central reusable part of InfrastructureLibrary, and is further subdivided as shown in the figure below. The UML CORE package is subdivided into a number of interrelated sub-packages: PrimitiveTypes, Abstractions, Basic, and Constructs. The figure B.2 shows relationships between UML CORE sub-packages. We refer the reader to Appendix C, for details concerning UML CORE.

The package PrimitiveTypes is a simple package that contains a number of predefined types that are commonly used when metamodeling, and as such they are used both in the infrastructure library itself, but also in metamodels like MOF and UML. The package Abstractions contains a number of fine-grained packages with only a few meta-classes each, most of which are abstract. The purpose of this package is to provide a highly reusable set of meta-classes to be specialized when defining new metamodels. The package Constructs also contains a number of fine-grained packages, and brings together many of the aspects of the Abstractions.

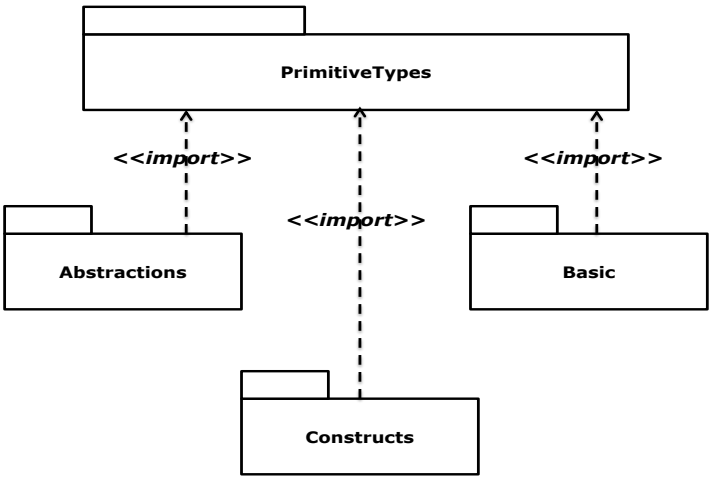

FIGURE B.2: The Main Sub-Packages of UML CORE.

The meta-classes in Constructs tend to be concrete rather than abstract, and are geared towards an object-oriented modelling paradigm. Looking at metamodels such as MOF and UML, they typically import the Constructs package since the contents of the other packages of Core are then automatically included. The package Basic contains a subset of Constructs that is used primarily for *XMI (XML Metadata Interchange)* purposes. The Profiles package of UML contains the mechanisms used to create profiles of specific metamodels, and in particular of UML. This extension mechanism subsets the capabilities offered by the more general MOF extension mechanism.

The main purpose of *common warehouse metamodel* is to enable easy interchange of warehousing and business intelligence metadata between warehouse tools, warehouse platforms and warehouse metadata repositories in distributed heterogeneous environments. Hence, the CWM specification [26] addresses the metadata interchange requirement of the *object management group* repository architecture specific to the data warehousing domain. CWM uses MOF as its meta-metamodel. CWM uses UML as its graphical notation, and defines a base metamodel; that is, the *CWM Object Model* that is consistent with the UML CORE metamodel [215]. The CWM supports a *model-driven approach* to metadata interchange, in which object models representing tool- specific metadata are constructed according to the syntactic and semantic specifications of a common metamodel. The *common warehouse metamodel* is structured as a collection of related metamodels (or sub -metamodels), each metamodel occupying its own package, and with a minimal

number of inter-package dependencies. The CWM is also an extension of the *unified modelling language* for data warehousing and business intelligence. Thus, all CWM packages are dependent upon the core packages of the UML, which provides a syntactic foundation for CWM. The CWM metamodels, and any models based on them, are defined in UML. The CWM is also a *domain-specific extension* of the OMG metamodeling architecture [24], and as such, implicitly supports the MOF, UML, and XMI standards. Although CWM has certain *compatibilities* with various other standards (as outlined in subsequent sections), these compatibilities should be regarded as touch points for mapping or integration; they do not represent dependencies of any kind. CWM is not dependent upon any standards outside of those of the metamodeling architecture.

The *query-view-transformation* standard plays a central role in the the *model-driven development*, since it allows for the specification of model transformation rules. Related work [27, 28] have tried in 2008 to adapt the model-driven approach for the development of data warehouses using the *Model- Driven Architecture - MDA (a standard implementation of the model-driven engineering)* and the QVT. For example, the approach presented in [28] describes derivation of *OnLine Analytical Processing (OLAP)* schemas from *Entity-Relationship (ER)* schemas. The source and target models are respectively conform to ER and OLAP metamodels from the *common warehouse metamodel*. Authors describe how an ER schema is mapped to an OLAP schema and provide, also, a set of *query-view-transformation* rules (e.g., EntityToCube, AttributeToMeasure, RelationShipToDimension, etc.) to ensure this. Model transformations are a key technique for automatic management of modelling artefacts. The *query-view-transformation* represents a specification for model transformations still under development by the *object management group*. The QVT specification has a hybrid declarative/imperative nature, with the declarative part being split into a two-level architecture (see figure B.3).

The declarative parts of this specification are structured into two-layer architecture. The layers are:

- A user-friendly Relations metamodel and language that supports complex object pattern matching and object template creation. Traces between model elements involved in a transformation are created implicitly.

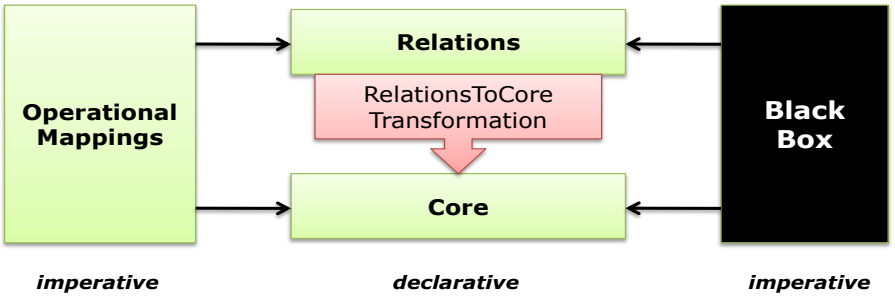

FIGURE B.3: Relationships Between QVT Metamodels.

- A Core metamodel and language defined using minimal extensions to *EMOF (Essential MOF)* and *OCL (Object Constraint Language)*. All trace classes are explicitly defined as MOF models, and trace instance creation and deletion is defined in the same way as the creation and deletion of any other object.

In addition to the declarative Relations and Core languages that embody the same semantics at two different levels of abstraction, there are two mechanisms for invoking imperative implementations of transformations from Relations or Core: one standard language, *Operational Mappings*, as well as non- standard *Black-box* MOF Operation implementations. Each relation defines a class that will be instantiated to trace between model elements being transformed, and it has a one-to-one mapping to an Operation signature that the *Operational Mapping* or *Black-box* implements.

The *two track unified process* is a software development process that implements the *Unified Process (UP)* and uses UML as a modelling language [216]. The 2TUP process (also called Y shaped process or Y lifecycle) answers to the constraints of change of the information systems that are subjected to two types of constraints: (1) the functional constraints and (2) the technical constraints. Thus, 2TUP is modelled by two branches (or tracks): The left branch makes an inventory of the functional needs and analyzes it, which produces a model focused on the needs of business users. The right branch contains a study of the technical needs and defines the technical architecture of the solution. The two branches are then merged into a medium branch which supports preliminary design, detailed design, coding, tests and validation steps.

The choice of 2TUP comes owing to the fact that the evolution of the 2TUP process is similar to the transformation process of the *model-driven architecture*. It

will simplify the mapping between the MDA concepts and 2TUP disciplines, and the integration of the MDA transformation process in our final process. In [217], authors present a software development process for MDA called M2T. The M2T process provides an implementation of the MDA approach while relying on a Y shaped development cycle. The left branch of the Y cycle corresponds to the PIM (*Platform Independent Model*), while the right one corresponds to what we call an explicit PDM representing the target platform. Authors propose the *Design Decision Metamodel (DDM)* to merge the PIM and the PDMs (*Platform Dependent Models*), in order to produce the PSM (*Platform Specific Model*). However, the DDM is not a standard and MOF-compliant language for models transformation such as the *Query-View-Transformation*.

Appendix C

UML CORE and CWM OLAP

The UML CORE package [25] is subdivided into a number of interrelated sub-packages: PrimitiveTypes, Abstractions, Basic, and Constructs. Figure C.1 shows the main elements of each sub- package and relations between them. Each sub-package is organised into one or more diagrams.

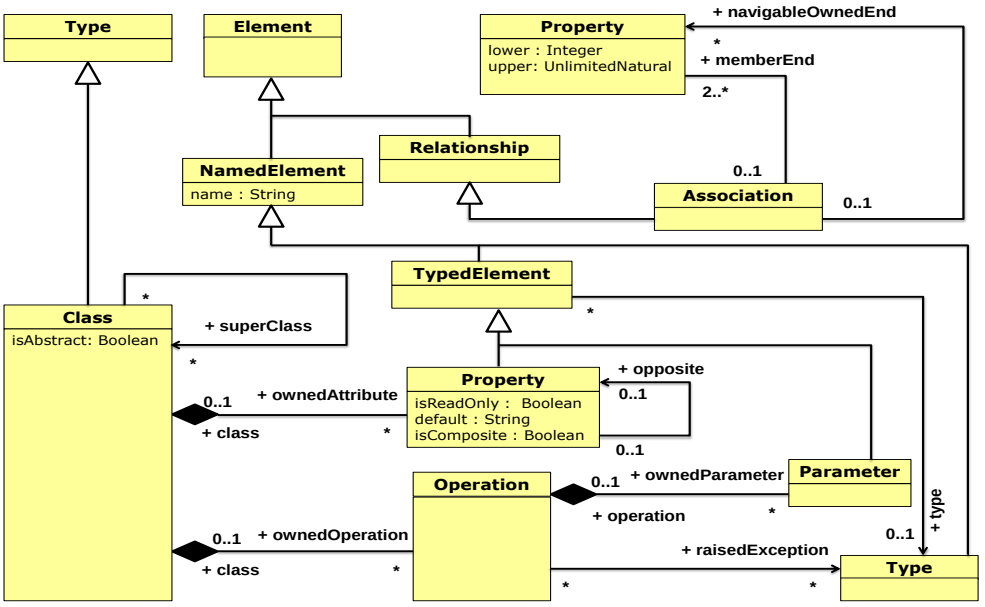

FIGURE C.1: Part of the UML CORE Metamodel.

For example, the meta-classes in the Basic sub-package are specified using four diagrams: Types, Classes, DataTypes, and Packages diagrams. The Types diagram defines abstract meta-classes that deal with naming and typing of elements. Thus, we find Element, NamedElement, Type and TypedElement meta-classes. The

Classes diagram defines the constructs for class-based modelling. So, as example we find Class, Operation, Parameter and Property meta-classes. However, Relationship and Association meta-classes are specified by the Classes diagram of the Constructs package. For instance, a Class includes a name (Class inherits indirectly from NamedElement). A Class also includes attributes, represented by the *ownedAttribute* role and operations, represented by the *ownedOperation* role.

The primary objective of the CWM OLAP package [26] is to define a metamodel of essential and common *Online Analytical Processing (OLAP)* concepts to most data warehousing systems. Figure C.2 gives the major classes and associations of CWM OLAP package.

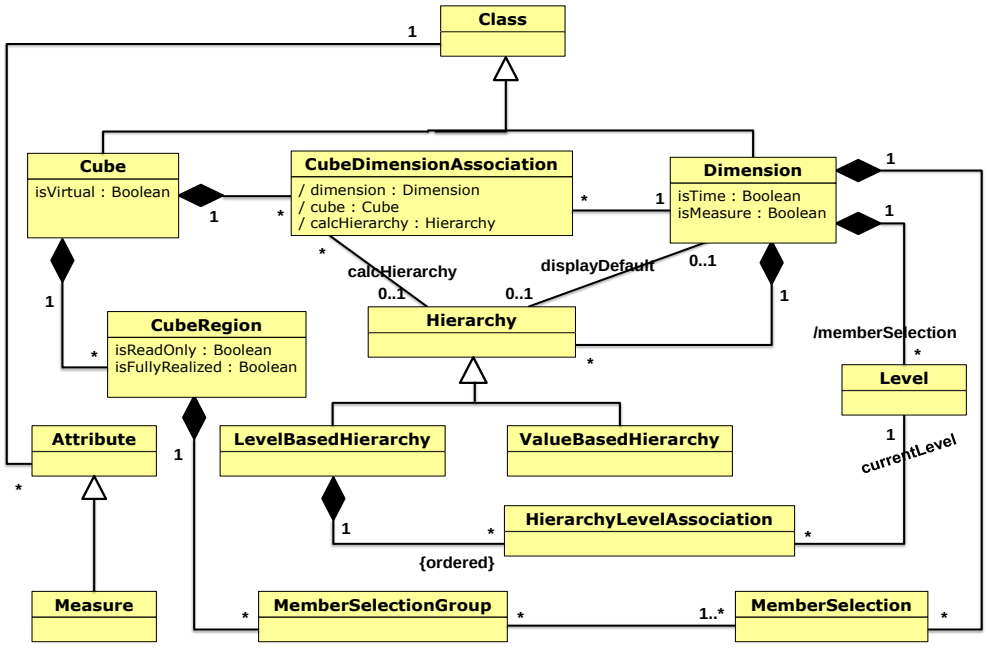

FIGURE C.2: Part of the CWM OLAP Metamodel.

For instance, a Cube is a collection of analytic values, that is, measures that share the same dimensionality. This dimensionality is specified by a set of unique dimensions (Dimension meta-class). The CubeDimensionAssociation meta- class relates a Cube to its defining dimensions. A Dimension has zero or more hierarchies. A Hierarchy is an organizational structure that describes a traversal pattern through a Dimension, based on parent/child relationships between members of a Dimension. The OLAP metamodel specifies two subclasses of Hierarchy: LevelBasedHierarchy class and ValueBasedHierarchy class. For example, the LevelBasedHierarchy

describes hierarchical relationships between specific levels of a Dimension (e.g., Day, Month, Quarter and Year levels).

Appendix D

Aleph System

The *A Learning Engine for Proposing Hypotheses (Aleph)* system was developed to be a prototype to explore ideas in *inductive logic programming* and was written Prolog. Since then, the implementation has evolved to emulate some of the functionality of several other ILP systems. The Aleph has a powerful representation language that allows representing complex expressions and simultaneously incorporating new background knowledge easily. Aleph also let choose the order of generation of the rules, change the evaluation function and the search order. Allied to all this characteristics the Aleph system is open source making it a powerful resource to all ILP researchers. The basic Aleph algorithm follows a very simple procedure that can be described in 4 steps [97]: (1) Select an initial example to be generalized. When there are not more examples, stop; (2) Construct of the more specific clause based on the restrictions language and the example selected in the last procedure. To this clause, we call bottom clause. To this step we call saturation; (3) Search for a more general clause than the bottom clause. These searches use the algorithm Branch-and-Bound. To this step, we call reduction; (4) Add the best clause to the theory, remove all redundant examples and return to step 1.

The Aleph has other important characteristics like:

- Instead of selecting one initial example to be generalized, it is possible to choose more than one. If we choose more than one initial example, it is

created a bottom clause to each one of them. After the reduction step, the best of all reductions it is added to the theory;

- Let us construct the more specific clause, defining the place where the bottom clause it is constructed;
- The search clauses can be changed, using other strategies instead of using the branch-and-bound algorithm;
- It is possible to remove redundant examples to give a better perspective of the result clauses.

Aleph uses three files to construct a theory. In order to work properly, these three files should all have the same name. These are: (1) a *file.b* that contains the background knowledge (intentional and extensional), the search, language restrictions and type restrictions and the system parameters. All this content is in the form of Prolog clauses. This file can also contain any directives understood by the Prolog compiler being used; (2) the *file.f* which contains the positive examples (only ground facts) to be learned with Aleph; and (3) a *file.n* that contains the negative examples (only facts without variables). This file may not exist (Aleph can learn only by positive examples). In order to use Aleph, a Prolog compiler is needed. To compile Aleph it can be used one of these two platforms: Yap (Yet Another Prolog) or SWI-Prolog. Both of these compilers are open-source and available for download[1].

The *mode declarations* stored in the *file.b* describe the relations (predicates) between the objects and the type of data. Those declarations allow informing Aleph if the relation can be used in the head (modeh declarations) or in the body (modeb declarations) of the generated rules. The declaration modes also describe the kind of arguments for each predicate, and have the follow format: *mode(RecallNumber, PredicateMode)*. The Recall number (also called recall), define the limit number of alternative instances for one predicate (used for non determinate predicates). A predicate instance it is a substitution of types for each variable or constant. The recall can be any positive number greater or equal to 1 or '*'. If it is known the limit of possible solutions for a particular instance, it is possible to define them

[1] Yap:http://www.dcc.fc.up.pt/~vsc/Yap/andSWI-Prolog:http://www.swi-prolog.org/

by the recall. For instance, if we want to declare the predicate *parent_of(P,D)* the recall should be 2, because the daughter D, has a maximum of two parents P. In the same way, if the predicate was *grandparents(GP,GD)* the recall should be 4, because the granddaughter GD has a maximum of four grandparents GP. The recall '*' is used when there are no limits for the number of solutions to one instance.

The Modes indicates the predicate format, and can be described as: *predicate(ModeType1, ModeType2, ... , ModeTypen)*. The ModeTypes can be organized in one of two ways: simple or structured. The simple modes can be one of: '+', specifying that when a predicate p appears in a clause, the corresponding argument it is an input variable; '-', specifying that the corresponding argument is an output variable; and '#', specifying that the corresponding argument is a constant. A structured ModeTypes is of the form *f(...)* where f is a function symbol, each argument of which is either a simple or structured ModeType. An example of this kind of ModeType is: *:- mode(1,mem(+ number,[+number—+list]))*. So as example, for the learning relation *uncle_of(U,N)* with the background knowledge *parent_of(P,D)* and *sister_of(S1,S2)*, the mode declarations could be:

```
:- modeh(1,uncle_of(+person,+person)).
:- modeb(*,parent_of(-person,+person)).
:- modeb(*,parent_of(+person,-person)).
:- modeb(*,sister_of(+person,-person)).
```

The declaration *modeh* indicates the predicate that will compose the head of the rules. For this case, *modeh* inform us that the head of the rules should be *uncle_of(U,N)* where U and N are from the type person. The symbol '+' that appears before the type indicates us that the argument of the predicate is a input variable. In this way, the head of the rules can be of the type *uncle_of(U,N)*, and not, for instance, *uncle_of(john,ana)*. The symbol '-' indicates us it is an output variable. Instead of '-', if the symbol '#' appeared, indicates us that the argument could be a constant. The *modeb* declaration indicates that the generated rules can have, in the body of the rules, the predicate *parent_of(P,D)*, where P and D are from the type person. The first modeb declaration in the example can be used to add parent of in the body of the rules and add one or more parent(s) to a daughter

(observe that call numbers have the value '*'). Similarly, the second declaration modeb let the predicate parent of be used in the body of the rules to find one or more daughters of a parent. At last, the third *modeb* declaration can be used to find one or more sister of a person. Types have to be specified for every argument of all predicates to be used in constructing a hypotheses [97]. To the Aleph, these types are names and these names means facts. For example, the description of objects for the type person could be: *person(john), person(leihla), person(richard), etc.* Variables of different types are treated distinctly, even if one is a sub-type of the other.

The *determination* statements declare the predicated that can be used to construct hypotheses. This declaration take the format: determination(Target_Pred/Arity_t, Body_Pred/Arity_b). The first argument is the name and arity of the target predicate. It is the predicate that will appear in the head of the induced rule. The second argument it is the name of the predicate that can appear in the body of the rule. A possible determination for a relation called uncle_of(U,N) is: determination(uncle_of/2, parent_of/2). Typically, lots of declarations should be done for a target predicate. In case of non declared determinations, the Aleph doesn't construct any rule. Determinations are only allowed for 1 target predicate on any given run of Aleph: if multiple target determinations occur, the first one is chosen.

Finally, the positive examples of the concept to be learned should be stored in the file with extension .f and the negative examples in the file with extension .n. For instance, to learn the concept *uncle_of(U,N)*, we could have the follow positive examples in the file with extension .f: *uncle_of(Sam,Henry), uncle_of(Martha,Henry), etc.* And the follow negative examples in the file with extension .n: *uncle_of(Lucy,Charles), uncle_of(Lucy,Dominic), etc.*

Bibliography

[1] E. Turban, R. Sharda, and D. Delen. *Decision Support and Business Intelligence Systems*. Prentice Hall, 2010. ISBN 9780136107293.

[2] Vidette Poe, Stephen Brobst, and Patricia Klauer. *Building a Data Warehouse for Decision Support*. Prentice-Hall, Inc., Upper Saddle River, USA, 1997. ISBN 0137696396.

[3] R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, USA, 2002. ISBN 0471200247.

[4] Ralph Kimball and Margy Ross. *The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence*. John Wiley & Sons, Inc., New York, USA, 2010.

[5] J. Trujillo and S. Luján-Mora. A UML Based Approach for Modeling ETL Processes in Data Warehouses. In *ER*, pages 307–320. Springer, 2003.

[6] S. Luján-Mora, J. Trujillo, and I.-Y. Song. A UML profile for multidimensional modeling in data warehouses. *Data Knowl. Eng.*, 59(3):725–769, 2006.

[7] N. Prat, J. Akoka, and I. Comyn-Wattiau. A UML-based data warehouse design method. *Decis. Support Syst.*, 42(3):1449–1473, 2006. ISSN 0167-9236.

[8] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, and S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. *Inf. Syst.*, 30(7):492–525, 2005. ISSN 0306-4379.

[9] A. Simitsis. Mapping conceptual to logical models for ETL processes. In *DOLAP*, pages 67–76. ACM, 2005. ISBN 1-59593-162-7.

[10] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual Design of Data Warehouses from E/R Schema. In *HICSS*, page 334. IEEE Computer Society, 1998. ISBN 0-8186-8251-5.

[11] M. Boehnlein and A. Ulbrich vom Ende. A.: Business Process Oriented Development of Data Warehouse Structures. In *Data Warehousing 2000, Physica Verlag*, pages 3–21, 2000.

[12] P. Westerman. *Data warehousing: using the Wal-Mart model*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2001. ISBN 1-55860-684-X.

[13] B. List, J. Schiefer, and A. M. Tjoa. Process-Oriented Requirement Analysis Supporting the Data Warehouse Design Process - A Use Case Driven Approach. In *DEXA*, pages 593–603. Springer, 2000.

[14] C. Kaldeich and J. Oliveira e Sá. Data Warehouse Methodology: A Process Driven Approach. In *CAiSE*, pages 536–549. Springer, 2004.

[15] Craig D. Weissman and Steve Bobrowski. The design of the force.com multitenant internet application development platform. In *SIGMOD*, SIGMOD '09, pages 889–896. ACM, 2009. ISBN 978-1-60558-551-2.

[16] Tang Liyang, Ni Zhiwei, Wu Zhangjun, and Wang Li. A conceptual framework for business intelligence as a service (saas bi). In *ICICTA*, ICICTA '11, pages 1025–1028. IEEE Computer Society, 2011. ISBN 978-0-7695-4353-6.

[17] Jean Bézivin. Model driven engineering: An emerging technical space. In *GTTSE*, pages 36–64. Springer, 2006.

[18] Ivan Bratko. Applications of machine learning: Towards knowledge synthesis. *New Generation Comput.*, 11(3):343–360, 1993.

[19] Pat Langley and Herbert A. Simon. Applications of machine learning and rule induction. *Commun. ACM*, 38(11):54–64, November 1995. ISSN 0001-0782.

[20] D. Varró and Z. Balogh. Automating model transformation by example using inductive logic programming. In *SAC*, pages 978–984, New York, NY, USA, 2007. ACM. ISBN 1-59593-480-4.

[21] Peter Stone and Manuela M. Veloso. Layered learning. In *ECML*, pages 369–381. Springer, 2000.

[22] João Gama and Pavel Brazdil. Cascade Generalization. *Mach. Learn.*, 41: 315–343, December 2000.

[23] Peter D. Turney. Exploiting context when learning to classify. In *ECML*, pages 402–407, London, UK, 1993. Springer-Verlag.

[24] J. Miller and J. Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), 2003.

[25] Object Management Group (OMG). The Unified Modeling Language (UML) Specification., 2008. URL http://www.omg.org/spec/UML/.

[26] Object Management Group (OMG). The Common Warehouse Metamodel (CWM) Specification., 2008. URL http://www.omg.org/spec/CWM/.

[27] Jose-Norberto Mazón and Juan Trujillo. An mda approach for the development of data warehouses. *Decis. Support Syst.*, 45:41–58, 2008.

[28] Leopoldo Zepeda, Matilde Celma, and Ramon Zatarain. A Mixed Approach for Data Warehouse Conceptual Design with MDA. In *ICCSA*, pages 1204–1217, Perugia, Italy, June 2008. Springer-Verlag.

[29] Object Management Group (OMG). The Query/View/Transformation (QVT) Specification., 2010. URL http://www.omg.org/spec/QVT/.

[30] Dániel Varró. Model Transformation by Example. In *MoDELS*, pages 410–424, Genova, Italy, October 2006. Springer.

[31] Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.

[32] W.H. Inmon. *Building the Data Warehouse,3rd Ed.* John Wiley & Sons, Inc., New York, USA, 2002. ISBN 0471081302.

[33] N. F. Noy. Semantic integration: a survey of ontology-based approaches. *SIGMOD Rec.*, 33(4):65–70, 2004. ISSN 0163-5808.

[34] M. Polo, I. García-Rodríguez, and M. Piattini. An MDA-based approach for database re-engineering. *J. Softw. Maint. Evol.*, 19(6):383–417, 2007. ISSN 1532-060X.

[35] D. Skoutas and A. Simitsis. Designing ETL processes using semantic web technologies. In *DOLAP*, pages 67–74. ACM, 2006. ISBN 1-59593-530-4.

[36] S. Dessloch, M.A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, pages 1307–1316. IEEE Computer Society, 2008.

[37] J. Pardillo, J.-N. Mazón, and J. Trujillo. Model-Driven Metadata for OLAP Cubes from the Conceptual Modelling of Data Warehouses. In *DaWaK*, pages 13–22. Springer-Verlag, 2008. ISBN 978-3-540-85835-5.

[38] J. Pardillo, J.J. Zubcoff, J.-N. Mazón, and J. Trujillo. Towards A Model-Driven Engineering Approach of Data Mining. In *IADIS*, pages 144–147, 2008.

[39] P. Vassiliadis, A. Simitsis, and S. Skiadopoulos. Conceptual modeling for ETL processes. In *DOLAP*, pages 14–21. ACM, 2002. ISBN 1-58113-590-4.

[40] S. Dessloch, M.A. Hernández, R. Wisnesky, A. Radwan, and J. Zhou. Orchid: Integrating Schema Mapping and ETL. In *ICDE*, pages 1307–1316. IEEE Computer Society, 2008.

[41] F. Hakimpour and A. Geppert. Resolution of Semantic Heterogeneity in Database Schema Integration Using Formal Ontologies. *Inf. Technol. and Management*, 6(1):97–122, 2005. ISSN 1385-951X.

[42] P. Chowdhary, T. Palpanas, F. Pinel, S.-K. Chen, and F. Y. Wu. Model-Driven Dashboards for Business Performance Reporting. In *EDOC*, pages 374–386. IEEE Computer Society, 2006. ISBN 0-7695-2558-X.

[43] M. Golfarelli and S. Rizzi. UML-Based Modeling for What-If Analysis. In *DaWaK*, pages 1–12. Springer-Verlag, 2008. ISBN 978-3-540-85835-5.

[44] P. Giorgini, S. Rizzi, and M. Garzetti. Goal-oriented requirement analysis for data warehouse design. In *DOLAP*, pages 47–56. ACM, 2005. ISBN 1-59593-162-7.

[45] N. Tryfona, F. Busborg, and J.G. Borch Christiansen. starER: a conceptual model for data warehouse design. In *DOLAP*, pages 3–8. ACM, 1999. ISBN 1-58113-220-4.

[46] J. Lechtenbörger and G. Vossen. Multidimensional normal forms for data warehouse design. *Inf. Syst.*, 28(5):415–434, 2003. ISSN 0306-4379.

[47] Alkis Simitsis. Modeling and managing ETL processes. In *VLDB PhD Workshop*. CEUR-WS.org, 2003.

[48] A. Simitsis, D. Skoutas, and M. Castellanos. Natural language reporting for ETL processes. In *DOLAP*, pages 65–72. ACM, 2008. ISBN 978-1-60558-250-4.

[49] B. List, R. M. Bruckner, K. Machaczek, and J. Schiefer. A Comparison of Data Warehouse Development Methodologies Case Study of the Process Warehouse. In *DEXA*, pages 203–215. Springer-Verlag, 2002. ISBN 3-540-44126-3.

[50] S. Luján-Mora and J. Trujillo. A Data Warehouse Engineering Process. In *ADVIS*, pages 14–23. Springer, 2004.

[51] C. Brandas. Unified Approach in the DSS Development Process. *Informatica Economica, Vol. 41, No. 1, pp. 98-102*, 2007.

[52] Stuart Kent. Model driven language engineering. *Electr. Notes Theor. Comput. Sci.*, 72(4), 2003.

[53] Vinay Kulkarni, Sreedhar Reddy, and Asha Rajbhoj. Scaling up model driven engineering - experience and lessons learnt. In *MoDELS*, pages 331–345. Springer, 2010.

[54] Jose-Norberto Mazon, Juan Trujillo, Manuel Serrano, and Mario Piattini. Applying MDA to the development of data warehouses. In *DOLAP*, DOLAP '05, pages 57–66, New York, NY, USA, 2005. ACM.

[55] Moshé M. Zloof. Query-by-example: the invocation and definition of tables and forms. In *VLDB*, pages 1–24, New York, NY, USA, 1975. ACM.

[56] Alexander Repenning and Corrina Perrone. Programming by example: programming by analogous examples. *Commun. ACM*, 43(3):90–97, 2000. ISSN 0001-0782.

[57] Allen Cypher, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky, editors. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA, 1993. ISBN 0-262-03213-9.

[58] Martin Erwig. Toward the automatic derivation of xml transformations. In *XSDM*, pages 342–354. Springer, 2003.

[59] Kouichi Ono, Teruo Koyanagi, Mari Abe, and Masahiro Hori. Xslt stylesheet generation by example with wysiwyg editing. In *SAINT*, pages 150–161, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1447-2.

[60] Ling Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD*, pages 485–496, New York, NY, USA, 2001. ACM. ISBN 1-58113-332-4.

[61] Dániel Varró. Model transformation by example. In *MoDELS*, pages 410–424. Springer, 2006.

[62] J. Abd-Ali and K. El Guemhioui. Automating Metamodel Mapping Using Machine Learning. In *3M4MDA*, pages 103–109, 2006.

[63] Manuel Wimmer, Michael Strommer, Horst Kargl, and Gerhard Kramler. Towards Model Transformation Generation By-Example. In *HICSS*, pages 285b–, Washington, DC, USA, 2007. IEEE Computer Society.

[64] Frédéric Jouault and Ivan Kurtev. Transforming models with atl. In *MoDELS Satellite Events*, pages 128–138. Springer, 2005.

[65] Michael Strommer, Marion Murzek, and Manuel Wimmer. Applying model transformation by-example on business process modeling languages. In *Proceedings of the 2007 conference on Advances in conceptual modeling: foundations and applications*, pages 116–125, Auckland, New Zealand, 2007. Springer-Verlag.

[66] M. Kessentini, H. Sahraoui, and M. Boukadoum. Model Transformation as an Optimization Problem. In *MoDELS*, pages 159–173, Toulouse, France, 2008. Springer-Verlag.

[67] M. Kessentini, H. Sahraoui, and M. Boukadoum. Méta-modélisation de la transformation de modéles par l'exemple : approche par méta-heuristiques. In *LMO*, 2009.

[68] Xavier Dolques, Marianne Huchard, and Clémentine Nebut. From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis. In *ICCS*, pages 093–106, Moscow, Russia, 2009.

[69] Yu Sun, Jules White, and Jeff Gray. Model transformation by demonstration. In *MoDELS*, pages 712–726. Springer, 2009.

[70] Stephan Roser and Bernhard Bauer. An approach to automatically generated model transformations using ontology engineering space. In *SWESE*, 2006.

[71] Elisabeth Kapsammer, Horst Kargl, Gerhard Kramler, Thomas Reiter, Werner Retschitzegger, and Manuel Wimmer. Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. In *MoDELS/UML*, pages 528–542. Springer, 2006.

[72] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE*, pages 117–128. IEEE Computer Society, 2002.

[73] Jean-Rémy Falleri, Marianne Huchard, Mathieu Lafourcade, and Clémentine Nebut. Metamodel matching for automatic model transformation generation. In *MoDELS*, pages 326–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 978-3-540-87874-2.

[74] Sinh Hoa Nguyen, Jan G. Bazan, Andrzej Skowron, and Hung Son Nguyen. Layered Learning for Concept Synthesis. *T. Rough Sets*, 3100:187–208, 2004.

[75] Andrzej Bieszczad and Kasia Bieszczad. Contextual learning in the neurosolver. In *ICANN*, pages 474–484. Springer, 2006.

[76] Stephen Muggleton and Keble Road. Predicate Invention and Utilisation. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:6–1, 1994.

[77] Irene Stahl. On the Utility of Predicate Invention in Inductive Logic Programming. In *ECML*, pages 272–286. Springer, 1994.

[78] Joao Gama. Combining Classifiers by Constructive Induction. In *ECML*, pages 178–189. Springer, 1998.

[79] Zhipeng Xie. Several Speed-Up Variants of Cascade Generalization. In *FSKD*, pages 536–540, Xi'an, China, September 2006. Springer.

[80] Ethem Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2nd edition, 2010. ISBN 026201243X, 9780262012430.

[81] Du Zhang and Jeffrey J. P. Tsai. *Advances in Machine Learning Applications in Software Engineering*. IGI Publishing, Hershey, PA, USA, 2007.

[82] Dragan Djuric, Vladan Devedzic, and Dragan Gasevic. Adopting Software Engineering Trends in AI. *IEEE Intelligent Systems*, 22:59–66, 2007.

[83] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[84] J. R. Quinlan. Induction of decision trees. *Mach. Learn.*, 1:81–106, March 1986. ISSN 0885-6125.

[85] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 1-55860-238-0.

[86] S. G. Fountoukis, M. P. Bekakos, and J. P. Kontos. Rule extraction from decision trees with complex nominal data. *Neural, Parallel Sci. Comput.*, 9: 119–128, June 2001. ISSN 1061-5369.

[87] John C. Platt. *Fast training of support vector machines using sequential minimal optimization*, pages 185–208. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-19416-3.

[88] Ingo Steinwart and Andreas Christmann. *Support Vector Machines*. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 0387772413.

[89] Mohamad H. Hassoun. *Fundamentals of Artificial Neural Networks*. MIT Press, Cambridge, MA, 1995.

[90] Laurene Fausett, editor. *Fundamentals of neural networks: architectures, algorithms, and applications*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1994. ISBN 0-13-334186-0.

[91] Nir Friedman and Moises Goldszmidt. *Learning Bayesian networks with local structure*, pages 421–459. MIT Press, Cambridge, MA, USA, 1999. ISBN 0-262-60032-3.

[92] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675.

[93] Thomas Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996. ISBN 0-19-509971-0.

[94] Shan-Hwei Nienhuys-Cheng and Ronald de Wolf. *Foundations of Inductive Logic Programming*, chapter 8, pages 127–159. Springer, 1997.

[95] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.

[96] Luc De Raedt. *Logical and Relational Learning*. Cognitive Technologies. Springer, 2008.

[97] A. Srinivasan. The aleph manual. Available in http://www.cs.ox.ac.uk/activities/machlearn/Aleph/, 1999.

[98] J. Ross Quinlan and R. Mike Cameron-Jones. FOIL: A midterm report. In *ECML*, volume 667, pages 3–20. Springer-Verlag, 1993.

[99] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[100] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[101] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297, 1998.

[102] S. Muggleton and C. Feng. Golem. Available at http://www-ai.ijs.si/ ilp-net2/systems/golem.html, 1992.

[103] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *SIGMOD*, pages 207–216, Washington, D.C., 26–28 1993.

[104] L. De Raedt and M. Bruynooghe. A theory of clausal discovery. In *IJCAI*, pages 1058–1063, 1993.

[105] L. De Raedt and L. Dehaspe. Clausal discovery. *Machine Learning*, 26: 99–146, 1997.

[106] H. Blockeel and L. De Raedt. Isidd: An interactive system for inductive database design. *Applied Artificial Intelligence*, 12(5):385–420, 1998.

[107] Simon Colton and Stephen Muggleton. Mathematical applications of inductive logic programming. *Mach. Learn.*, 64(1-3):25–64, 2006. ISSN 0885-6125.

[108] S. Colton and S. Muggleton. ILP for mathematical discovery. In *T. Horvath and A. Yamamoto, editors, ILP03, volume 2835 of LNAI*, pages 93–111. Springer, 2003.

[109] Simon Colton. *Automated Theory Formation in Pure Mathematics*. Springer-Verlag, 2002.

[110] Luc Dehaspe. *Frequent Pattern Discovery in First-Order Logic*. PhD thesis, Katholieke Universiteit Leuven, 1998.

[111] L. Dehaspe and L. De Raedt. Mining association rules in multiple relations. In *ILP*, volume 1297, pages 125–132. Springer-Verlag, 1997.

[112] Luc Dehaspe and Hannu Toivonen. Discovery of frequent DATALOG patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.

[113] Wim Van Laer. *From Propositional to First Order Logic in Machine Learning and Data Mining – Induction of first order rules with ICL*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, 2002.

[114] Lubos Popelínský. *Inductive Logic Programming. In On Practical Inductive Logic Programming, Doctorate thesis, FEE-CTUP, Prague*. PhD thesis, 2000.

[115] Steven M. Gustafson and William H. Hsu. Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem. In *EuroGP*, pages 291–301, London, UK, 2001. Springer-Verlag.

[116] David Jackson and Adrian P. Gibbons. Layered learning in boolean GP problems. In *EuroGP*, pages 148–159. Springer-Verlag, 2007.

[117] Stephen Muggleton. Optimal Layered Learning: A PAC Approach to Incremental Sampling. In *ALT*, pages 37–44, London, UK, 1993. Springer-Verlag.

[118] Stephen Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295–318, 1991.

[119] Irene Stahl. The Appropriateness of Predicate Invention as Bias Shift Operation in ILP. *Mach. Learn.*, 20:95–117, July 1995.

[120] Riverson Rios and Stan Matwin. Predicate Invention from a Few Examples. In *AI*, pages 455–466, London, UK, 1998. Springer-Verlag.

[121] David H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.

[122] Kai Ming Ting and Ian H. Witten. Stacked generalization: when does it work? In *IJCAI*, pages 866–871. Morgan Kaufmann, 1997.

[123] Kai Ming Ting and Ian H. Witten. Issues in stacked generalization. *J. Artif. Intell. Res. (JAIR)*, 10:271–289, 1999.

[124] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.

[125] Eclipse. The Model To Model (M2M) Transformation Framework., 2010. URL http://www.eclipse.org/m2m/.

[126] Software & Information Industry Association. Software-as-a-Service: A Comprehensive Look at the Total Cost of Ownership of Software Applications. http://www.winnou.com/saas.pdf, September 2006.

[127] Moez Essaidi. ODBIS: towards a platform for on-demand business intelligence services. In *EDBT/ICDT Workshops*. ACM, 2010.

[128] Moez Essaidi and Aomar Osmani. *Business Intelligence-as-a-Service: Studying the Functional and the Technical Architectures*, chapter 9, pages 199–221. IGI Global, 2012.

[129] Michael Lawley and Jim Steel. Practical declarative model transformation with tefkat. In *MoDELS Satellite Events*, pages 139–150. Springer, 2005.

[130] Moez Essaidi, Aomar Osmani, and Céline Rouveirol. Transformation Learning in the Context of Model-Driven Data Warehouse: An Experimental Design Based on Inductive Logic Programming. In *ICTAI*, pages 693–700. IEEE, 2011.

[131] Moez Essaidi, Aomar Osmani, and Céline Rouveirol. Learning Dependent-Concepts in ILP: Application to Model-Driven Data Warehouse. In *Latest Advances in Inductive Logic Programming*. Imperial College Press, 2012.

[132] Moez Essaidi and Aomar Osmani. Towards Model-driven Data Warehouse Automation using Machine Learning. In *IJCCI (ICEC)*, pages 380–383, Valencia, Spain, 2010. SciTePress.

[133] J.-N. Mazón, J. Pardillo, E. Soler, O. Glorio, and J. Trujillo. Applying the i* Framework to the Development of Data Warehouses. In *iStar*, pages 79–82. CEUR-WS.org, 2008.

[134] Microsoft MSDN Library. SQL Server Integration Services., 2009. URL http://msdn.microsoft.com/en-us/library/ms141026.aspx.

[135] Microsoft MSDN Library. Creating a Simple ETL Package., 2009. URL http://msdn.microsoft.com/en-us/library/ms169917.aspx.

[136] OMG. Catalog of Modeling and Metadata Specifications., 2008. URL http://www.omg.org/technology/documents/modeling_spec_catalog.htm.

[137] Ehud Y. Shapiro. *Algorithmic Program Debugging*. MIT Press, Cambridge, MA, USA, 1983. ISBN 0262192187.

[138] Rakesh Agrawal, Heikki Mannila, Ramakrishnan Srikant, Hannu Toivonen, and A. Inkeri Verkamo. Fast discovery of association rules. In *Advances in Knowledge Discovery and Data Mining*, pages 307–328. AAAI/MIT Press, 1996. ISBN 0-262-56097-6.

[139] Zoltán Balogh and Dániel Varró. Model transformation by example using inductive logic programming. *Software and Systems Modeling*, 8:347–364, 2009.

[140] Xavier Dolques, Marianne Huchard, and Clémentine Nebut. From transformation traces to transformation rules: Assisting model driven engineering

approach with formal concept analysis. In *ICCS*, pages 093–106, Moscow, Russia, 2009.

[141] Xavier Dolques, Marianne Huchard, Clementine Nebut, and Philippe Reitz. Learning Transformation Rules from Transformation Examples: An Approach Based on Relational Concept Analysis. *EDOC Workshops, IEEE International*, 0:27–32, 2010.

[142] Michael Strommer and Manuel Wimmer. A framework for model transformation by-example: Concepts and tool support. In *TOOLS*, pages 372–391, Zurich, Switzerland, June 2008. Springer.

[143] Stephen H. Muggleton. Inverse entailment and progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13:245–286, 1995.

[144] Domenico Corapi, Oliver Ray, Alessandra Russo, Arosha K. Bandara, and Emil C. Lupu. Learning rules from user behaviour. In *AIAI*, pages 459–468. Springer, 2009.

[145] Frédéric Jouault and Jean Bézivin. KM3: A DSL for Metamodel Specification. In *FMOODS*, pages 171–185. Springer, 2006.

[146] K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45:621–645, July 2006. ISSN 0018-8670.

[147] Anna Gerber, Michael Lawley, Kerry Raymond, Jim Steel, and Andrew Wood. Transformation: The missing link of mda. In *ICGT*, pages 90–105. Springer, 2002.

[148] Adrian Rutle, Uwe Wolter, and Yngve Lamo. A diagrammatic approach to model transformations. In *EATIS*, 2008.

[149] Zinovy Diskin and Uwe Wolter. A diagrammatic logic for object-oriented visual modeling. *Electr. Notes Theor. Comput. Sci.*, 203(6):19–41, 2008.

[150] Adrian Rutle, Alessandro Rossini, Yngve Lamo, and Uwe Wolter. A diagrammatic formalisation of mof-based modelling languages. In *TOOLS*, pages 37–56. Springer, 2009.

[151] Perdita Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and System Modeling*, 9(1):7–20, 2010.

[152] Moez Essaidi and Aomar Osmani. Model driven data warehouse using MDA and 2TUP. *Journal of Computational Methods in Sciences and Engineering*, 10:119–134, 2010.

[153] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.

[154] Marouane Kessentini, Manuel Wimmer, Houari Sahraoui, and Mounir Boukadoum. Generating transformation rules from examples for behavioral models. In *BM-FA*, pages 2:1–2:7. ACM, 2010.

[155] Floriana Esposito, Giovanni Semeraro, Nicola Fanizzi, and Stefano Ferilli. Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning*, 38(1-2):133–156, 2000.

[156] Wei Zhang, Hong Mei, Haiyan Zhao, and Jie Yang. Transformation from cim to pim: A feature-oriented component-based approach. In *MoDELS*, pages 248–263. Springer, 2005.

[157] Nitin Bhatia and Vandana. Survey of nearest neighbor techniques. *CoRR*, abs/1007.0085, 2010.

[158] S. Kramer. Relational learning vs. propositionalization: Investigations in inductive logic programming and propositional machine learning. *AI Communications*, 13(4):275–276, 2000.

[159] Stefan Kramer, Nada Lavrač, and Peter Flach. *Propositionalization approaches to relational data mining*, pages 262–286. Springer-Verlag New York, 2000.

[160] Stephen Muggleton. Inductive logic programming. *New Generation Computing*, 8(4):295–318, 1991.

[161] Microsoft. Microsoft AdventureWorks 2008R2, 2011. URL http://msftdbprodsamples.codeplex.com/.

[162] A. Srinivasan. A learning engine for proposing hypotheses (Aleph). http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph, 2006.

[163] Robert Wrembel and Christian Koncilia. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*. IGI Global, 2007.

[164] Tom Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories, 2004.

[165] Kashif Mehmood, Samira Si-Said Cherfi, and Isabelle Comyn-Wattiau. Data quality through conceptual model quality - reconciling researchers and practitioners through a customizable quality model. In *ICIQ*, pages 61–74. HPI/MIT, 2009.

[166] Gwenaël Kersulec, Samira Si-Said Cherfi, Isabelle Comyn-Wattiau, and Jacky Akoka. Un environnement pour lévaluation et lamélioration de la qualité des modèles de systèmes dinformation. In *INFORSID*, pages 329–344, 2009.

[167] Bob Becker. Another look at degenerate dimensions. http://www.kimballgroup.com/category/design-tips/, 2003.

[168] Robert Kosara, Fabian Bendix, and Helwig Hauser. Parallel sets: Interactive exploration and visual analysis of categorical data. *Transactions on Visualization and Computer Graphics*, 12:558–568, 2006.

[169] M. Essaidi and A. Osmani. Data Warehouse Development Using MDA and 2TUP. In *SEDE*, pages 138–143. ISCA, 2009.

[170] SAP BusinessObjects. SAP BusinessObjects Hosted On-Demand Platform., 2010. URL http://www.ondemand.com.

[171] PivotLink. PivotLink Platform., 2010. URL http://www.pivotlink.com.

[172] MicroStrategy. MicroStrategy Platform., 2010. URL http://www.microstrategy.com.

[173] MicroStrategy. An Architecture for Software-as-a-Service (SaaS) Business Intelligence. http://www.b-eye-network.com/files/SaaS_WP.pdf, 2009.

[174] LogiXML. LogiXML Solution., 2010. URL http://www.logixml.com.

[175] Talend. Talend Platform., 2010. URL http://www.talend.com.

[176] Pentaho. Pentaho Platform., 2010. URL http://www.pentaho.com.

[177] OpenI. OpenI Reporting., 2010. URL http://openi.org.

[178] Apache. Apache ServiceMix Platform., 2010. URL http://servicemix.apache.org/.

[179] JBoss. JBoss ESB Platform., 2010. URL http://www.jboss.org/jbossesb.

[180] Eclipse. Eclipse-Birt Platform., 2010. URL http://www.eclipse.org/birt/.

[181] JBoss. JBoss Drools., 2010. URL http://www.jboss.org/drools.

[182] C. Thomsen and T.B. Pedersen. A Survey of Open Source Tools for Business Intelligence. In *DaWaK*, pages 74–84, 2005.

[183] Intelligence Power Project Hosted by sourceforge.net. Intelligence Power ODBIS., 2009. URL http://odbis-project.sourceforge.net.

[184] dotProject. Project Management Tool: dotProject, 2010. URL http://www.dotproject.net/.

[185] yWorks. Web-Based Diagramming Tool: yWorks, 2010. URL http://www.yworks.com/.

[186] Gliffy. Online Diagram Software and Flowchart Software - Gliffy, 2010. URL http://www.gliffy.com/.

[187] OpenLaszlo. OpenLaszlo: The Premier Platform for Rich Internet Applications, 2010. URL http://www.openlaszlo.org/.

[188] MxGraph. MxGraph - JavaScript Diagramming and Graph Visualization, 2010. URL http://www.mxgraph.com/.

[189] JS-Graph-IT. JS-Graph-IT: A JavaScript Library for Graphs Representation, 2010. URL http://js-graph-it.sourceforge.net.

[190] Jalava. Jalava : Web-based Diagram Editor, 2010. URL http://jalava.buildyourownapps.com/.

[191] VMware. VMware vFabric Cloud Application Platform., 2011. URL https://www.vmware.com/products/.

[192] GemStone Systems. GemFire : Enterprise Data Solutions., 2011. URL http://www.gemstone.com/.

[193] Nicolas Prat, Jacky Akoka, and Isabelle Comyn-Wattiau. An MDA Approach to Knowledge Engineering. *Expert Syst. Appl.*, 39(12):10420–10437, 2012.

[194] W.H. Inmon, D. Strauss, and G. Neushloss. *DW 2.0: The Architecture for the Next Generation of Data Warehousing*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008. ISBN 0123743192, 9780123743190.

[195] Victoria Nebot and Rafael Berlanga Llavori. Building data warehouses with semantic data. In *EDBT/ICDT Workshops*. ACM, 2010.

[196] Juan Manuel Pérez, Rafael Berlanga, and María José Aramburu. A relevance model for a data warehouse contextualized with documents. *Inf. Process. Manage.*, 45(3):356–367, 2009. ISSN 0306-4573.

[197] Liya Wu, Gilad Barash, and Claudio Bartolini. A service-oriented architecture for business intelligence. *Service-Oriented Computing and Applications*, 0: 279–285, 2007.

[198] Philip Russom. Next Generation Data Warehouse Platforms. http://www.oracle.com/database/docs/tdwi-nextgen-platforms.pdf, 2009.

[199] Torben Bach Pedersen. Warehousing the world: a few remaining challenges. In *DOLAP*, pages 101–102, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-827-5.

[200] Karen Henricksen and Jadwiga Indulska. Developing context-aware pervasive computing applications: Models and approach. *Pervasive and Mobile Computing*, 2(1):37–64, 2006.

[201] Tai-Jong Kim and Min-Cheol Kim. Context awareness using semantic web technology in the ubiquitous learning service. In *Proceeding sof the international conference on Computational Science and Its Applications, Part I*, ICCSA 2008, pages 501–515, Berlin, Heidelberg, 2008. Springer-Verlag.

[202] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. In *Proceedings of the 1994 First Workshop on Mobile Computing Systems and Applications*, WMCSA 1994, pages 85–90, Washington, DC, USA, 1994. IEEE Computer Society.

[203] Eduardo Fernández-Medina, Jan Jürjens, Juan Trujillo, and Sushil Jajodia. Model-driven development for secure information systems. *Information & Software Technology*, 51(5):809–814, 2009.

[204] Rodolfo Villarroel, Eduardo Fernández-Medina, and Mario Piattini. Secure information systems development - a survey and comparison. *Computers & Security*, 24(4):308–321, 2005.

[205] David A. Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.

[206] Michael Hafner, Ruth Breu, Berthold Agreiter, and Andrea Nowak. Sectet: an extensible framework for the realization of secure inter-organizational workflows. *Internet Research*, 16(5):491–506, 2006.

[207] Fredrik Seehusen and Ketil Stølen. A transformational approach to facilitate monitoring of high-level policies. In *POLICY*, pages 70–73. IEEE Computer Society, 2008.

[208] Fredrik Seehusen and Ketil Stølen. Maintaining information flow security under refinement and transformation. In *FAST*, pages 143–157. Springer-Verlag, 2007.

[209] N. Moebius, K. Stenzel, and W. Reif. Generating formal specifications for security-critical applications - a model-driven approach. In *IWSESS*, pages 68–74, Washington, DC, USA, 2009. IEEE Computer Society.

[210] Slim Kallel, Anis Charfi, Mira Mezini, Mohamed Jmaiel, and Andreas Sewe. A holistic approach for access control policies: from formal specification to aspect-based enforcement. *Int. J. Inf. Comput. Secur.*, 3(3/4):337–354, 2009.

[211] Object Management Group (OMG). The OMG Security Specifications Catalog., 2008. URL http://www.omg.org/technology/documents/formal/omg_security.htm.

[212] Object Management Group (OMG). The Ontology Definition Metamodel (ODM) Specification., 2009. URL http://www.omg.org/spec/ODM/.

[213] Object Management Group (OMG). OMG Mission Statement., 2010. URL http://www.omg.org/gettingstarted/gettingstartedindex.htm.

[214] Object Management Group (OMG). The Object Management Architecture., 2010. URL http://www.omg.org/oma/.

[215] Object Management Group. The MetaObject Facility Specification. http://www.omg.org/mof/, 2006.

[216] P.,, Roques and F.,, Vallée. *UML 2 en action : De l'analyse des besoins à la conception J2EE*. Eyrolles, third edition, 2004. ISBN 2-212-11462-1.

[217] A. Belangour, J. Bézivin, and M. Fredj. Towards a new software development process for MDA. In *Proc. of the European ECMDA*, pages 1–16, 2006.