# Model-Driven Data Warehouse Automation: A Dependent-Concept Learning Approach

Moez Essaidi, Aomar Osmani, Céline Rouveirol

LIPN - UMR CNRS 7030, Université Paris-Nord,
99 Avenue Jean-Baptiste Clément, 93430 Villetaneuse, France.
{essaidi, osmani, rouveirol}@lipn.univ-paris13.fr

**Abstract.** We propose, in this chapter, a new methodology that combines the model-driven paradigm and machine learning techniques, in order to automate the development of data warehousing components. The main goal is to automatically derive the transformation rules to be applied in the *model-driven data warehouse* process. The proposed solution allows the simplicity of *decision support systems* design and the reduction of time and costs of development. We use the *inductive logic programming* framework to express the model transformation context. We find that in *model-driven data warehouse* application, dependencies exist between transformations. So, we investigate a new machine learning methodology, *learning dependent-concepts*, that is suitable to solve this kind of problem. The experimental evaluation shows that the dependent-concept learning approach gives significantly better results.

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

## 1 Introduction

*Decision Support Systems* and *Business Intelligence Systems* [88, 68] 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 [40, 41] 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. Several data warehouse design frameworks and engineering processes have been proposed during the last few years. However, the *framework-oriented* approaches [45, 69, 78] fail to provide an integrated and a standard framework that is designed for all layers of the data warehousing architecture. The *process-oriented* approaches [94, 44, 32] 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.

The *Model-Driven Data Warehouse* gathers approaches that align the development of the data warehouse with a general model-driven engineering paradigm [4]. 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., *Query-View-Transformation*). 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 [90]) 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 framework of *Model-Driven Data Warehousing*, several steps are needed to automatically learn the transformation rules. The first step (the modelling step), which has been addressed in the previous papers [14, 15], consists in isolating stages 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. And also to effectively deploy the application, with respect to standards and data warehousing requirements in organisations. Then, We propose to express the model transformation problem as an *Inductive Logic Programming* one [56] and to use existing project traces to find 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.

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* [83, 59], *context learning* [89, 5], *predicate invention* [57, 80] and *cascade learning* [22, 99], 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 2 presents the terminology used and outlines the concerned research fields. In Section 3, the learning aspects of the solution are detailed. The section 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. Section 4 gives experimental results and discussion. Our main perspectives and the future research challenges are presented in Section 5. Section 6 summarizes our contributions and gives our final conclusions and remarks.

## 2 Background

In the problem that we are going to deal with, several concepts and research fields are considered. This section 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 provides also a review of works in various areas (i.e., Model-Driven Data Warehouse approaches, Model Transformation By-Example framework and Concept Learning Strategies) related to the provided methodology.

### 2.1 Model-Driven Data Warehouse

The *Model-Driven Engineering* represents a promising approach to support software development practices [35, 4, 42]. The *Model-Driven Architecture (MDA)* standard [50] represents the *Object Management Group* implementation to support the model-driven approach. 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 (UML)*, the *Meta-Object Facility (MOF)*, specific platforms models (i.e., CORBA, JEE), and the *Common Warehouse Metamodel (CWM)* to design data warehouse components. 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 (figure 1 is an example).

The *Model-Driven Data Warehouse* represents approaches that align the development of the data warehouse with a general *Model-Driven Engineering* paradigm. Related work [46, 101] 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 [101] describes derivation of *OnLine Analytical Processing (OLAP)* schemas from *Entity-Relationship (ER)* schemas. The source and target models are respectivly 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.

The designed transformation rule (by figure 1) shows a candidate Entity that gets transformed to a corresponding Cube. The generated Cube having 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 multidimentional 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.

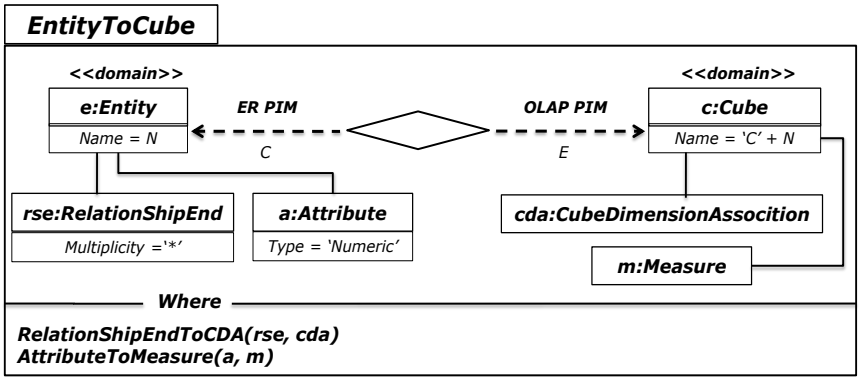


**Fig. 1.** Graphical Notation of EntityToCube Relation.

Through this simple example we shows that the designed transformations require an expert who knows the business domain, the principles of model-driven approach, and the transformation languages [63, 29, 11]. Thus, the actuel *Model-Driven Data Warehouse* automation processes are partial because the design transformations remain manual (for example, no proposal of a framework to learn these transormations). 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.

### 2.2 Model Transformation By-Example

The *Model Transformation By-Example* is related to several others *by-example* based approaches: *query-by-example*, *programming-by-example*, and XSLT generation by-example. The *query-by-example* approach [103] aims at proposing a language for querying relational data constructed from sample tables filled with example rows and constraints. *Programming-by-example* [71, 6], 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 [12, 64, 100], which generate XSLT code to carry out transformations between XML documents.

Varró et al. in [92], present an automated *Model Transformation By-Example* approach using the *Inductive Logic Programing*, an improvement of the initial proposal introduced in [91]. 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 classes diagram (based on the *Unified Modelling Language*) is mapped into a target relational database diagram.

In [1], 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 [95], present a conceptual framework for *Model Transformation By-Example* to derive ATL (*Atlas Transformation Language* [31]) 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 [84], 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 this paper.

In [37], authors present a by-example approach, named *model transformation as optimization by examples (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 [38] authors use the *Simulated Annealing (SA)* to improve the performances of the approach.

Dolques et al. in [10], 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.

In [85], authors 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 [73], metamodels are mapped to a pivot ontology, then an ontology-based reasoning is used to generate a *Relational-QVT* transformation. In [34], 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* [48] algorithm allows similarity values propagation in a labelled graph whose vertices are potential mappings, authors in [19] adapt it for metamodel alignment.

### 2.3 Related Concept-Learning Strategies

The goal of machine learning is to program computers to use example data or past experience to solve a given problem [2]. 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. Machine learning algorithms and techniques have long been used for various purposes in software engineering (testing, validation, security, etc.). The works in [102, 9] 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 the transformation rule from previous projects experiences using a machine learning approach.

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 [52]: *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 $E$.*

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: *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.*

The *Inductive Logic Programming* [60] is 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 [43]. An ILP learning task is defined by four aspects: *model theory* defines the semantic constraints on hypotheses, i.e., what to search for; *proof theory* describes the strategy to perform the search; *declarative bias* explicitly defines the hypothesis space, i.e., where to search; and *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 follow two important settings: *descriptive* and *predictive settings*. In our approach, we are concerned by the predictive setting using the well known Aleph framework [79]. The predictive setting is formally defined as below:

**Definition 1 (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 $B$.*

We will investigate a new machine learning methodology stemming from the application needs: Learning Dependent-Concepts. Following work about 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).

Stone et al. introduce in [83] the *Layered Learning* machine learning paradigm. In [59] authors study the problem of constructing the approximation of higher level concepts by composing the approximation of lower level concepts. Authors in [25, 28] present an alternative to standard genetic programming that applies layered learning techniques to decompose a problem. The layered learning approach presented by Muggleton in [55] aims at the construction of a large theory in small pieces. Compared to layered learning, the *Dependent-Concept Learning (DCL)* approach aims to find all concepts theory using the theories of concepts

on which they depend. 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 [83] is a hierarchy, whereas our dependency structure is a *directed acyclic graph*. A breadth-first search algorithm is used to explore the dependency graph.

Within the field of *Inductive Logic Programming* the term *Predicate Invention* is introduced [54] and it involves the decomposition of predicates being learned into useful sub-concepts. Muggleton in [57] defines *Predicate Invention* as the augmentation of a given theoretical vocabulary to allow finite axiomatisation of the observational predicates. Stahl in [80, 81], studies the utility of *Predicate Invention* task in ILP and its capabilities as a bias shift operation. Rios et al. investigate in [72] on specification language extension when no examples are explicitly given of the invented predicate. The proposed *Dependent-Concept Learning* and *Predicate Invention* approaches share the fact that 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.

In [23], authors introduce the *Cascade Generalization* method. This approach is compared to other approaches that generate and combine different classifiers like the *Stacked Generalization* approach [96, 86, 87]. In [99], Xie proposes several speed-up variants of the original cascade generalization and show that the proposed variants are much faster than the original one. As the *Cascade Generalization*, the *Dependent-Concept Learning* approach extends the background knowledge at each level by the information on concepts of the sub-level (according to the *dependency-graph*). But, within the DCL approach, we use the same classifiers for all iterations. In our experiments, we report the results of the extension of the background knowledge by instances (as first setting) and the learned theory/rules (as second setting). The first setting is named DCLI and the second setting is named DCLR.

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 [90]. The machine learning approaches that exploit context to synthesize concepts are proposed in [89, 5]. In [89] author provides a precise formal definition of context and list four general strategies for exploiting contextual information. Authors in [5] introduce an enhanced architecture that enables contextual learning in the *Neurosolver* (a problem solving system). Nevertheless, the notion of context is different in the *Dependent-Concept Learning* approach. 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 that improves the performance of the learner.

## 3 Learning-Approach

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 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 section, 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.

### 3.1 Key Concepts Formalisation

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 2.** *(***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., it's 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). $\mu$ is neither injective (several model elements may be associated to the same meta-element) nor subjective (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 (InvoiceFact)* element in a DSPIM (MDPIM) is an instance of *Class (Cube)* meta-class in the UML CORE (CWM OLAP) metamodel.

**Definition 3.** *(***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 [30].*

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.

[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 [30].

In [7], authors 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 [7]. For example, in [24], 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 [75] authors discuss a formalization of modeling and model transformation using a generic formalism, the *Diagrammatic Predicate Logic (DPL)*. The *DPL* [8, 74] 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 4.** *(***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* $MM$ *[82].*

### 3.2 Relational Learning Setting

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 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 [56, 43] have been selected to deal with this learning problem. As ILP suffers from a scaling-up problem, the proposed architecture [15, 16] 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.

We consider the machine learning problem as defined in [51]. 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 ($\forall e^- \in E^-, h, B \not\geq e^-$). 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.

We used in our experiments the well known Aleph 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.

### 3.3 Problem Statement in a Relational Learning Setting

In the ILP framework (regarding the background knowledge and examples), a model $M_i$ is characterized by its description $MD_i$, i.e., a set of predicates that correspond to the contained 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)*. The *one-to-many* association that relates *Customer* to *Invoice* is mainly defined by *association(customer-invoice, customer, invoice)* (others predicates, given in the

additional report, are used to define multiplicities of the association). Then, the logic description of models from project's traces constitutes the generated background knowledge program in ILP.

**Definition 5.** *(***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 [39]. 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-modeling-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 concept *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 [13], 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 6.** *(***Dependency Graph** *after [13]) 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* $p$*.*

### 3.4 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 $c_j$ (Definition 5 – 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 – this framework is referred to as DCLI) and (ii) $B_j$ is extended with child concept intensional definitions: all children concepts are learned as sets of rules and are added to $B_j$ – 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 $B_j$, extensionnally or intensionnally. In the next Section (i.e., Evaluation), we provide results concerning the impact of child concepts' representation (extensional vs. intensional) on the the quality of the $c_j$.

Finnaly, 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 $\forall c_i \in C_d$: set of transformation examples (i.e., examples) $E = \{E_1, E_2, \ldots, E_n\}$ is given; and defined as (where $|TM|$ is the number of training models): $E_i = \{R_i^j(M^j, N^j) \mid R^j(M^j, N^j) \subseteq \mathbb{P}(M^j) \times \mathbb{P}(N^j),\ j \leq |TM|\}$ and a background knowledge $B$ which provide 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:*** $\forall c_i \in C_d$, based on $E_d$ and following a *BFS* strategy[3], learn a transfor-

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

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

mation 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 DCL approach aims to find all concepts theory using the theories of concepts on which they depend. 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 [83] is a hierarchy, whereas our dependency structure is a *directed acyclic graph*. A breadth-first search algorithm is used to explore the dependency-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 by the information on concepts of the sub-level (according to the dependency graph). But, within the proposed DCL, we use the same classifiers for all iterations. In our experiments, we report the results of the extension of the background knowledge by instances (first setting named DCLI) and the learned theory (second setting named DCLR).

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 [90]. However, the notion of context is different in DCL. 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.

## 4 Evaluation

For the evaluation, we propose to compare the following approaches: (i) The *Independent-Concept Learning (ICL)* approach, which proposes to learn the set of considered concepts independently. And (ii) The *Dependent-Concept Learning (DCL)* approach, which consider a dependency graph to learn the concepts. Within this approach, we benchmark two settings: (i) the background knowledge $B$ of dependent-concepts (i.e., parent-concepts) is updated with their child instances (denoted as DCLI) and (ii) with their chlid intensional definitions (denoted as DCLR).

### 4.1 Materials and Methods

We use a set of real-world data models for experimentations. The models represent projects' traces such as the example presented in figure 2. In each project trace, we find the *source-model(s)*, the *target-model(s)* and the *transformations*. The *source-models* description includes mainly 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.

We define the language bias using the metamodel level ($M2$ level) of the meta-modelling 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). CWM OLAP defines the predicates used for representing target-models examples (denoted as MDPIMs in figure). 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 2). 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.

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. 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. In the example, *classtocube(invoice)* is a positive example and *classtocube(customer)*, *classtocube(seller)*, *classtocube(region)* are negative examples.

For the experiments presented in [18], we have selected 10 model instances (of database schemas), provided by an industrial partner, and describing several application domains *(invoices, sales, e-commerce, banking-investment, and so on)*. Concerning the experimentations of this paper, we use the *Microsoft AdventureWorks 2008R2* sample database family reference databases [49]. 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

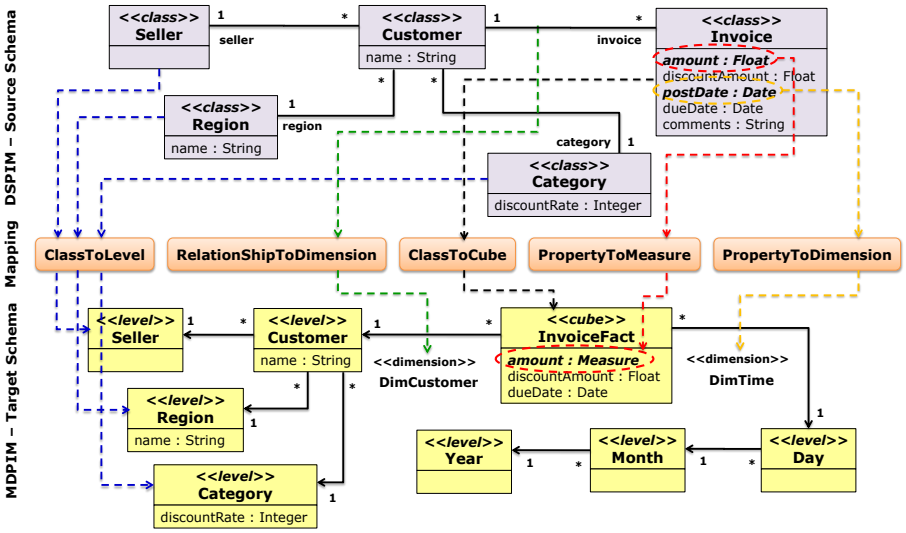


**Fig. 2.** Mapping UML CORE Instance (DSPIM) to CWM OLAP Instance (MDPIM).

(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.

Concerning the number of examples, we have $\|E_{ClassToCube}\| = 71$ denoting the number of example (positives and negatives) used to learn *ClassToCube* concept. Then, concerning *PropertyToMeasure*, $\|E_{PropertyToMeasure}\| = 249$, and for other concepts $\|E_{PropertyToDimension}\| = 245$, $\|E_{RelationShipToDimension}\| = 93$, $\|E_{ElementToHierarchyPath}\| = 338$, and $\|E_{ElementToDimensionLevel}\| = 338$. Average results from the 10x10-fold cross validation are then reported.

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, and (iii) a designation of one predicate as the "head" predicate to be learned. (iv) Lists of positive and negative examples 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. 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)).
```

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

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

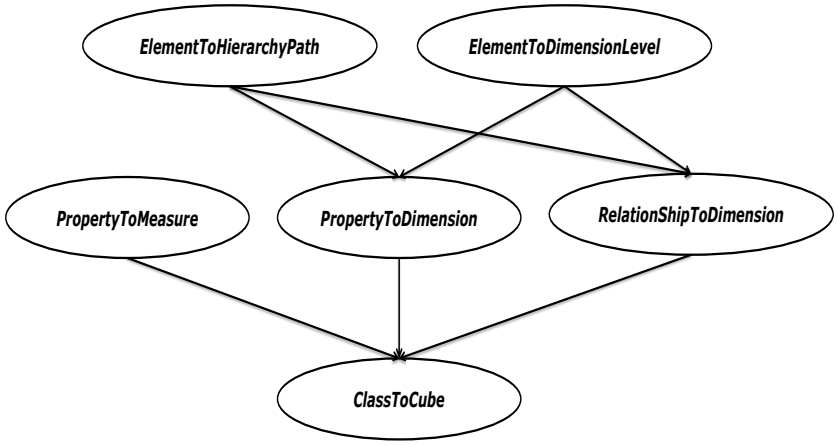


**Fig. 3.** The Considered Dependency Graph of Second Experiments.

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 The *Independent-Concept Learning (ICL)* and the *dependent-concept learning (with more different settings)* approaches. We identify the concept dependencies illustrated by the graph in figure 3:

– $ClassToCube \preceq PropertyToMeasure$: The *PropertyToMeasure* concept depends on the concept *ClassToCube*. In general, the context of 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* $\preceq$ *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.
- *ClassToCube* $\preceq$ *RelationShipToDimension*: Indeed, *dimensions* are, also, obtained from *relationships* of the *Class* that is transformed into *Cube*. The *CubeDimensionAssociation* meta-class relates a *Cube* to its defining dimensions as showed by the CWM OLAP metamodel in [18]. These relationships define the axes of analysis in the target multidimensional schema [97].
- *(PropertyToDimension, RelationShipToDimension)* $\preceq$ *ElementToHierarchyPath*: 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.

### 4.2 Results and Discussion

In the first experiments, we examined the accuracy of the learned rules to show the impact of the number of training models and examples and we report the obtained test accuracy curves for *ClassToCube* and *PropertyToDimension*. The accuracy of current experiments based on the new dataset (of AdventureWorks) confirm the results reported in [18]. 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. Accuracy is defined, based on the *contingency table* (or confusion matrix), as:

$$Accuracy = \frac{TP + TN}{P + N} \tag{1}$$

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).

For *ClassToCube*, Aleph induces the following rules with the best score:

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

For *PropertyToDimension*, Aleph induces the following rule with the best score:

```
propertytodimension(A) :-  property(A,date,1,1).
```

Concerning concepts *PropertyToMeasure*, *RelationshipToDimension* and *ClassToLevel*, Aleph induces the following rules with the best score. For each concept, the obtained rules include predicates of child-concepts in the dependency-graph, when learned in the *Dependent-Concept Learning* framework:

```
% PropertyToMeasure
propertytomeasure(A) :- associationOwnedAttribute(B,A),
          classtocube(B), property(A,float,1,1).
propertytomeasure(A) :-associationOwnedAttribute(B,A),
          classtocube(B), property(A,integer,1,1).

% RelationshipToDimension
relationshiptodimension(A) :- association(A,B,C), classtocube(B).

% ClassToLevel
classtolevel(A) :- association(B,C,A), relationshiptodimension(B).
classtolevel(A) :- association(B,C,A), classtocube(C),
          relationshiptodimension(B).
```

We note that the learned rules are close to the rules designed manually. We compare the resulted rules with those provided by related work. For example in [101], the source-model context of the proposed *EntityToCube* is formed by *Entity*, *RelationShipEnd (with multiplicity = '*')* and *Attribute (numeric types)* relations. Indeed, as it is shown above in the resulted *ClassToCube* transformation rule, we find also the relations: (*classtocube(A) :- associationOwnedAttribute(A,B), property(B,float,1,1), association(C,A,D).)* and *classtocube(A) :- associationOwnedAttribute(A,B), property(B,integer,1,1), association(C,A,D).*

Regarding the resulted rules, *associationOwnedAttribute(A,B) and association(C,A,D)* atoms are assiciated with *RelationShipEnd (with multiplicity = '*')* relation; then *property(B,float,1,1)* and *property(B,integer,1,1)* to *Attribute (numeric types)* relation. 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 resulted rules are found in a reasonable time. Indeed, we note 30s average search time for each concept in this configuration.

The *Receiver Operating Characteristics (ROC)* graphs are a useful technique for visualizing, organizing and selecting classifiers based on their performance [20].

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:

$$tp\ rate = \frac{TP}{P} \quad ; \quad fp\ rate = \frac{FP}{N} \tag{2}$$

Additional terms associated with ROC curves are sensitivity and specificity:

$$sensitivity = recall = \frac{TP}{P} \quad ; \quad specificity = 1 - fp\ rate = \frac{TN}{N} \tag{3}$$

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.

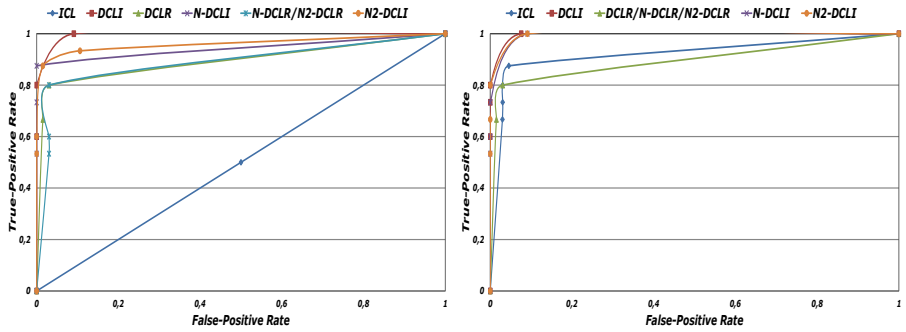


**Fig. 4.** Learning *PropertyToMeasure* (n=5 for left) and (n=10 for right).

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.

We use also, the *Area Under the ROC Curve (AUC)* as a common measure to compare the tested methods. The obtained results within current experiments are reported by figures 4, 5, 6, 7 and 8). Figures show that $n = 10$ setting (right part of each figure) gives best performances compared to $n = 5$. This confirms that the choice of this parameter is important to deal with noisy information of database models in general. Indeed, *data quality* and *conceptual models quality* [47, 36] play an important role in the design of *information systems*, and in particular *decision support systems*. Then, comparing ICL, DCLI and

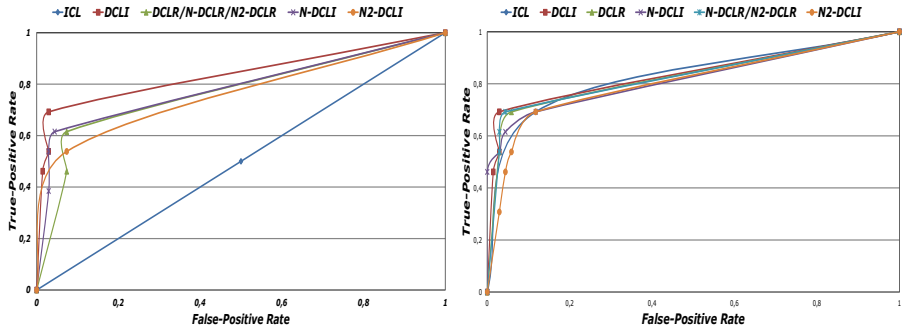


**Fig. 5.** Learning *PropertyToDimension* (n=5 for left) and (n=10 for right).

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 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.

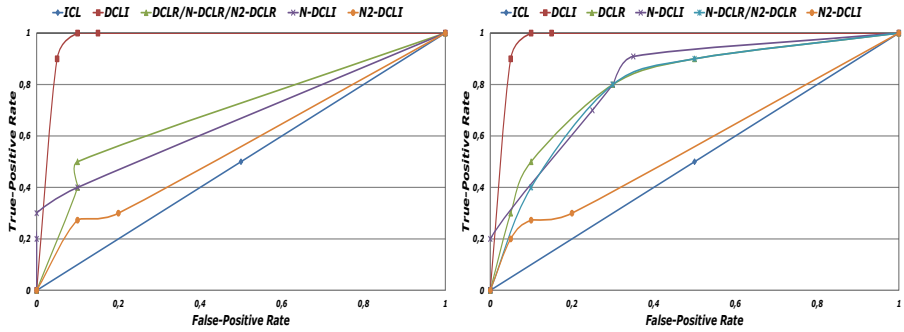


**Fig. 6.** Learning *RelationshipToDimension* (n=5 for left) and (n=10 for right).

Also, 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 *Relation-*

*shipToDimension*, results integrate the error rate from *ClassToCube* learned rules. 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 more important more when we learn top-level concepts (i.e., parent-concepts) in the dependency-graph. So in this case, the contribution of DCL becomes more significant. We observe for example that the gap is moez important for *ElementToHierarchyPath* and *ElementToDimensionLevel* concepts (in figures 7 and 8)) than *PropertyToMeasure* concept (in figure 4). 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.

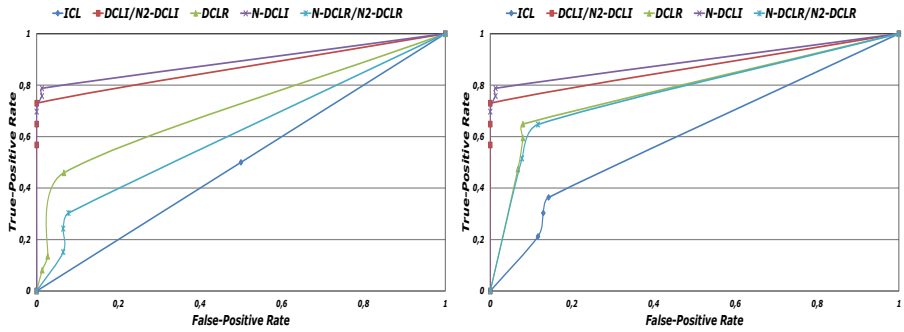


**Fig. 7.** Learning *ElementToHierarchyPath* (n=5 for left) and (n=10 for right).

Then, considering the N-DCLI/N2-DCLI and N-DCLR/N2-DCLR, 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 figures 5 and 6, *PropertyToDimension* and *RelationshipToDimension* are most impacted than *PropertyToMeasure* (in figure 4). 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.

Then, the *ElementToHierarchyPath* and *ElementToDimensionLevel* concepts are impacted by the noisy data of *ClassToCube*, but less than *PropertyToDimen-*

*sion* and *RelationshipToDimension*. We observe that *ElementToHierarchyPath* and *ElementToDimensionLevel* are not in direct dependence with *ClassToCube*.

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 behaviors 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 behavior of approaches that learn on noisy-datasets remains close to the DCLs working 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.

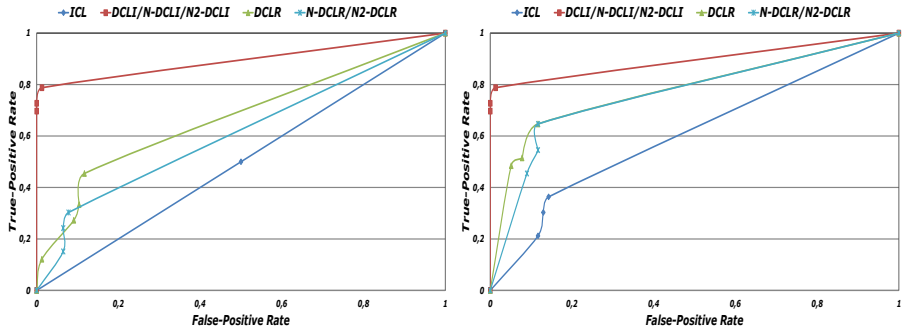


**Fig. 8.** Learning *ElementToDimensionLevel* (n=5 for left) and (n=10 for right).

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.

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 experiment, 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 average seek time of a concept in the first experiment is *30-32s* and it is estimated at *36-39s* for the second. 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.

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 partitioning by layer (or component) and a local partitioning (by design-level). The proposed frameworks, largely based on *Unified Modelling Language (UML)* and *Common Warehouse Metamodel (CWM)* 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 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. 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. 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 and the resulted information's allow defining a language bais (i.e., the transformations context) of and a search bias (i.e., the dependency-graph).

*Why DCL is better?* because adding a child-concept description allows the definition a new information or other context to consider when learning the parent-concept. This adds dependency information considered as an informational context 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, find first 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.

## 5 Future Research Directions

Our future work, regarding *Model-Driven Data Warehouse* automation will experiments the case when a business goals model is considered during transformations. For example, the derivation of the MDPIM from the pair (DSPIM, MDCIM), where MDCIM 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 (MDA)* and the conceptual transformation learning framework to knowledge engineering seems also an interrresting and a challenging future work. For example, a recent work [70] 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* [27] 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 [58] for web-data integration and data warehouses contextualization with documents [66]. 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 [27], 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 [98]. 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 [67], 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 [17]: 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 [65], the 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.

Metadata management for *Business Intelligence-as-a-Service* infrastructures and cloud-based databases will be 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 (OMG)* 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. In our future work, we will study in detail the CWM MIP in order to define new features that can impouve the proposed architecture.

The *Ontology Definition Metamodel (ODM)* has been used as a basis for ontology development as well as for generation of OWL ontologies. 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 ODM is used for ontology development and analysis on research in *context-aware* systems. As part of the OMG metamodeling architecture (ODM is a MOF-comliant 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 [21]. 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 require to apply innovative semantic reasoning techniques to security metrics and contextual information. We consider this as an interessting problem for model-driven approach and its application. Our innovative ideas around thees topics are discussed below.

Perspective, 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 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 [21, 93], where authors briefly explore some of the important related works [3, 26, 77, 76, 53, 33] and standards [61, 62] 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 represents 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.

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 [62] is an OMG 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 ODM 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 OMG security specifications [61] 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 MDA. 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 9 illustrate the workflow that we explain below.

**Semantic Reasoning:** the policies adaptation flow starts by a semantic analysis step. This step considers at the input several semantic conceptions: context, users activities/profiles, auditing, etc. These informations are represented in general by ontologies and/or rules. 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 policies models are conforming to the specifications cited above (ATLAS, RAD, etc.) or other profiles already recognized by the

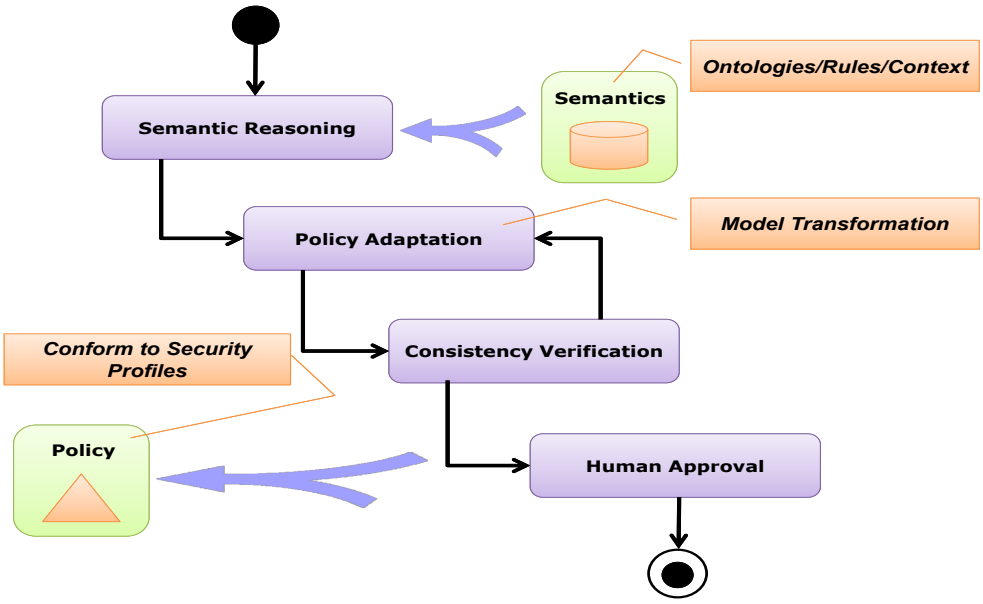


**Fig. 9.** Semantic Model-Driven Policy Adaptation.

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.

## 6 Conclusion

This chapter 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 data warehouses require a transformation phase of the models that necessitate a high level of expertise and a large time span to develop.

The main goal of this work 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.

The modelling step [14, 15], 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 (UML)* and the *Common Warehouse Metamodel (CWM)* 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)*. This DCL method is implemented using the Aleph ILP system 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.

## References

1. J. Abd-Ali and K. El Guemhioui. Automating Metamodel Mapping Using Machine Learning. In *3M4MDA*, pages 103–109, 2006.
2. E. Alpaydin. *Introduction to Machine Learning.* The MIT Press, 2nd edition, 2010.
3. D. A. Basin, J. Doser, and T. Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Trans. Softw. Eng. Methodol.*, 15(1):39–91, 2006.
4. J. Bézivin. Model driven engineering: An emerging technical space. In *GTTSE*, pages 36–64. Springer, 2006.

5. A. Bieszczad and K. Bieszczad. Contextual learning in the neurosolver. In *ICANN*, pages 474–484. Springer, 2006.
6. A. Cypher, D. C. Halbert, D. Kurlander, H. Lieberman, D. Maulsby, B. A. Myers, and A. Turransky, editors. *Watch what I do: programming by demonstration*. MIT Press, Cambridge, MA, USA, 1993.
7. K. Czarnecki and S. Helsen. Feature-based survey of model transformation approaches. *IBM Syst. J.*, 45:621–645, July 2006.
8. Z. Diskin and U. Wolter. A diagrammatic logic for object-oriented visual modeling. *Electr. Notes Theor. Comput. Sci.*, 203(6):19–41, 2008.
9. D. Djuric, V. Devedzic, and D. Gasevic. Adopting Software Engineering Trends in AI. *IEEE Intelligent Systems*, 22:59–66, 2007.
10. X. Dolques, M. Huchard, and C. Nebut. From transformation traces to transformation rules: Assisting model driven engineering approach with formal concept analysis. In *ICCS*, pages 093–106, Moscow, Russia, 2009.
11. Eclipse. The Model To Model (M2M) Transformation Framework., 2010.
12. M. Erwig. Toward the automatic derivation of xml transformations. In *XSDM*, pages 342–354. Springer, 2003.
13. F. Esposito, G. Semeraro, N. Fanizzi, and S. Ferilli. Multistrategy theory revision: Induction and abduction in inthelex. *Machine Learning*, 38(1-2):133–156, 2000.
14. M. Essaidi and A. Osmani. Data Warehouse Development Using MDA and 2TUP. In *SEDE*, pages 138–143. ISCA, 2009.
15. M. Essaidi and A. Osmani. Model driven data warehouse using MDA and 2TUP. *Journal of Computational Methods in Sciences and Engineering*, 10:119–134, 2010.
16. M. Essaidi and A. Osmani. Towards Model-driven Data Warehouse Automation using Machine Learning. In *IJCCI (ICEC)*, pages 380–383, Valencia, Spain, 2010. SciTePress.
17. M. Essaidi and A. Osmani. *Business Intelligence-as-a-Service: Studying the Functional and the Technical Architectures*, chapter 9, pages 199–221. IGI Global, 2012.
18. M. Essaidi, A. Osmani, and C. 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.
19. J.-R. Falleri, M. Huchard, M. Lafourcade, and C. Nebut. Metamodel matching for automatic model transformation generation. In *MoDELS*, pages 326–340, Berlin, Heidelberg, 2008. Springer-Verlag.
20. T. Fawcett. Roc graphs: Notes and practical considerations for researchers. Technical report, HP Laboratories, 2004.
21. E. Fernández-Medina, J. Jürjens, J. Trujillo, and S. Jajodia. Model-driven development for secure information systems. *Information & Software Technology*, 51(5):809–814, 2009.
22. J. Gama. Combining Classifiers by Constructive Induction. In *ECML*, pages 178–189. Springer, 1998.
23. J. a. Gama and P. Brazdil. Cascade Generalization. *Mach. Learn.*, 41:315–343, December 2000.
24. A. Gerber, M. Lawley, K. Raymond, J. Steel, and A. Wood. Transformation: The missing link of mda. In *ICGT*, pages 90–105. Springer, 2002.
25. S. M. Gustafson and W. H. Hsu. Layered Learning in Genetic Programming for a Cooperative Robot Soccer Problem. In *EuroGP*, pages 291–301, London, UK, 2001. Springer-Verlag.
26. M. Hafner, R. Breu, B. Agreiter, and A. Nowak. Sectet: an extensible framework for the realization of secure inter-organizational workflows. *Internet Research*, 16(5):491–506, 2006.
27. W. 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.
28. D. Jackson and A. P. Gibbons. Layered learning in boolean GP problems. In *EuroGP*, pages 148–159. Springer-Verlag, 2007.
29. F. Jouault, F. Allilaire, J. Bézivin, and I. Kurtev. Atl: A model transformation tool. *Sci. Comput. Program.*, 72(1-2):31–39, 2008.
30. F. Jouault and J. Bézivin. KM3: A DSL for Metamodel Specification. In *FMOODS*, pages 171–185. Springer, 2006.
31. F. Jouault and I. Kurtev. Transforming models with atl. In *MoDELS Satellite Events*, pages 128–138. Springer, 2005.

32. C. Kaldeich and J. O. e Sá. Data Warehouse Methodology: A Process Driven Approach. In *CAiSE*, pages 536–549. Springer, 2004.
33. S. Kallel, A. Charfi, M. Mezini, M. Jmaiel, and A. Sewe. A holistic approach for access control policies: from formal specification to aspect-based enforcement. *Int. J. Inf. Comput. Secur.*
34. E. Kapsammer, H. Kargl, G. Kramler, T. Reiter, W. Retschitzegger, and M. Wimmer. Lifting metamodels to ontologies - a step to the semantic integration of modeling languages. In *MoDELS/UML*, pages 528–542. Springer, 2006.
35. S. Kent. Model driven language engineering. *Electr. Notes Theor. Comput. Sci.*, 72(4), 2003.
36. G. Kersulec, S. S.-S. Cherfi, I. Comyn-Wattiau, and J. Akoka. Un environnement pour lévaluation et lamélioration de la qualité des modèles de systèmes dinformation. In *INFORSID*, pages 329–344, 2009.
37. M. Kessentini, H. Sahraoui, and M. Boukadoum. Model Transformation as an Optimization Problem. In *MoDELS*, pages 159–173, Berlin, Heidelberg, 2008. Springer-Verlag.
38. M. Kessentini, H. Sahraoui, and M. Boukadoum. Méta-modélisation de la transformation de modéles par l'exemple : approche par méta-heuristiques. In *LMO*, 2009.
39. M. Kessentini, M. Wimmer, H. Sahraoui, and M. Boukadoum. Generating transformation rules from examples for behavioral models. In *BM-FA*, pages 2:1–2:7. ACM, 2010.
40. R. Kimball and M. Ross. *The Data Warehouse Toolkit: The Complete Guide to Dimensional Modeling*. John Wiley & Sons, Inc., New York, USA, 2002.
41. R. Kimball and M. Ross. *The Kimball Group Reader: Relentlessly Practical Tools for Data Warehousing and Business Intelligence*. John Wiley & Sons, Inc., New York, USA, 2010.
42. V. Kulkarni, S. Reddy, and A. Rajbhoj. Scaling up model driven engineering - experience and lessons learnt. In *MoDELS*, pages 331–345. Springer, 2010.
43. N. Lavrac and S. Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994.
44. 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.
45. 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.
46. J.-N. Mazón and J. Trujillo. An mda approach for the development of data warehouses. *Decis. Support Syst.*, 45:41–58, 2008.
47. K. Mehmood, S. S.-S. Cherfi, and I. 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.
48. S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE*, pages 117–128. IEEE Computer Society, 2002.
49. Microsoft. Microsoft AdventureWorks 2008R2, 2011.
50. J. Miller and J. Mukerji. MDA Guide Version 1.0.1. Technical report, Object Management Group (OMG), 2003.
51. T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–226, 1982.
52. T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.
53. N. Moebius, K. Stenzel, and W. Reif. Generating formal specifications for security-critical applications - a model-driven approach. In *IWSESS*, 2009.
54. S. Muggleton. Inductive Logic Programming. *New Generation Computing*, 8:295–318, 1991.
55. S. Muggleton. Optimal Layered Learning: A PAC Approach to Incremental Sampling. In *ALT*, pages 37–44, London, UK, 1993. Springer-Verlag.
56. S. Muggleton and L. D. Raedt. Inductive logic programming: Theory and methods. *J. Log. Program.*, 19/20:629–679, 1994.
57. S. Muggleton and K. Road. Predicate Invention and Utilisation. *Journal of Experimental and Theoretical Artificial Intelligence*, 6:6–1, 1994.
58. V. Nebot and R. B. Llavori. Building data warehouses with semantic data. In *EDBT/ICDT Workshops*. ACM, 2010.
59. S. H. Nguyen, J. G. Bazan, A. Skowron, and H. S. Nguyen. Layered Learning for Concept Synthesis. *T. Rough Sets*, 3100:187–208, 2004.

60. S.-H. Nienhuys-Cheng and R. de Wolf. *Foundations of Inductive Logic Programming*, chapter 8, pages 127–159. Springer, 1997.
61. O. M. G. (OMG). The OMG Security Specifications Catalog., 2008.
62. O. M. G. (OMG). The Ontology Definition Metamodel (ODM) Specification., 2009.
63. O. M. G. (OMG). The Query/View/Transformation (QVT) Specification., 2010.
64. K. Ono, T. Koyanagi, M. Abe, and M. Hori. Xslt stylesheet generation by example with wysiwyg editing. In *SAINT*, pages 150–161, Washington, DC, USA, 2002. IEEE Computer Society.
65. T. B. Pedersen. Warehousing the world: a few remaining challenges. In *DOLAP*, pages 101–102, New York, NY, USA, 2007. ACM.
66. J. M. Pérez, R. Berlanga, and M. J. Aramburu. A relevance model for a data warehouse contextualized with documents. *Inf. Process. Manage.*, 45(3):356–367, 2009.
67. Philip Russom. Next Generation Data Warehouse Platforms. `http://www.oracle.com/database/docs/tdwi-nextgen-platforms.pdf`, 2009.
68. V. Poe, S. Brobst, and P. Klauer. *Building a Data Warehouse for Decision Support*. Prentice-Hall, Inc., Upper Saddle River, USA, 1997.
69. N. Prat, J. Akoka, and I. Comyn-Wattiau. A UML-based data warehouse design method. *Decis. Support Syst.*, 42(3):1449–1473, 2006.
70. N. Prat, J. Akoka, and I. Comyn-Wattiau. An MDA Approach to Knowledge Engineering. *Expert Syst. Appl.*, 39(12):10420–10437, 2012.
71. A. Repenning and C. Perrone. Programming by example: programming by analogous examples. *Commun. ACM*, 43(3):90–97, 2000.
72. R. Rios and S. Matwin. Predicate Invention from a Few Examples. In *AI*, pages 455–466, London, UK, 1998. Springer-Verlag.
73. S. Roser and B. Bauer. An approach to automatically generated model transformations using ontology engineering space. In *SWESE*, 2006.
74. A. Rutle, A. Rossini, Y. Lamo, and U. Wolter. A diagrammatic formalisation of mof-based modelling languages. In *TOOLS*, pages 37–56. Springer, 2009.
75. A. Rutle, U. Wolter, and Y. Lamo. A diagrammatic approach to model transformations. In *EATIS*, 2008.
76. F. Seehusen and K. Stølen. Maintaining information flow security under refinement and transformation. In *FAST*, 2007.
77. F. Seehusen and K. Stølen. A transformational approach to facilitate monitoring of high-level policies. In *POLICY*, pages 70–73. IEEE Computer Society, 2008.
78. A. Simitsis. Mapping conceptual to logical models for ETL processes. In *DOLAP*, pages 67–76. ACM, 2005.
79. A. Srinivasan. A learning engine for proposing hypotheses (Aleph). http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph, 2006.
80. I. Stahl. On the Utility of Predicate Invention in Inductive Logic Programming. In *ECML*, pages 272–286. Springer, 1994.
81. I. Stahl. The Appropriateness of Predicate Invention as Bias Shift Operation in ILP. *Mach. Learn.*, 20:95–117, July 1995.
82. P. Stevens. Bidirectional model transformations in QVT: semantic issues and open questions. *Software and System Modeling*, 9(1):7–20, 2010.
83. P. Stone and M. M. Veloso. Layered learning. In *ECML*, pages 369–381. Springer, 2000.
84. M. Strommer, M. Murzek, and M. Wimmer. Applying model transformation by-example on business process modeling languages. In *ER*, pages 116–125, Berlin, Heidelberg, 2007. Springer-Verlag.
85. Y. Sun, J. White, and J. Gray. Model transformation by demonstration. In *MoDELS*, pages 712–726. Springer, 2009.
86. K. M. Ting and I. H. Witten. Stacked generalization: when does it work? In *IJCAI*, pages 866–871. Morgan Kaufmann, 1997.
87. K. M. Ting and I. H. Witten. Issues in stacked generalization. *J. Artif. Intell. Res. (JAIR)*, 10:271–289, 1999.
88. E. Turban, R. Sharda, and D. Delen. *Decision Support and Business Intelligence Systems*. Prentice Hall, 2010.
89. P. D. Turney. Exploiting context when learning to classify. In *ECML*, pages 402–407, London, UK, 1993. Springer-Verlag.
90. D. Varró. Model Transformation by Example. In *MoDELS*, pages 410–424, Genova, Italy, October 2006. Springer.

91. D. Varró. Model transformation by example. In *MoDELS*, pages 410–424. Springer, 2006.
92. 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.
93. R. Villarroel, E. Fernández-Medina, and M. Piattini. Secure information systems development - a survey and comparison. *Computers & Security*, 24(4):308–321, 2005.
94. P. Westerman. *Data warehousing: using the Wal-Mart model*. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2001.
95. M. Wimmer, M. Strommer, H. Kargl, and G. Kramler. Towards Model Transformation Generation By-Example. In *HICSS*, page 285b, Washington, DC, USA, 2007. IEEE Computer Society.
96. D. H. Wolpert. Stacked Generalization. *Neural Networks*, 5:241–259, 1992.
97. R. Wrembel and C. Koncilia. *Data Warehouses and OLAP: Concepts, Architectures and Solutions*. IGI Global, 2007.
98. L. Wu, G. Barash, and C. Bartolini. A service-oriented architecture for business intelligence. *Service-Oriented Computing and Applications*, 0:279–285, 2007.
99. Z. Xie. Several Speed-Up Variants of Cascade Generalization. In *FSKD*, pages 536–540, Xi'an, China, September 2006. Springer.
100. L. L. Yan, R. J. Miller, L. M. Haas, and R. Fagin. Data-driven understanding and refinement of schema mappings. In *SIGMOD*, pages 485–496, New York, NY, USA, 2001. ACM.
101. L. Zepeda, M. Celma, and R. Zatarain. A Mixed Approach for Data Warehouse Conceptual Design with MDA. In *ICCSA*, pages 1204–1217, Perugia, Italy, June 2008. Springer-Verlag.
102. D. Zhang and J. J. P. Tsai. *Advances in Machine Learning Applications in Software Engineering*. IGI Publishing, Hershey, PA, USA, 2007.
103. M. M. Zloof. Query-by-example: the invocation and definition of tables and forms. In *VLDB*, pages 1–24, New York, NY, USA, 1975. ACM.