Journal of Information & Knowledge Management
Vol. 18, No. 2 (2019) 1950025 (37 pages)
© World Scientific Publishing Co.
DOI: 10.1142/S0219649219500254

A Framework for Knowledge Models Transformation: A Step Towards Knowledge Integration and Warehousing

Rim Ayadi*,§, Yasser Hachaichi*,† and Jamel Feki*,‡
*Multimedia Information Systems and Advanced Computing Laboratory
University of Sfax, Sfax, Tunisia
†Department of Computer Science and Quantitative Methods
Higher Institute of Business Administration of Sfax, Sfax, Tunisia
‡Faculty of Computing and IT, University of Jeddah
Jeddah, Saudi Arabia
§rim.ayadi@yahoo.fr

Published 10 June 2019

Abstract. An intelligent decision support system should based on a knowledge warehouse (KW). A KW gathers knowledge initially expressed in different formalisms and therefore heterogeneous. Consequently, the KW building process requires knowledge homogenisation. This paper deals with this main issue; it introduces a three-layer architecture for a KW; more precisely, it focuses on the first layer architecture called *Knowledge Acquisition and Transformation*. This layer aims to transform heterogeneous knowledge models into the MOT (Modeling with Object Types) semi-formal language [Paquette, G (2002). *Knowledge and Skills Modeling: A Graphical Language for Designing and Learning*. Sainte-Foy: University of Quebec Press (in French).] that we have selected as a pivot knowledge model. For this transformation step, first, we design four meta-models; one for MOT and one for each of the three explicit knowledge models, namely, *decision tree*, *association rules* and *clustering*. Secondly, we define 15 transformation rules that we formalise in ATL (Atlas Transformation Language). Finally, we exemplify the knowledge transformation in order to show its usefulness for the KW building process.

Keywords: Knowledge warehouse; data mining; knowledge model; heterogeneous knowledge models; transformation rules; MOT language.

1. Introduction

Data sources (e.g. database, data warehouse) are data-rich, but knowledge-poor as they are designed for data and therefore are completely inappropriate for knowledge storage and management. However, they remain useful as we can extract valuable knowledge from these sources in order to boost the efficiency of the enterprise decision-making process.

On the one hand, the enterprise performance depends closely on professional skills, experience and tacit knowledge held by individuals. On the other hand, it

§Corresponding author.

depends on explicit knowledge extracted from data. However, in practice and in almost enterprises, managers involved in the decision-making process rarely share their knowledge, thus making themselves the main repositories of knowledge. Consequently, an important problem arises since these people take their knowledge with them when they leave (Dymond, 2002). Moreover, even if the enterprise knowledge is extracted or elicited, problems with knowing that it exists occur, where it is located, how to share it and how to represent it for judicious exploitation.

Actually, knowledge is often expressed in heterogeneous formats depending on the knowledge origin: knowledge acquired through collaboration and interaction between individuals or knowledge extracted from data sources using automated techniques such as data mining (e.g. decision trees, association rules, clustering) (Silwattananusarn and Tuamsuk, 2012). Knowledge is generally used for a given situation in order to make a decision, but it will not be necessarily reused in other similar situations. In addition, it is stored in various computer systems dispersed in one or more companies, without being explicitly interconnected/interrelated and in particular without the existence of a pivot model of representation and exploitation.

Hence, the need for knowledge management (KM) (Nemati *et al.*, 2002; Liebowitz and Frank, 2016) and a new structure to gather knowledge are increasingly felt. For this, we consider that collecting, extracting, memorising and representing knowledge for intelligent use (i.e. the aim of KM) is a stimulating and motivating concern. The main objectives are (i) to help companies to improve their efficiency and profitability by relying their decisions on useful knowledge; (ii) increase the skills of experts and their capacity for creativity in order to anticipate developments and innovations and (iii) control/monitor the internal and external environment (customers, suppliers, competitors, etc.) of companies to manage better their competitive situations.

To meet these requirements, the concept of knowledge warehouse (KW) (Ayadi *et al.*, 2013) has emerged as a technological solution for gathering, storing and sharing explicit knowledge between decision-makers; it is an appropriate computer infrastructure for an intelligent decision support system. A KW intends to accumulate and manage knowledge issued from multiple sources and initially of heterogeneous types, formats and codifications.

In this context, we propose a three-layer architecture for the KW (cf. Sec. 3.1); these layers are (i) *knowledge acquisition and transformation*, (ii) *knowledge storage* and (iii) *knowledge exploitation and maintenance*. This paper details the first layer that aims to homogenise heterogeneous knowledge by transforming them into a common pivot knowledge model, namely the MOT language (Modeling with Object Types) (Paquette, 2002). More precisely, it focuses on transforming explicit knowledge into MOT. To do so, we design four meta-models, one meta-model for MOT and one for each of the three popular explicit knowledge models (i.e. *decision trees*, *association rules* and *clustering*) we expect to load into the KW.

Our choice for MOT language is two-fold: First, MOT is a semi-formal language for graphical representation of knowledge; being graphic, it facilitates the modeling of explicit knowledge of various formats as well as the articulation of experts' tacit knowledge. Secondly, MOT defines a set of four standard *typed knowledge units* called *conceptual*, *procedural*, *strategic* and *factual*. These knowledge units allow modelling all types of knowledge models as decision trees, association rules, etc.

This paper answers *How to put together initially heterogeneous explicit knowledge in a KW*? The answer passes through unifying and integrating different models of knowledge. For the KW construction process, such unification performs through knowledge transformations into the pivot MOT model. Indeed, we transform automatically three explicit knowledge models, namely. *decision trees*, *association rules* and *clustering* into MOT. For these automatic transformations, we define a set of rules based on structural correspondences between each knowledge meta-model (MM) and the MOT target MM. We formalise and implement these transformation rules using ATL (Atlas Transformation Language) (ATLAS, 2006). Note that actually, the transformation process is not our ultimate objective; in fact, it is a means to prepare for the integration step.

This paper is organised as follows: Sec. 2 is devoted to presenting key concepts and work related to the research topic of this paper. Section 3 introduces our architecture for a KW, the MOT modeling language along with its meta-model. Section 4 is an overview of models transformation. Sections 5–7 define a MM for each of the three popular knowledge models treated in this paper. For each MM, we define a set of ATL transformation rules. Section 8 presents an introductory example in order to show the usefulness of the present work and its continuity. Finally, Sec. 9 concludes this paper; it positions the progress of this research and enumerates its ongoing extensions.

2. Key Concepts and Related Work

In today's competitive environment, companies are increasingly forced to meet the expectations of their partners in a flexible manner through an effective exploitation of knowledge during their decision-making process. Thus, many companies have spent time, efforts and money to discover important information about themselves, their customers and their competitors (Dymond, 2002). Therefore, knowledge is a powerful IT capability that helps companies to improve and succeed their decision-making process and to manage better competitive situations. As stated by Peter F. Drucker (Drucker, 1993; Michael, 1999): "Knowledge is now fast becoming the sole factor of production, sidelining both capital and labor". From this context was born the need for KM.

2.1. *Key concepts*

To introduce the objective of KM, it is essential to understand what knowledge is. In fact, the distinction between the technical terms data, information and knowledge are slightly tenuous, but it is important to clarify the meanings at this step.

- *Data*: are fundamental elements, qualitative or quantitative (e.g. age, size, amount), which are collected and measured to describe an event as simple observations (Davenport and Prusak, 1998; Chauvet and Ghetty, 2003). They represent the computerisation of daily life and can be considered as the atoms of knowledge (Michael, 1999).
- *Information*: is a datum that makes sense and leads to understanding (Davenport and Prusak, 1998; Michael, 1999; Chauvet and Ghetty, 2003). Indeed, a datum in the abstract does not provide information, while it is a datum expressed in a conceptual context. In general, information is considered organised and sorted data, which can be used to answer a specific question. It is the aggregation of data (e.g. averages, trends, percentages) (Michael, 1999).
- *Knowledge*: is new information that has a higher level of aggregation and interpretation; it originates from the person's brain. It is built with information validated through rules and tests, acquired by an intelligent process and enriched by personal experience, synthesis and beliefs. This joins the equation proposed by (Mach, 1995) and (Chauvet and Ghetty, 2003): "*Knowledge = Information + Human Interpretation*". In addition, knowledge may include work procedures and processes, details and conceptual relationships between subjects in a given domain, glossaries, concept trees, dependence rules, etc. (Michael, 1999; Suciu *et al.*, 2012).

A simpler and more understandable definition for these concepts has been suggested by (Huang *et al.*, 1999), which is "*Data are collected, sorted, grouped, analyzed and interpreted. When data are processed in this manner, they become information. Information contains substance and purpose. Knowledge is generated when information is combined with context and experience*" (Michael, 1999).

In this respect, KM is the operation of adding value to information by capturing tacit knowledge and converting/transforming it into explicit knowledge, then by filtering, memorising, retrieving, disseminating and sharing explicit knowledge and finally by creating and testing new knowledge (Nemati *et al.*, 2002). This knowledge creation/conversion model, known as the knowledge spiral (Nonaka, 1991; Nonaka and Takeuchi, 1995), is based on the distinction between two types of knowledge:

- *Tacit knowledge*: is a personal knowledge that has an important cognitive dimension (Nonaka, 1991). It includes the views, beliefs, perspectives and mental models ingrained in a person's mind (Nonaka, 1991; Nemati *et al.*, 2002). Thus, it consists of acquired skills, expertise, intuitions, experiences, etc. This knowledge is difficult to articulate (Nonaka, 1991) or formalise in a form useable by other people as opposed to explicit knowledge.
- *Explicit knowledge*: is knowledge that can be clearly articulated and codified; it is formal and systematic (Nonaka, 1991) using a system of languages, symbols, rules, objects or equations. Thus, this knowledge is easily memorised, communicated to others, shared (Nonaka, 1991) and physically transferable because it appears in

a tangible form (e.g. written procedures, universal principles, mathematical models).

Synergistic relationship and the interaction between tacit and explicit knowledge are proposed (Nonaka, 1991) in order to grant continuous innovation within companies:

Socialisation (*tacit to tacit*) is the creation of new tacit knowledge from other tacit knowledge, through ideas, experiences and technical skills shared by several members of the company. This exchange of tacit knowledge is achieved through observation, imitation and practice (i.e. without using a language). We can relate this process to learning (Chauvet and Ghetty, 2003).

Articulation (*tacit to explicit*) is the conversion of tacit knowledge to explicit knowledge through concepts, hypotheses, models, etc.

Combination (*explicit to explicit*) is the combination and the integration of several types of explicit knowledge to create new explicit knowledge such as new patterns and new relations.

Internalisation (*explicit to tacit*) is the conversion of existing explicit knowledge, shared and improved during discussions, into new tacit knowledge to update and extend the decision-maker's own tacit knowledge. This process is significantly related to learning by the practice (Chauvet and Ghetty, 2003). Explicit knowledge is implanted in sequences that can reach the stage of reflex and automatism.

2.2. *Related work*

For business intelligence purposes, we suggest to gather knowledge (i.e. tacit knowledge captured and explicit knowledge) into a KW that we suggest as a solution to implement, apply and improve all phases of the KM process. The KW architecture should provide the necessary infrastructure to capture, organise and memorise explicit knowledge and improve the sharing and exploitation of this knowledge for intelligent decision-making activities across companies (Nemati *et al.*, 2002; Ayadi *et al.*, 2013).

Several related works of the literature (Michael, 1999; Kerschberg, 2001; Dymond, 2002; Nemati *et al.*, 2002; Zhang and Liang, 2006; Qing-Lan and Zhi-Jun, 2009; Irfan and Uddin-Shaikh, 2010; Hussain *et al.*, 2012; Hamad and Qader, 2014) have introduced the KW concept and proposed basic architectures for it.

Generally, these works provide non-detailed architecture and consider that the KW concept is analogous to the data warehouse (Michael, 1999; Dymond, 2002; Nemati *et al.*, 2002) that can be viewed as a three-part process of capturing, storing and accessing data. Thus, this three-part data warehouse "bowtie" structure is also found in the KW (Dymond, 2002).

In fact, most of these architectures, often reflecting a particular point of view, consist mainly of three layers (Kerschberg, 2001; Dymond, 2002; Nemati *et al.*, 2002; Zhang and Liang, 2006; Qing-Lan and Zhi-Jun, 2009):

- *Data sources layer*: consists of the company's internal data sources (e.g. documents, databases, data warehouse, electronic messages and website repository) and external data such as web services that can be used to augment internal data.
- *KM layer*: allows the creation of the appropriate "knowledge repository" for the company, by using various services such as data mining services, security services, ontology services. It also depicts the processes that acquire, refine/transform and store knowledge in the repository.
- *Knowledge presentation layer*: allows knowledge workers to obtain personalised information by means of the Knowledge Portal. The latter must support the communication and collaboration of users to share knowledge and to combine their tacit and explicit knowledge.

In addition, Michael (1999) considers the KW as *Knowledge Components* defined at the smallest level for knowledge decomposition. These knowledge components are cataloged and stored for reuse. In Michael (1999), the author has proposed some knowledge components issued from instructional design theories such as *Generality, Example, Analogy,* .... *Generality* is a statement or diagram that applies to all instances, such as a definition of a concept or a flowchart for a procedure, *Example* is a specific instance of a concept, procedure or principle, whereas *Analogy* is a comparison of two objects, which states their similarities and differences. However, these knowledge components can be stored in several physical places; i.e. not gathered in a common repository.

In Dymond (2002), the author considers the KW as a knowledge base modelled like a tree where nodes are objects. Each object has a name, attributes and methods. Methods are executed according to tree search algorithms. These algorithms are procedures to navigate the tree.

In Hussain *et al.* (2012), the authors propose a basic architecture for a KW. Modules of this architecture extract knowledge from sources (e.g. decision trees, knowledge maps, production rules) and then convert them into a common format based on objects. They suggest storing objects into an object text file having object name, attribute, procedures, generalisation, composition, aggregation, association and relationship.

Hamad and Qader (2014) build a KW using only association rules, as a data mining technique. The form of this knowledge is used to achieve decision-making process by finding hidden relations (rules) and predicting future events from large amounts of data. The logical structures for storing knowledge in the KW are analogous to data warehouse tables.

In general, authors do not tackle how decision-makers update and exploit knowledge stored in the KW. In addition, the majority of the literature works do not specify the format of the knowledge to be stored in the KW nor how to unify heterogeneous knowledge as well as how to integrate them. Furthermore, some of these works (Dymond, 2002; Hussain *et al.*, 2012; Hamad and Qader, 2014) are not interested in tacit knowledge while building the KW. Consequently, no methods for capturing and transforming tacit knowledge into explicit knowledge have been addressed so far.

Relying on the conclusions from the studied works, we propose, in the remaining of this paper, our complete vision of what should be a KW and then we suggest an appropriate architecture.

3. The KW and the MOT Language

As mentioned before, our objective is to build a KW by federating heterogeneous knowledge into a KW that gathers explicit knowledge possibly coming from multiple sources, having heterogeneous formats and relating to several activities of a business domain. The KW is prepared (i.e. creation, organisation, storage, presentation and sharing) in order to support an intelligent decision-making process (Ayadi *et al.*, 2013).

3.1. *KW architecture*

The KW is a common repository where knowledge collected from different sources is prepared, homogenised and stored for an intelligent decision-making process. Figure 1 depicts our three-layer architecture for a KW: *knowledge acquisition and transformation*, *knowledge storage* and *knowledge exploitation and maintenance*. This architecture encompasses seven modules (rectangular blocks in Fig. 1) that we explain hereafter. In Fig. 1, arrows link a module to its input and its output.

(1) *Data preparation module*: It enables collecting and cleaning heterogeneous-format data sets as databases, data warehouses and flat files.
(2) *Knowledge extraction module*: It extracts the disseminated knowledge from the cleaned data using data mining techniques (i.e. decision trees, association rules, clustering (Silwattananusarn and Tuamsuk, 2012; Zaki and Meira, 2014)). In addition, it converts knowledge hidden in the initial data into explicit knowledge formally expressed in an appropriate model.
(3) *Tacit knowledge capture and explicitness module*: This module allows the articulation (cf. Sec. 2.1) of tacit knowledge to explicit knowledge. First, we capture knowledge either from observations (e.g. observation of a production process) or from oriented interviews (i.e. cognitive knowledge and ideas gained from experts). Secondly, we define a set of rules that convert, on the one hand, knowledge issued from observations into decision trees modelling successive steps/actions, and on the other hand, knowledge issued from oriented interviews into association rules (i.e. *If* condition *Then* result). Finally, we solve the possible conflicts between elicited knowledge and then integrate the result into the KW.
(4) *Knowledge normalisation module*: It aims to transform the elicited tacit knowledge and the explicit knowledge into a common pivot model (e.g. MOT).
(5) *Knowledge integration/unification module*: It unifies knowledge modelled in MOT and then records them into a global common repository. To do this,

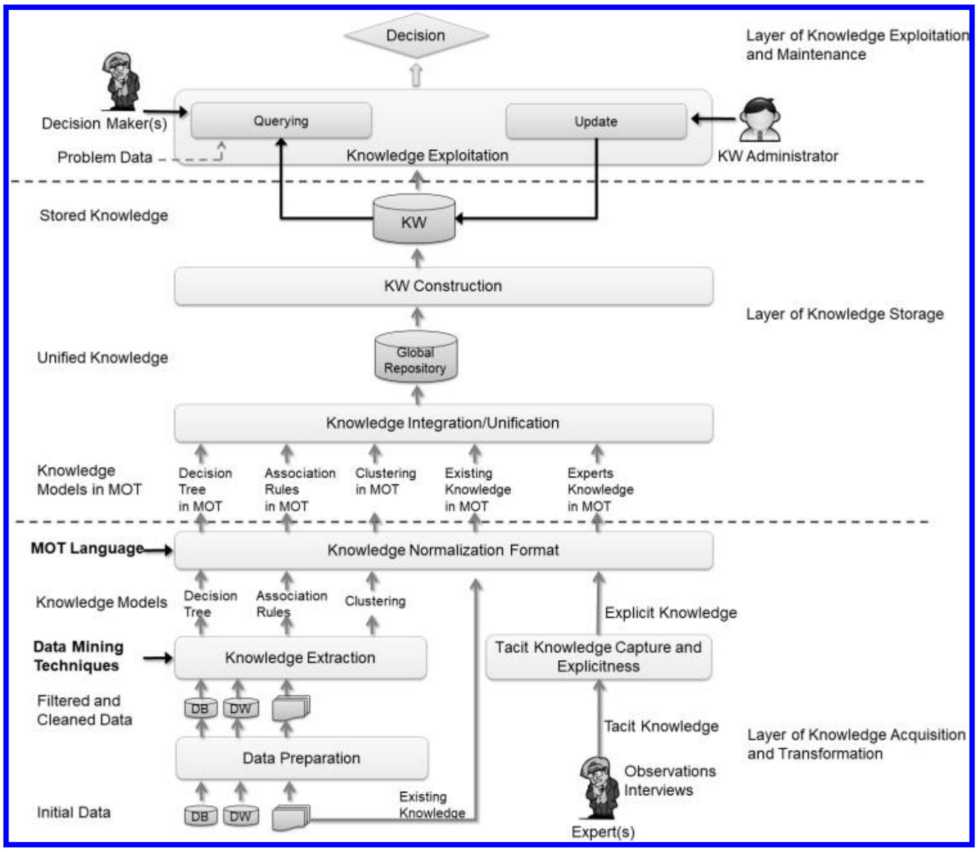


Fig. 1. Architecture of a KW (Ayadi *et al.*, 2013).

we identify the semantic relationships between knowledge such as identity, similarity, hyperonymy and meronymy (Ayadi *et al.*, 2017b). This module helps to exploit knowledge, infer new knowledge and ultimately remove redundancy.

(6) *KW construction module*: This module stores harmonised knowledge in the KW. It should enable the KW Administrator managing knowledge evolution.

(7) *Knowledge exploitation module*: It is the knowledge manipulation tool. It enables decision-makers querying the KW and the KW system to learn new decisions. Indeed, when the decision-maker is satisfied with a decision suggested by the KW system, (s)he can confirm the correctness of the decision. Otherwise, the decision-maker will be able to modify the decision and then inform the KW administrator to upgrade the KW with this human decision and its context; this helps its use by decision-makers having similar profile (a collaborative module is useful, it is beyond the objective of this paper).

The remainder of this paper focuses on the *knowledge normalisation* module for which we need a pivot language for modelling heterogeneous knowledge models (explicit knowledge models, namely, *decision tree*, *association rules* and *clustering*). In fact, there are several languages for knowledge description: *Informal*, *formal* and *semi-formal* languages.

Informal languages are natural language-based; consequently, their main lack is their ambiguity and inaccuracy.

A *formal* language has a rigid syntax vocabulary based on symbols, letters or lexemes[1]; it is hard to understand by novice people. Furthermore, it is very difficult to extract an intuition about the functionality of a system given the huge amount of details of a formal specification only.

Finally, a *semi-formal* language has a graphic formalism for knowledge expression; its rich grammar expressiveness and simplicity make it easy to use by knowledge experts, so that we can involve them during the modelling process of knowledge. It offers many advantages compared with an informal language since it provides a representational guide that helps structuring the modelling process and facilitating the articulation of experts' tacit knowledge. Compared with a formal language, a semi-formal one helps to enlarge people to express their knowledge with less assistance from knowledge engineers. Therefore, a semi-formal language saves efforts and time during knowledge elicitation. These benefits encouraged us to elect a semi-formal language for knowledge representation.

Actually, there are several formalisms for knowledge representation such as semantic networks, semantic trees, entity-relationship models, information flow models, object-oriented models, MOT, concept maps, etc. (Paquette, 1996). In the following, we provide the main characteristics for the four commonly used semi-formal languages (Paquette, 2002): *semantic trees, concept maps, semantic networks* and *MOT*.

(1) Semantic tree (Dinarelli *et al.*, 2010): enables a hierarchical classification of concepts. However, trees lead to ambiguous interpretation because they clarify neither the direction nor the type of links between concepts.
(2) *Concept map* (Canas *et al.*, 2001): can represent the relationships between concepts using labelled but non-oriented links; this complicates the interpretation of links. In addition, an inaccurate label can make the knowledge transfer process difficult.
(3) *Semantic network* (Collins and Quillian, 1969): although semantic networks orient and label links between knowledge, they suffer from the lack of standard terminology for the graphical representation.
(4) *MOT* (modelling with object types) (Paquette, 2002): is a language suitable for non-computer skilled designers for knowledge modelling. MOT offers the feature of knowledge modelling according to two levels of abstraction: facts and abstract

[1]Lexeme: a basic lexical unit of a language, consisting of one word or several words, considered as an abstract unit and applied to a family of words related by form or meaning

> knowledge. The latter is, in turn, divided into three types: *concepts*, *procedures* and *principles*. Usually, these types of knowledge are the subject of several taxonomies of knowledge models that are actually complementary but different (Paquette, 1996). Therefore, the integration of various types of knowledge, based on oriented and typed links, into a single model is a major benefit of the MOT language; it offers an easy formalism that helps experts to build a global and coherent vision of the main processes, concepts and strategies in order to formalise their knowledge (Paquette, 1996).

Relying on its offered advantages, we select the MOT as a pivot language for representing knowledge initially expressed in different formats. In what follows, we introduce the MOT language and we develop its meta-model.

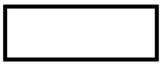
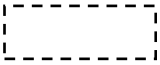
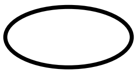
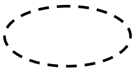
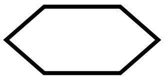
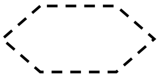
3.2. *Modelling with object types (MOT) language*

MOT is a product of the research center LICEF[2] (Paquette, 2002, 2010). Its use of typed entities along with the ability of representing different types of knowledge at two abstraction levels (i.e. abstract and factual) within the same schema make MOT an efficient tool capable to represent all types of explicit knowledge as well as the articulation of experts' tacit knowledge. Although it is semi-formal, MOT offers a graphical notation and typed objects to associate semantics to entities and relationships (Héon *et al.*, 2010).

3.2.1. *Knowledge and relationship types in MOT*

MOT classifies knowledge into two categories of knowledge: *abstract* and *factual*. Table 1 illustrates the three types of the *abstract* category of knowledge (*concept*, *procedure* and *principle*) and their corresponding types of the *factual* category of knowledge (*example*, *trace* and *statement*) (Héon *et al.*, 2010).

Table 1. MOT knowledge categories and their shapes.

Knowledge categories / Knowledge types	Abstract knowledge		Factual knowledge	
Conceptual *The What*	Concept		Example	
Procedural *The How*	Procedure		Trace	
Strategic *The Why, When, Who*	Principle		Statement	

[2]LICEF: Laboratory of Cognitive Informatics and Training Environments of the Quebec Tele University.

- *Concept*: This abstract knowledge represents a class of objects. It is the abstraction of a concrete object called *Example* (e.g. the concept *client* is the abstraction of *James*).
- *Procedure*: It describes the sets of operations that apply on objects. The instantiation of a procedure gives a concrete fact called *Trace* (e.g. the procedure *Buy products* is the abstraction of *Buy books*).
- *Principle*: It helps establishing cause–effect links between objects, determines the conditions that may apply to the implementation of actions and represents the agents that act on something. The instantiation of a principle is a concrete fact called *Statement* (e.g. the principle *Income of client* > 300 is the abstraction of *Income of the client James* $= 400 > 300$).

In MOT, knowledge relates by typed links (cf. Table 2) (Héon *et al.*, 2009, 2010). Each type of link has its own semantics that respects a set of integrity rules (described in Paquette, 2002) (e.g. if the origin and destination of a link are of type *Concept*, then this link is of type *C* or *S*).

3.2.2. *The MOT meta-model*

As we have selected the MOT for knowledge modelling, we need to define the MOT meta-model (MM); to do so we refer to Paquette (1996, 2002) and we give in Fig. 2

Table 2. Semantics of typed relationships in MOT.

Link type	Acronym	Role
Instantiation	I	Connects an abstract knowledge to a fact (i.e. a *link I* from the concept *Renault cars* to the example *Jean's car*)
Composition/multiple composition	C/C*	Connects knowledge to one of its component parts (i.e. a *link C* from the concept *car* to the concept *motor*)
Specialisation	S	Connects two abstract knowledge of the same type, one of which is a-kind-of the other (i.e., = a *link S* from the concept *Renault* to the concept *Car*)
Precedence	P	Associates two procedural knowledge or strategic knowledge, the first must be completed before the second begins (i.e. a *link P* from the procedure *Make the plan* to the procedure *Write Text*).
Input/product	I/P	Associates procedural knowledge and conceptual knowledge to represent the input or the product of procedural knowledge (i.e. a *link I/P* from the concept *Plan*, representing the input, to the procedure *Write Text*).
Regulation	R	Used from strategic knowledge to another knowledge to clarify a constraint to satisfy conceptual knowledge and control the execution of procedural knowledge or the selection of other strategic knowledge (i.e. a *link R* from the principle *Figure with three angles* to the concept *Triangle*)
Application	A	Based on a combination of two areas: the first plays the role of meta-knowledge for the second. Thus, this link allows the association of a fact to knowledge (i.e. a *link A* from the example *species*, which is the fact of the concept *taxonomic ranks*, to the concept *dog*)

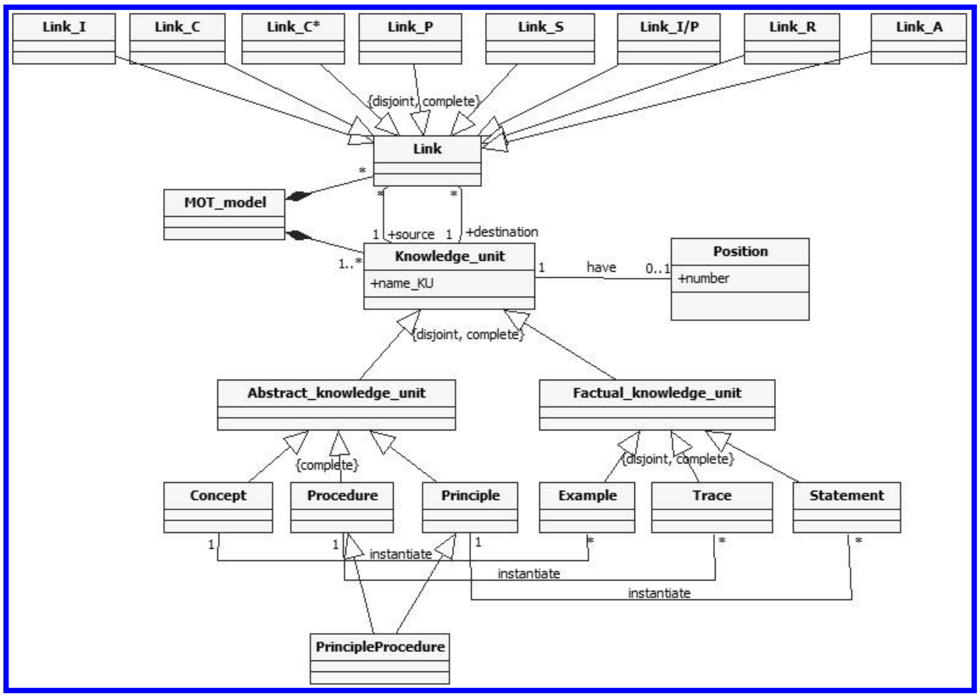


Fig. 2. Extended MOT meta-model.

the MOT MM as a UML (Unified Modeling Language) class diagram that shows two main classes:

- *Knowledge_Unit* (KU) class: This class has one attribute called *Name_KU* and two subclasses: *Abstract_knowledge_unit* (AKU) and *Factual_knowledge_unit* (FKU). For each AKU (i.e. *Concept*, *Procedure*, *Principle*) it may correspond n ($n \geq 1$) FKUs (i.e. *Example*, *Trace*, *Statement*).
- *Link* class: A link between two KUs is of type {I, C, C*, P, S, I/P, R, A}.

We have extended this MM with two new classes (i.e. *Position* and *PrincipleProcedure*). With this extension, the MOT MM becomes able to describe (i.e. model) explicit knowledge represented as decision tree or association rules. The *Position* class is useful to keep track of the position of each node in the decision tree (cf. Sec. 5). Whereas the *PrincipleProcedure* class is useful in defining transformation rules for association rules where a KU could be a *Principle* and a *Procedure* simultaneously (i.e. a KU that will play both the double role of an antecedent and a consequent of type action within an association rules base) (cf. Sec. 6).

In this extended MM, we have considered transformations for three knowledge models: decision trees, association rules and clustering. Note that to transform further knowledge models into MOT, we may need to extend its MM with new classes or attributes, if necessary.

As our objective is to define transformation rules that represent the passage channel between a source knowledge model and our target model (MOT), we give an overview of model transformation techniques.

4. Models Transformation

Model-driven engineering (MDE) provides software tools to create and transform models. In MDE, all or part of a computer application is generated from models. Interest in MDE was strongly amplified when the OMG (Object Management Group) (OMG, 2015) developed its initiative MDA (Model-Driven Architecture) (OMG, 2003). MDA is a software development approach based on the massive use of models at different abstraction levels and on transformations between models. In fact, a *transformation* is a set of rules that establish structural correspondences from a source model towards a target model. These correspondences base on the MMs of the source and target models.

4.1. *Model transformation languages*

Let us recall that our goal is to transform different explicit knowledge models into MOT that we consider as a pivot knowledge model. There are several model transformation languages; the most adopted as standard for MDA are QVT (Query-View-Transformation) (OMG, 2011) and ATL (Atlas Transformation Language) (ATLAS, 2006).

QVT standard is OMG-recommended; it enables expressing queries, views and transformations on models. The QVT architecture consists of three parts called *Relations*, *Core* and *Operational Mappings*. The QVT language is a concrete linguistic object for expressing model transformations with a hybrid declarative/imperative nature (Giandini *et al.*, 2009). The declarative part helps writing a simple set of model transformation rules, whereas the imperative part is close to conventional programming languages.

ATL is a model transformation language based on OCL (Object Constraint Language) (OMG, 2012). ATL accepts declarative and imperative constructions; it is a plugin of the Eclipse platform, available on its EMF (Eclipse Modelling Framework) tool (Koegel and Helming, 2016). It is currently widely used since it is judged more efficient than QVT (Jouault *et al.*, 2008; Erata *et al.*, 2015), in that it: (i) runs faster, (ii) facilitates expressing the set of correspondences between models using the condition clause in the transformation rules and (iii) compiles and runs on a virtual machine. Given these benefits, we opted for ATL to formalise transformation rules.

An ATL transformation (Jouault *et al.*, 2008) (cf. Fig. 3) defines the way of generating a model *Mb* (conform to its MM *MMb*) from a model *Ma* (conform to its MM *MMa*). These models are in XMI format (XML Metadata Interchange) (OMG, 2014). Such a transformation is in its turn defined as a transformation model

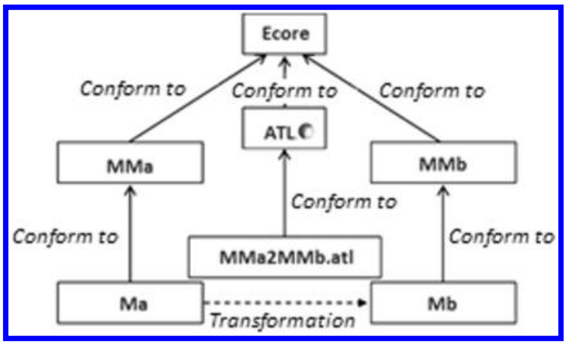


Fig. 3. Overview of ATL transformations.

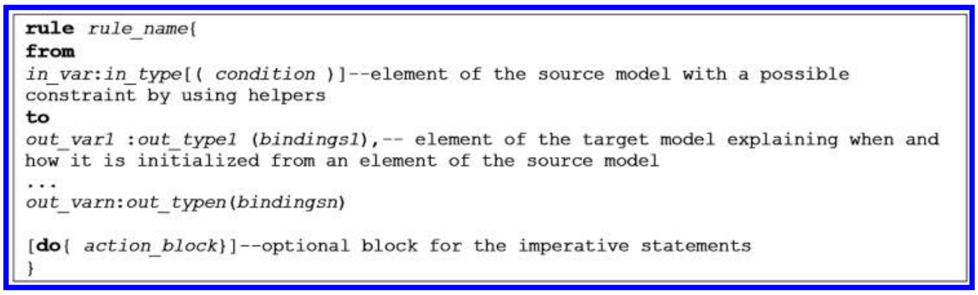
(*MMa2MMb.atl*). The latter model must conform to an MM that defines the model transformation semantics. All these MMs must conform to the considered meta-meta-model (*Ecore*); they describe graphically as *Ecore* diagrams (Fig. 6 is an example of *Ecore* diagram for the decision tree MM).

An ATL transformation module is mainly composed of a set of *Rules* and *Helpers* (Jouault *et al.*, 2008):

- *Rules*: correspond to the transformation of a set of source elements into a set of target elements. For this, ATL offers three types of declarative rules (*Matched rule* (i.e. the standard rule) (cf. Fig. 4), *lazy rule*, *unique lazy rule*) and one type of imperative rules (*Called rule*).
- *Helpers*: are ATL functions from the OCL standard. They are either of type *operation* calculated at each call or of type *attribute* calculated only once.

4.2. *Overview of our transformation approach*

In order to transform and then represent our three knowledge models (decision tree, association rules and clustering) into MOT, we perform two processing steps: the first step bases on ATL transformation rules specific for each source knowledge model whereas the second step bases on transformation rules formalised in Java language. To do so, we designed four MM: one MM for each of the three knowledge



```
rule rule_name{
from
in_var:in_type[( condition )]--element of the source model with a possible
constraint by using helpers
to
out_var1 :out_type1 (bindings1),-- element of the target model explaining when and
how it is initialized from an element of the source model
...
out_varn:out_typen(bindingsn)

[do{ action_block}]--optional block for the imperative statements
}
```

Fig. 4. Syntax for ATL declarative *Matched rule*.

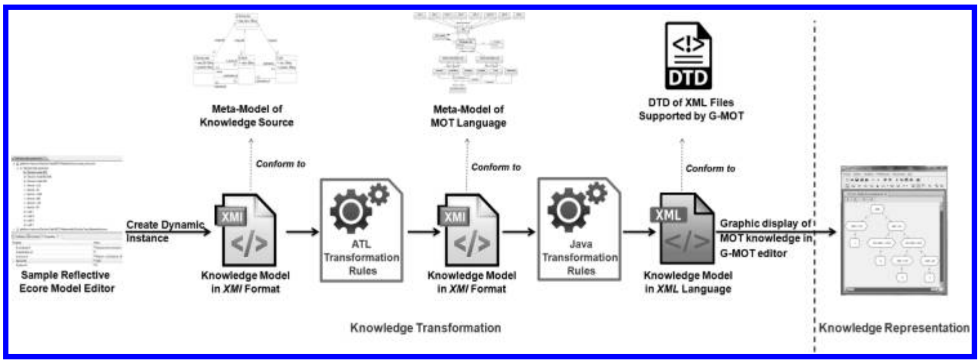

Meta-Model of Knowledge Source
Meta-Model of MOT Language
DTD
DTD of XML Files Supported by G-MOT
Conform to
Conform to
Conform to
Sample Reflective Ecore Model Editor
Create Dynamic Instance
XMI
Knowledge Model in XMI Format
ATL Transformation Rules
XMI
Knowledge Model in XMI Format
Java Transformation Rules
XML
Knowledge Model in XML Language
Graphic display of MOT knowledge in G-MOT editor
Knowledge Transformation
Knowledge Representation

Fig. 5. Transformation approach towards MOT.

models (decision trees, association rules and clustering) and one for the MOT MM. Then we defined these MM as *Ecore* diagrams. Figure 5 shows an overview of our transformation approach.

In our approach, a source knowledge model is instantiated with the Sample Reflective *Ecore Model Editor*[3] that generates the model in XMI format (corresponding to *Ma* model in Fig. 3). Then, we transform this latter model into MOT according to XMI format (*Mb* model in Fig. 3) by applying our transformation rules formalised in ATL and implemented under Eclipse environment. Both source and target models conform to the MM of the knowledge source and to the MM of MOT, respectively.

After that, we transform the obtained MOT-XMI file into XML that represents an MOT model conform to the DTD of XML files supported by the G-MOT knowledge-modelling editor.[4] This transformation is necessary to display graphically the obtained MOT, using G-MOT editor.

5. Transformation of Decision Trees into MOT

In this section, we are interested in the decision tree as a knowledge model and its transformation into MOT.

5.1. *Decision tree*

Figure 6 depicts the MM of a decision tree (DT) which has three components: *decision nodes*, *branches* and *leaves*.

- *Decision node*: is the place where we make a choice; it has a name *name_DN* and is identified by its position *positionN* in the tree. It is the source of n ($n \geq 1$) branches and is a destination of zero or one branch.

[3]Reflective *Ecore* Model Diagram Editor is a graphical modelling framework (GMF)-based Eclipse plugin which provides a graphical editor for any EMF model file, using only the meta-model such as *.ecore* file.
[4]The editor is distributed by the LICEF Research Center and it is the most recent contribution comparing to editors MOT2.3 and MOTplus. It is available on http://poseidon.licef.ca/gmot/.

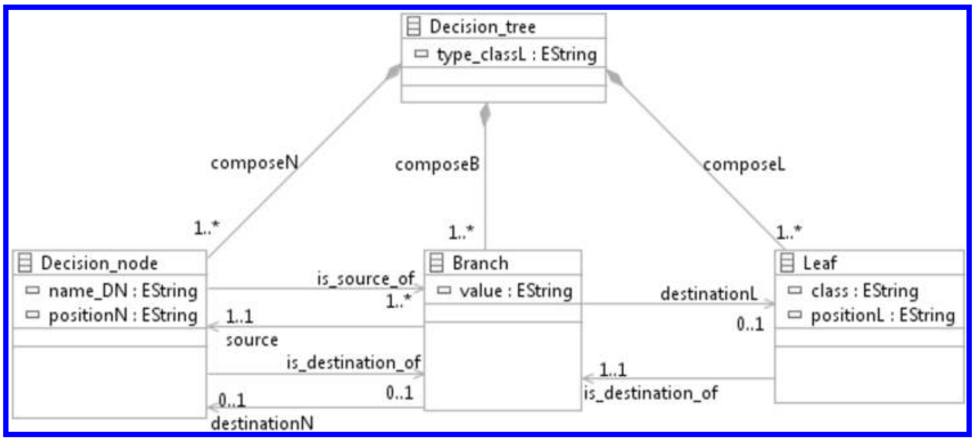


Fig. 6. Decision tree MM (as *Ecore* diagram).

- *Branch*: represents a possible decision; its label is a *value.* It associates a source *decision node* to a *decision node* or a *leaf.*
- *Leaf*: is a terminal node representing a final decision. Its label is the *class* of the object to classify; it is identified by its position *positionL* in the tree. Note that semantically the classes of all leaves of a same tree are either of type *Conclusion* (e.g. risky clients) or *Action* (e.g. buy a product).

Figure 7 shows an example of decision tree; it helps to classify the clients of a bank and infer whether they are risky clients or not.[5] This classification bases on the values of two attributes, namely, AGE and INCOME of clients.

For design purposes, we consider that the position of a tree node (decision node or leaf node) is an identifier for that node. In order to guarantee this uniqueness property, we determine the position according to the following principle: assign 0 (zero) to the root node (AGE in Fig. 7), then the position of each child is the

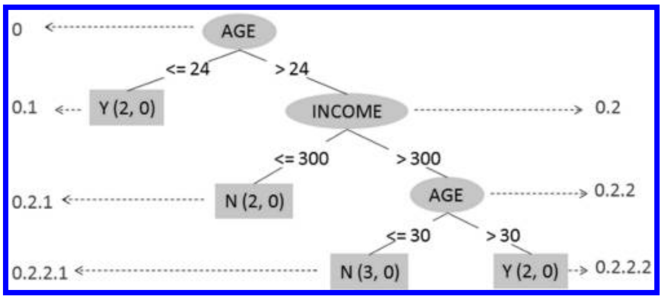


Fig. 7. Decision tree of risky customers with nodes identifiers.

[5] http://jaub.developpez.com/tutoriels/weka/weka/.

concatenation of its parent number with the child rank (from left to right in its level), we separate concatenated numbers with dot ("."). Thus, for instance, in Fig. 7, the leaf "Y" and the node "INCOME" are numbered 0.1 and 0.2, respectively.

5.2. *Decision tree transformation rules*

To transform a decision tree into MOT, we define four transformation rules to match the decision tree MM elements with elements of the MOT MM (Ayadi *et al.*, 2015). The first three rules generate a set of AKUs (abstract knowledge units), whereas the last one links them. Note that in these rules, *root* refers to the top-level node (*position* $N = 0$) of a decision tree.

Rule 1. R2Pr

The root R of a decision tree transforms into an abstract knowledge unit of type *Principle Pr1*. The name and position of $Pr1$ are those of R.

Rule 2. B2Pr

The value of a branch $Br1$ from node $n1$ to a consecutive node $n2$ transforms into an abstract knowledge unit of type *Principle* $Pr2$. The name of $Pr2$ is the concatenation of the name of $n1$ with the value of $Br1$. The position of $Pr2$ is that of $n2$.

Rule 3. L2AKU

A leaf $F1$ transforms into an abstract knowledge unit $U1$. The name of $U1$ is the *class* of $F1$. The position of $U1$ is *position* L of $F1$ concatenated with the literal constant "**.1**".

In rule L2AKU, the type of an abstract knowledge unit depends on the semantics of the *class* of $F1$ (i.e. *Conclusion* or *Action*). According to the *class* of $F1$, the following two subrules determine the type of the abstract knowledge unit.

Rule 3.1. L2Pr
If the *class* of $F1$ is *Conclusion* then $F1$ transforms into abstract knowledge unit of type *Principle*.

Rule 3.2. L2Proc
If the *class* of $F1$ is *Action* then $F1$ transforms into abstract knowledge unit of type *Procedure*.

To link all AKUs obtained by rules 1–3, we rely on their position as stated by rule 4.

Rule 4. AKU2AKU

Two abstract knowledge units $U1$ and $U2$ having, respectively, their position x and $x.i$ will be connected by a link directed from $U1$ to $U2$.
As an example, if x is **0.2** (position of U1) and $x.i$ is **0.2**.1 then connect $U1$ to $U2$.

Once the link is established from $U1$ to $U2$, we determine the link type using the following two subrules:

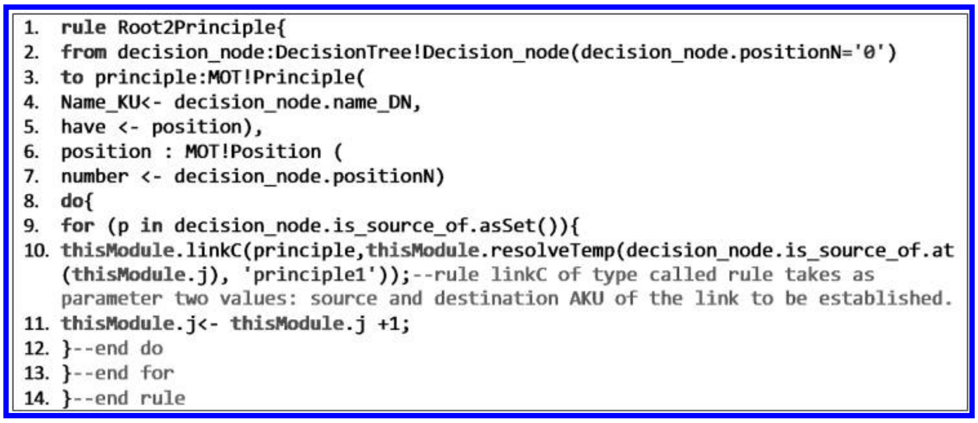
Rule 4.1. Pr2Pr
If $U1$ is a principle $Pr1$ obtained with $R2Pr$ and $U2$ is a principle $Pr2$ issued from $B2Pr$ then the link from $U1$ to $U2$ is of type C.

Rule 4.2. Pr2AKU
If $U1$ is a principle $Pr2$ obtained with $B2Pr$, and $U2$ is an AKU issued from $B2Pr$ or $L2AKU$ then the link from $U1$ to $U2$ is of type P.

We formalise these four transformation rules in ATL.

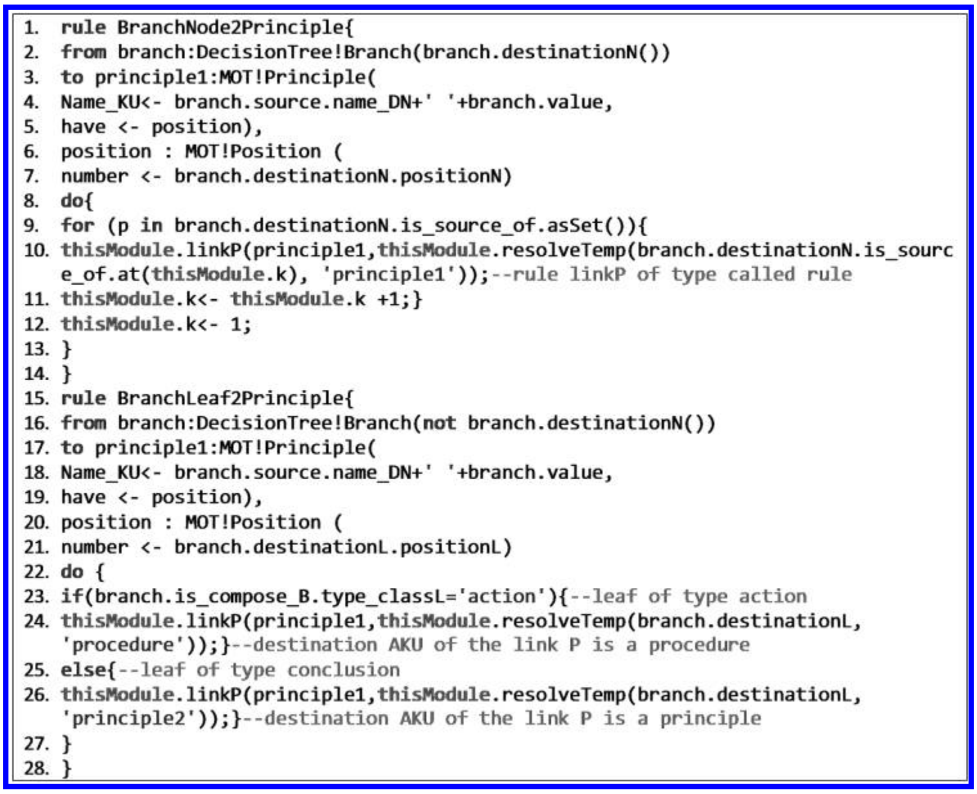
The ATL rule *Root2Principle* (cf. Fig. 8, line 1) formalises the two transformation rules R2Pr (lines 2–7) and *Pr2Pr* (line 10).

Since a branch associates a decision node to another decision node or a leaf, then to formalise rule *B2Pr* (cf. Fig. 9) in ATL we define two subrules: (i) *BranchNode2Principle* (lines 1–14) when the destination of the branch is a decision node and (ii) *BranchLeaf2Principle* (lines 15–28) when the destination of the branch is a leaf.



```
1.  rule Root2Principle{
2.  from decision_node:DecisionTree!Decision_node(decision_node.positionN='0')
3.  to principle:MOT!Principle(
4.  Name_KU<- decision_node.name_DN,
5.  have <- position),
6.  position : MOT!Position (
7.  number <- decision_node.positionN)
8.  do{
9.  for (p in decision_node.is_source_of.asSet()){
10. thisModule.linkC(principle,thisModule.resolveTemp(decision_node.is_source_of.at
    (thisModule.j), 'principle1'));--rule linkC of type called rule takes as
    parameter two values: source and destination AKU of the link to be established.
11. thisModule.j<- thisModule.j +1;
12. }--end do
13. }--end for
14. }--end rule
```

Fig. 8. ATL code of rules *R2Pr* and *Pr2Pr*.



```
1.  rule BranchNode2Principle{
2.  from branch:DecisionTree!Branch(branch.destinationN())
3.  to principle1:MOT!Principle(
4.  Name_KU<- branch.source.name_DN+' '+branch.value,
5.  have <- position),
6.  position : MOT!Position (
7.  number <- branch.destinationN.positionN)
8.  do{
9.  for (p in branch.destinationN.is_source_of.asSet()){
10. thisModule.linkP(principle1,thisModule.resolveTemp(branch.destinationN.is_sourc
    e_of.at(thisModule.k), 'principle1'));--rule linkP of type called rule
11. thisModule.k<- thisModule.k +1;}
12. thisModule.k<- 1;
13. }
14. }
15. rule BranchLeaf2Principle{
16. from branch:DecisionTree!Branch(not branch.destinationN())
17. to principle1:MOT!Principle(
18. Name_KU<- branch.source.name_DN+' '+branch.value,
19. have <- position),
20. position : MOT!Position (
21. number <- branch.destinationL.positionL)
22. do {
23. if(branch.is_compose_B.type_classL='action'){--leaf of type action
24. thisModule.linkP(principle1,thisModule.resolveTemp(branch.destinationL,
    'procedure'));}--destination AKU of the link P is a procedure
25. else{--leaf of type conclusion
26. thisModule.linkP(principle1,thisModule.resolveTemp(branch.destinationL,
    'principle2'));}--destination AKU of the link P is a principle
27. }
28. }
```

Fig. 9. Rules *B2Pr* and *Pr2AKU* formalised in ATL.

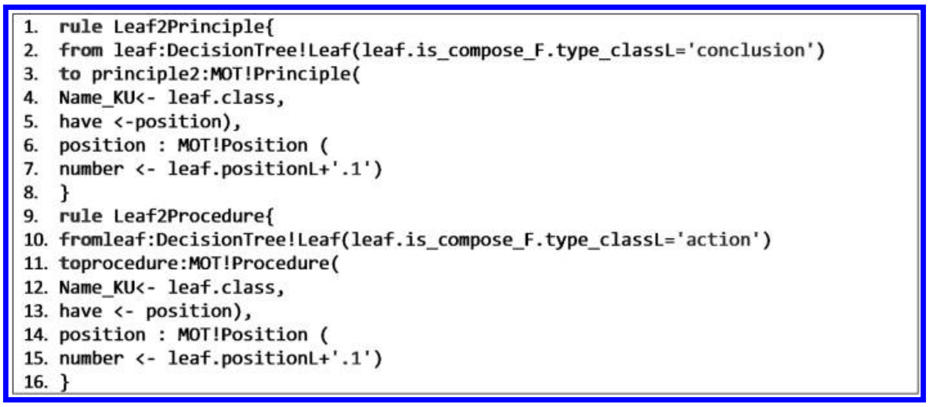
Furthermore, to determine whether the destination of a branch is a decision node or a leaf, we define the Boolean helper *destinationN*() that returns TRUE if the destination is a decision node and FALSE otherwise. We invoke this helper as an OCL constraint in line 2 of the *BranchNode*2*Principle* rule and in line 16 of the *BranchLeaf*2*Principle* rule (cf. Fig. 9).

Furthermore, we invoke the rule *linkP* of type *Called rule* in lines 10, 24 and 26 of Fig. 9 in order to link *Principle* to abstract knowledge unit according to the rule *Pr2AKU*.

In Fig. 10, we formalise rules *L*2*Pr* and *L*2*Proc* in ATL by rules *Leaf*2*Principle* (lines 1–8) and *Leaf*2*Procedure* (lines 9–16), respectively.

Remember that the AKU type (i.e. *Principle* or *Procedure*) depends on the semantics of the *class* of the leaf (i.e. *Conclusion* or *Action*). Lines 2 and 10 in Fig. 10 determine this type.

Applying these five ATL transformation rules on the decision tree of *Risky customers* (cf. Fig. 7) generated in XMI format by the Sample Reflective Ecore Model Editor (cf. Fig. A.1), we obtain the corresponding MOT target model in XMI format (cf. Fig. A.2). We transform this latter into XML format for display (cf. Fig. 5).



```
rule Leaf2Principle{
from leaf:DecisionTree!Leaf(leaf.is_compose_F.type_classL='conclusion')
to principle2:MOT!Principle(
Name_KU<- leaf.class,
have <-position),
position : MOT!Position (
number <- leaf.positionL+'.1')
}
rule Leaf2Procedure{
fromleaf:DecisionTree!Leaf(leaf.is_compose_F.type_classL='action')
toprocedure:MOT!Procedure(
Name_KU<- leaf.class,
have <- position),
position : MOT!Position (
number <- leaf.positionL+'.1')
}
```

Fig. 10. Rule *L2AKU* formalised in ATL.

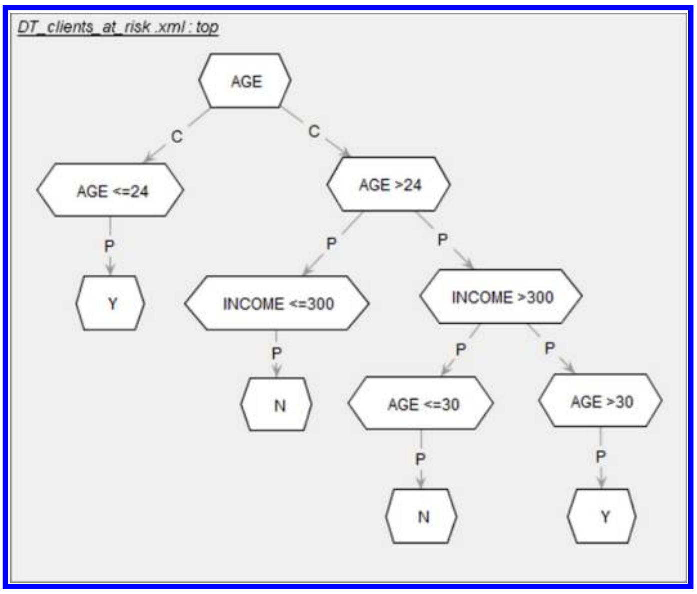


Fig. 11. Decision tree of risky customers (cf. Fig. 7) transformed into MOT and displayed using G-MOT editor.

Figure 11 graphically illustrates the obtained decision tree displayed with G-MOT editor.

In the next section, we deal with association rules, as a source knowledge model, and their transformation into MOT.

6. Transformation of Association Rules into MOT

This section presents the Association Rules (AR) data mining technique and defines the MM of AR along with our eight transformation rules to generate MOT model from an AR Base.

6.1. *Association rules*

The data mining technique that discovers interesting relations between variables in large datasets produces knowledge expressed as AR; it is known as AR technique. The set of obtained rules constitutes an AR Base (ARB).

An AR has the form "*If Condition then Action*" where *Condition* and *Action* are called *Antecedent* and *Consequent*, respectively; each of which is, in turn, composed of *item*(*s*). An example of AR is for the market basket analysis: "*If onions and potatoes then hamburger*" would indicate that if a customer buys onions and potatoes together, he/she is likely to buy also hamburger.

Figure 12 shows the MM of an AR base.

This MM shows the four components of an AR Base called *RBname*. This base is composed of *nbAR* ($nbAR \geq 1$) ARs; each rule has an identifier that is a sequential number *num*. In an ARB, each antecedent may imply more than one consequent. Moreover, each item has a name *name_I* and a type *type_I* (i.e. *Conclusion* or *Action*). One item can participate in the *Antecedent* and in the *Consequent* of two different rules.

Table 3 depicts an AR base conform to the MM of Fig. 12; this base is extracted from the dataset of a company to help its endeavours to more effectively target its marketing campaign towards core customers.[6] Five attributes describe the dataset: *Age*, *Education*, *Income*, *Marital Status* and *Purchase*.

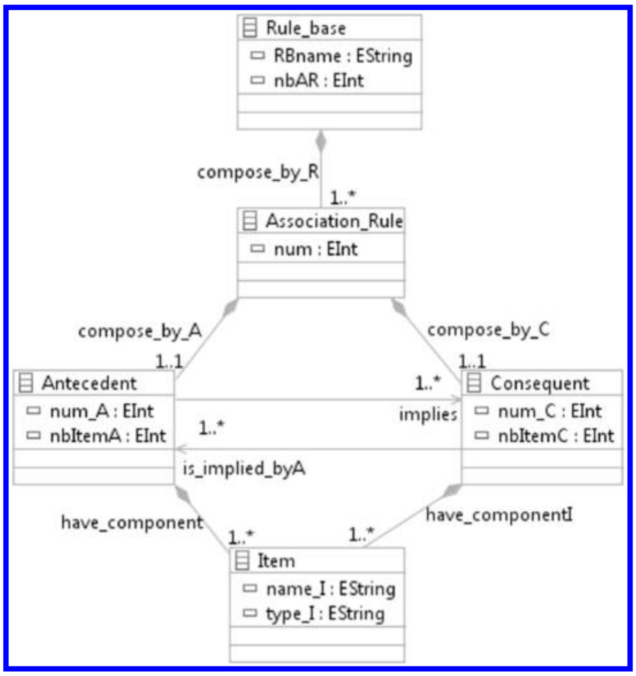


Fig. 12. *Ecore* meta-model for the AR base.

[6]http://www.onlamp.com/pub/a/python/2006/02/09/ai_decision_trees.html.

Table 3. ARB of customers.

AR#	Antecedent	Consequent
1	“Marital Status=single” and “Education=master’s”	“Purchase=will buy”
2	“Age=>55”	“Purchase=will buy”
3	“Purchase=will buy” and “Age=36–55”	“Marital Status=single”
4	“Marital Status=single” and “Age=36–55”	“Purchase=will buy”
5	“Education=master’s”	“Purchase=will buy”
6	“Marital Status=single” and “Income=low”	“Purchase=will buy”
7	“Purchase=will buy” and “Education=master’s”	“Marital Status=single”

6.2. *Association-rules-base transformation rules*

Here, we define transformation rules to link elements (classes or attributes) from the ARB MM with elements of the MOT MM (Ayadi *et al.*, 2017a).

Note that in the MOT model, *Antecedents* are always of a single type of knowledge unit (KU). However, *Consequents* could be of one among three types of KUs. The choice of the appropriate type depends first on the semantics of items participating in *Consequents*, which are of type *Action* or *Conclusion*, and secondly on whether these items are antecedents in other rules or not.

Thus, before defining transformation rules, we need to classify all items of an ARB $B1$ into the following three subsets:

- S_{Ant}: Subset of items from $B1$ that participate only in *Antecedents*.
- S_{Cons}: Subset of items from $B1$ that participate only in *Consequents*.
- S_{AC}: Subset of items from $B1$ that participate in *Antecedents* and *Consequents* in different rules.

This classification helps to identify the type of knowledge unit that will represent the item. It performs automatically by browsing all ARs. However, the KW Administrator should intervene to decide the type of the *Consequent* item (i.e. *Action* or *Conclusion*) because this decision is semantics-dependent.

Based on these three subsets and the types of *Consequents*, we define six transformation rules to generate abstract knowledge units from an ARB.

Rule 1. $\mathbf{I}_{S_{\text{Ant}}}$2Pr

An item I belonging to S_{Ant} transforms into an abstract knowledge unit of type *Principle* $Pr1$. The name of $Pr1$ is that of I.

Rule 2. $\mathbf{I}_{S_{\text{Cons}}}$2Pr

An item I of type *Conclusion* belonging to S_{Cons} transforms into an abstract knowledge unit of type *Principle* $Pr2$. The name of $Pr2$ is that of I.

Rule 3. I$_{S_{\text{Cons}}}$2Proc

An item I of type *Action* belonging to the subset S_{Cons} transforms into an abstract knowledge unit of type *Procedure Proc*1. The name of *Proc*1 is that of I.

Rule 4. I$_{S_{\text{AC}}}$2PrProc

An item I of type *Action* belonging to the subset S_{AC} transforms into an abstract knowledge unit of type *PrincipleProcedure PrProc*. The name of *PrProc* is that of I.

In this rule, we use the new type abstract knowledge unit hybrid *Principle Procedure* introduced in the MOT MM (cf. Fig. 2). This AKU is useful to represent a knowledge unit that plays the double role of *Principle* as an *Antecedent* (cf., rule I$_{S_{\text{Ant}}}$2Pr) and *Procedure* as a *Consequent* of type *Action* (cf. rule I$_{S_{\text{Cons}}}$2Proc).

Rule 5. I$_{S_{\text{AC}}}$2Pr

An item I of type *Conclusion* belonging to the subset S_{AC} becomes an AKU of type *Principle Pr*3. The name of *Pr*3 is that of I.

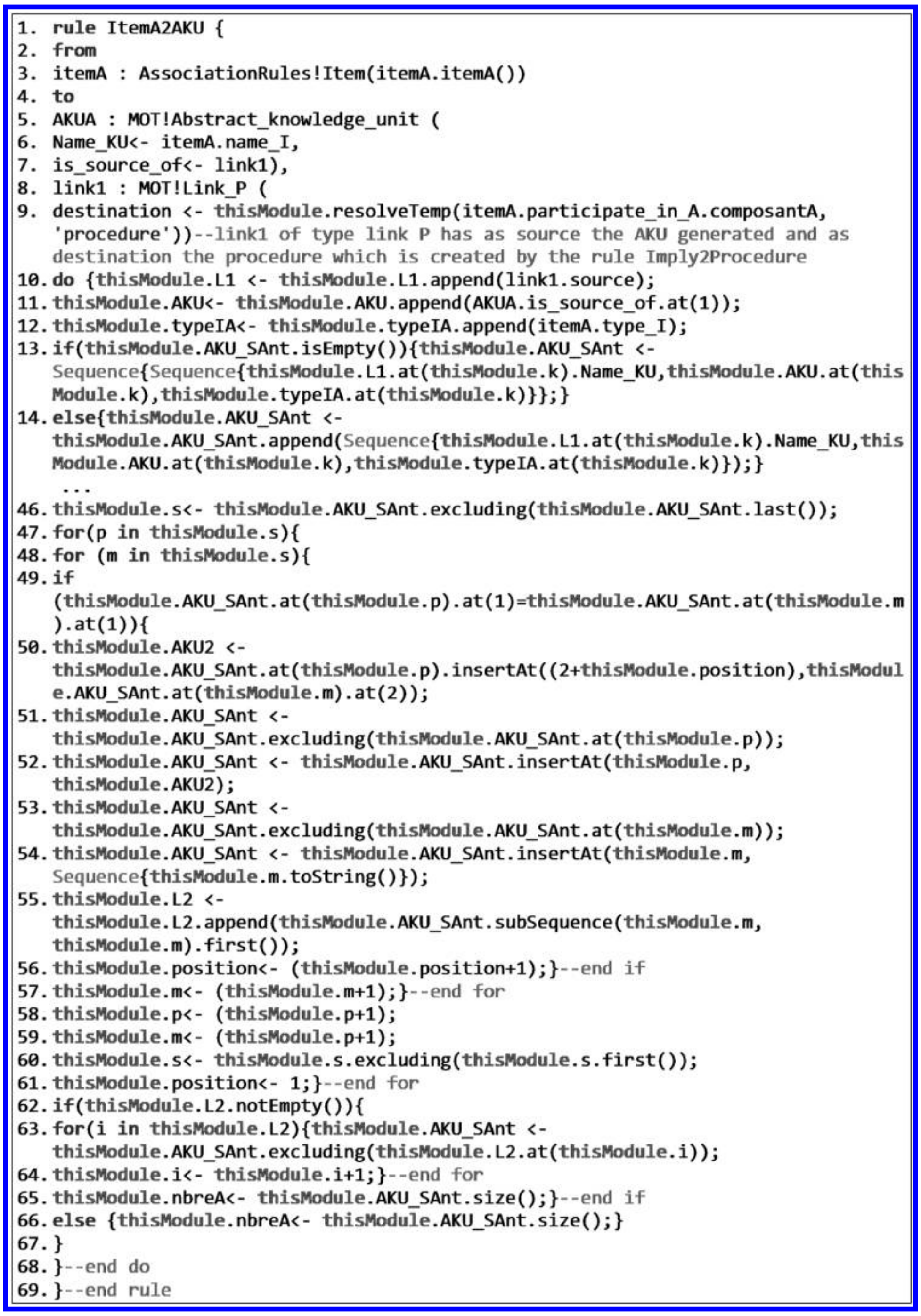
Rule 6. IR2Proc

Any *implies* association between an antecedent $A1$ and a consequent $C1$ in an association rule $R1$ ($R1$ belongs to ARB $B1$) transforms into an AKU of type *Procedure Proc*2. The name of *Proc2* is the string "*IR*" concatenated with the rule number *num* and the base name *RBname* (e.g. if the rule number (num) is 1 within the ARB $B1$ then we obtain the name IR1_B1 for the association *implies*).

The following two rules enable us to link AKU obtained by the above rules. The first deals with *Antecedents*, whereas the second is for *Consequents*.

Rule 7. AKU2Proc

An AKU *ak*1 issued from an association rule *Ri* (by applying I$_{S_{\text{Ant}}}$2Pr or I$_{S_{\text{AC}}}$2PrProc or I$_{S_{\text{AC}}}$2Pr) is linked to a procedure *Proc*2 issued from the same rule *Ri* (by applying IR2Proc) by a link of type *link P*. *ak*1 and *Proc*2 are the source and destination of *link P*, respectively.



```
rule ItemA2AKU {
from
itemA : AssociationRules!Item(itemA.itemA())
to
AKUA : MOT!Abstract_knowledge_unit (
Name_KU<- itemA.name_I,
is_source_of<- link1),
link1 : MOT!Link_P (
destination <- thisModule.resolveTemp(itemA.participate_in_A.composantA, 'procedure'))--link1 of type link P has as source the AKU generated and as destination the procedure which is created by the rule Imply2Procedure
do {thisModule.L1 <- thisModule.L1.append(link1.source);
thisModule.AKU<- thisModule.AKU.append(AKUA.is_source_of.at(1));
thisModule.typeIA<- thisModule.typeIA.append(itemA.type_I);
if(thisModule.AKU_SAnt.isEmpty()){thisModule.AKU_SAnt <- Sequence{Sequence{thisModule.L1.at(thisModule.k).Name_KU,thisModule.AKU.at(thisModule.k),thisModule.typeIA.at(thisModule.k)}};}
else{thisModule.AKU_SAnt <- thisModule.AKU_SAnt.append(Sequence{thisModule.L1.at(thisModule.k).Name_KU,thisModule.AKU.at(thisModule.k),thisModule.typeIA.at(thisModule.k)});}
 ...
thisModule.s<- thisModule.AKU_SAnt.excluding(thisModule.AKU_SAnt.last());
for(p in thisModule.s){
for (m in thisModule.s){
if (thisModule.AKU_SAnt.at(thisModule.p).at(1)=thisModule.AKU_SAnt.at(thisModule.m).at(1)){
thisModule.AKU2 <- thisModule.AKU_SAnt.at(thisModule.p).insertAt((2+thisModule.position),thisModule.AKU_SAnt.at(thisModule.m).at(2));
thisModule.AKU_SAnt <- thisModule.AKU_SAnt.excluding(thisModule.AKU_SAnt.at(thisModule.p));
thisModule.AKU_SAnt <- thisModule.AKU_SAnt.insertAt(thisModule.p, thisModule.AKU2);
thisModule.AKU_SAnt <- thisModule.AKU_SAnt.excluding(thisModule.AKU_SAnt.at(thisModule.m));
thisModule.AKU_SAnt <- thisModule.AKU_SAnt.insertAt(thisModule.m, Sequence{thisModule.m.toString()});
thisModule.L2 <- thisModule.L2.append(thisModule.AKU_SAnt.subSequence(thisModule.m, thisModule.m).first());
thisModule.position<- (thisModule.position+1);}--end if
thisModule.m<- (thisModule.m+1);}--end for
thisModule.p<- (thisModule.p+1);
thisModule.m<- (thisModule.p+1);
thisModule.s<- thisModule.s.excluding(thisModule.s.first());
thisModule.position<- 1;}--end for
if(thisModule.L2.notEmpty()){
for(i in thisModule.L2){thisModule.AKU_SAnt <- thisModule.AKU_SAnt.excluding(thisModule.L2.at(thisModule.i));
thisModule.i<- thisModule.i+1;}--end for
thisModule.nbreA<- thisModule.AKU_SAnt.size();}--end if
else {thisModule.nbreA<- thisModule.AKU_SAnt.size();}
}
}--end do
}--end rule
```

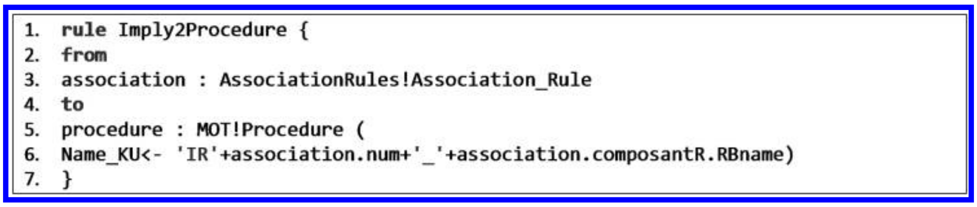
Fig. 13. Extract of the rule *ItemA*2AKU formalised in ATL.

Rule 8. Proc2AKU

An AKU *ak*2 issued from an association rule *Ri* (by applying $\mathrm{I}_{S_{\mathrm{Cons}}}$2Pr or $\mathrm{I}_{S_{\mathrm{Cons}}}$2Proc or $\mathrm{I}_{S_{\mathrm{AC}}}$2PrProc or $\mathrm{I}_{S_{\mathrm{AC}}}$2Pr) is linked to a procedure *Proc*2 issued from the same rule *Ri* (by applying IR2Proc) by a link of type *link P* directed from *Proc*2 to *ak*2.

Let us recall that the *link P* in the MOT model associates two procedural or strategic knowledge units in which the first must complete before the second starts (cf. Sec. 3.2.1).

We formalise these eight transformation rules in ATL. To do this, we define two helpers *itemA*() and *itemC*() of type *operation* to check, respectively, whether an item participates to an *Antecedent* or to a *Consequent*. These helpers are invoked, as OCL constraints, in lines 3 of rules *ItemA2AKU* (cf. Fig. 13) and *ItemC2AKU* (cf. Fig. B.1). In fact, the *ItemA2AKU* rule (respectively, *ItemC2AKU*) transforms an item participating in the *Antecedent* (respectively, the *Consequent*) into an AKU.



```
rule Imply2Procedure {
from
association : AssociationRules!Association_Rule
to
procedure : MOT!Procedure (
Name_KU<- 'IR'+association.num+'_'+association.composantR.RBname)
}
```

Fig. 14. Formalisation of rule IR2Proc in ATL.

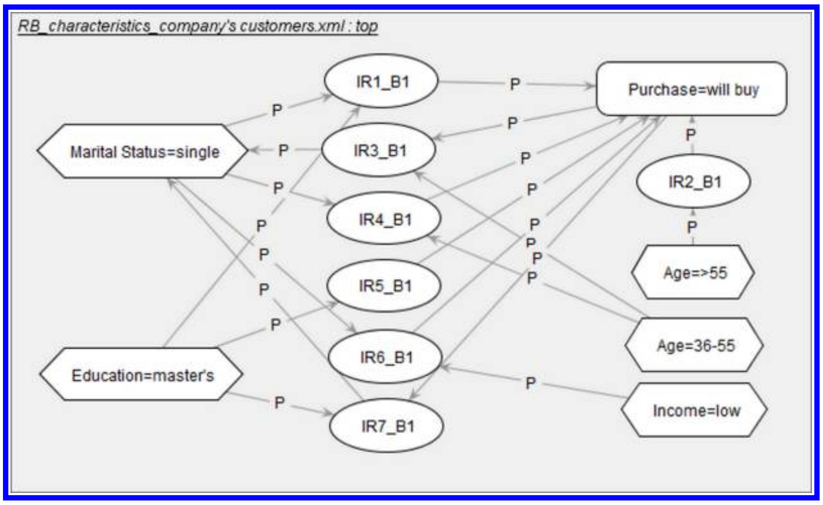


Fig. 15. MOT diagram for the ARB in Table 3, displayed using G-MOT editor.

The *ItemC2AKU* ATL transformation rule formalises the five rules $\mathrm{I}_{S_{\mathrm{Ant}}}$2Pr, $\mathrm{I}_{S_{\mathrm{Cons}}}$2Pr, $\mathrm{I}_{S_{\mathrm{Cons}}}$2Proc, $\mathrm{I}_{S_{\mathrm{AC}}}$2PrProc and $\mathrm{I}_{S_{\mathrm{AC}}}$2Pr (cf. Fig. B.1).

The linking rules AKU2Proc and Proc2AKU are defined, respectively, in lines 7–9 inside the two rules *ItemA2AKU* (cf. Fig. 13) and *ItemC2AKU* (cf. Fig. B.1).

Finally, we formalise rule *IR*2*Proc* through the rule *Imply*2*Procedure* (cf. Fig. 14).

Applying the three ATL transformation rules (in Figs. 13, 14 and B.1) on the ARB of Table 3, we obtain an *XMI* file that conforms to the MOT MM. Following the same transformation steps of our proposed approach (cf. Fig. 5), we obtain the diagram of the ARB in MOT displayed using the G-MOT editor (cf. Fig. 15). In Fig. 15, the new graphical symbol for knowledge (rounded rectangle) represents the new *AKU* of type *PrincipleProcedure* that extends the MOT MM (cf. Fig. 2).

In the next section, we continue our transformation process into MOT with the set of clusters as the third model of knowledge.

7. Transformation of Clusters into MOT

In this section, we complete our approach with transforming a third knowledge source, namely, clustering. We first introduce the clustering data mining technique that produces knowledge as clusters and then we define the MM for clusters. Finally, we define transformation rules to generate the corresponding MOT model.

7.1. *Clustering method*

Clustering is the process of organising objects into clusters of similar objects. A cluster is therefore a collection of objects, which are similar to each other and dissimilar to objects of other clusters. Clustering is also called unsupervised classification since the number of classes generated by this process is unknown a priori (Basciani *et al.*, 2016). There are various techniques of clustering including *k-means* (Doshi *et al.*, 2015) and *hierarchical clustering* algorithms. In the following, we propose the MM for clusters. The MM in Fig. 16 shows the main component *Cluster* labelled *name_C*.

Figure 17 exemplifies two clusters {*Customers_little old_high income_single* and *Customers_young_low income_married*} extracted from the *bank_classif_deployment*[7] dataset, using the *k*-means technique. This dataset describes bank customers by a set of criteria such as *age*, *family status* and *income*.

7.2. *Transformation rules of clusters into MOT*

In order to transform clusters into MOT, we define three transformation rules that connect elements of the clusters' MM to those of the MOT MM.

Rule 1. C2Cp

[7] http://eric.univ-lyon2.fr/~ricco/tanagra/fichiers/fr_Tanagra_KMeans_Deploiement.pdf.

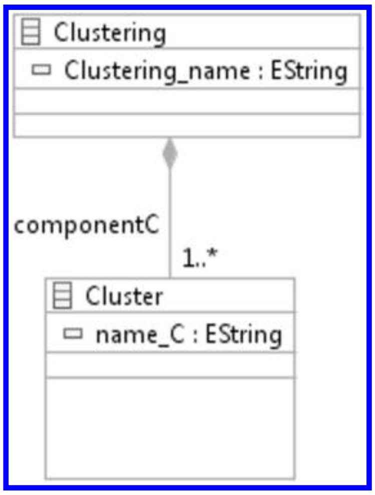


Fig. 16. Meta-model for clusters as an *Ecore* diagram.

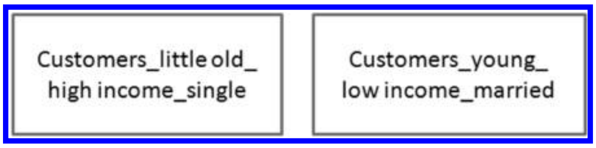

Customers_little old_ high income_single
Customers_young_ low income_married

Fig. 17. Clusters of the *bank_classif_deployment* dataset.

A cluster $c1$ transforms into an AKU of type concept $Cp1$. The name of $Cp1$ is that of $c1$.

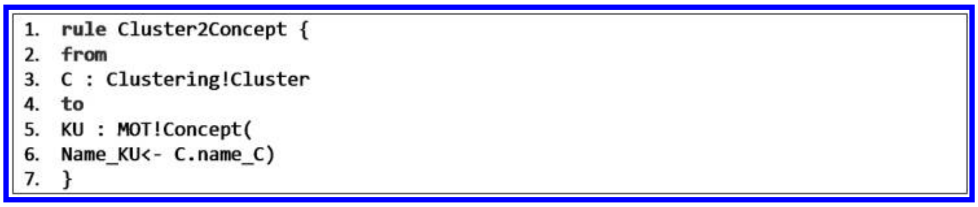
Rule 2. RC2Cp

The clustering class $cc1$ transforms into an AKU of type concept $Cp2$. The name of $Cp2$ is that of $cc1$.

Rule 3. $\mathbf{Cp_2 2Cp_1}$

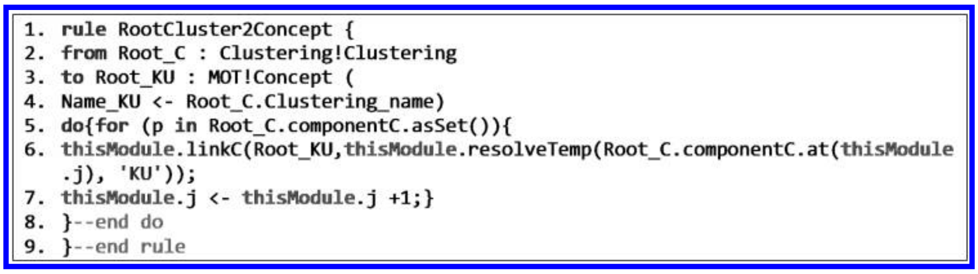
A concept $Cp1$ issued from rule $C2Cp$ is linked to a concept $Cp2$ issued from rule $RC2Cp$ by link of type *link C*. $Cp2$ and $Cp1$ are the source and destination of *link C*, respectively.

We formalised these three transformation rules in ATL. The ATL rule called *Cluster2Concept* (cf. Fig. 18) formalises $C2Cp$, and the ATL rule called



```
rule Cluster2Concept {
from
C : Clustering!Cluster
to
KU : MOT!Concept(
Name_KU<- C.name_C)
}
```

Fig. 18. Rule $C2Cp$ formalised in ATL.



```
1. rule RootCluster2Concept {
2. from Root_C : Clustering!Clustering
3. to Root_KU : MOT!Concept (
4. Name_KU <- Root_C.Clustering_name)
5. do{for (p in Root_C.componentC.asSet()){
6. thisModule.linkC(Root_KU,thisModule.resolveTemp(Root_C.componentC.at(thisModule
   .j), 'KU'));
7. thisModule.j <- thisModule.j +1;}
8. }--end do
9. }--end rule
```

Fig. 19. Rules $RC2Cp$ and $Cp_2 2Cp_1$ formalised in ATL.

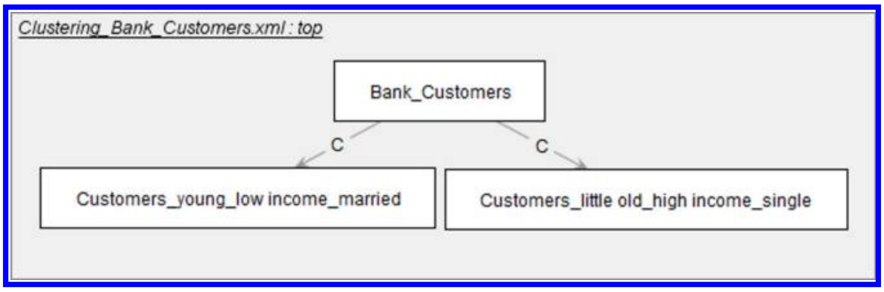

Clustering_Bank_Customers.xml : top
Bank_Customers
C
C
Customers_young_low income_married
Customers_little old_high income_single

Fig. 20. MOT diagram for clusters in Fig. 17.

RootCluster2Concept (cf. Fig. 19) formalises rules $RC2Cp$ (lines 2–4) and $Cp_2 2Cp_1$ (the called rule *linkC* in line 6).

After executing rules $C2Cp$, $RC2Cp$ and $Cp_2 2Cp_1$ on the set of clusters of our running example in Fig. 17, we obtain the MOT model in *XMI* format. Finally, the obtained clusters are plotted in MOT using G-MOT editor (cf. Fig. 20).

8. Next Step: From Transformation to Integration

So far, Secs. 5–7 are dedicated to transforming three heterogeneous knowledge models into MOT. Our immediate aim is to integrate MOT knowledge models into a global repository called *KW* that we consider the keystone for the construction of an Intelligent Decision Support System.

In order to demonstrate the usefulness of the present work and its next steps, we illustrate with an introductory example how to take decisions relying on such a

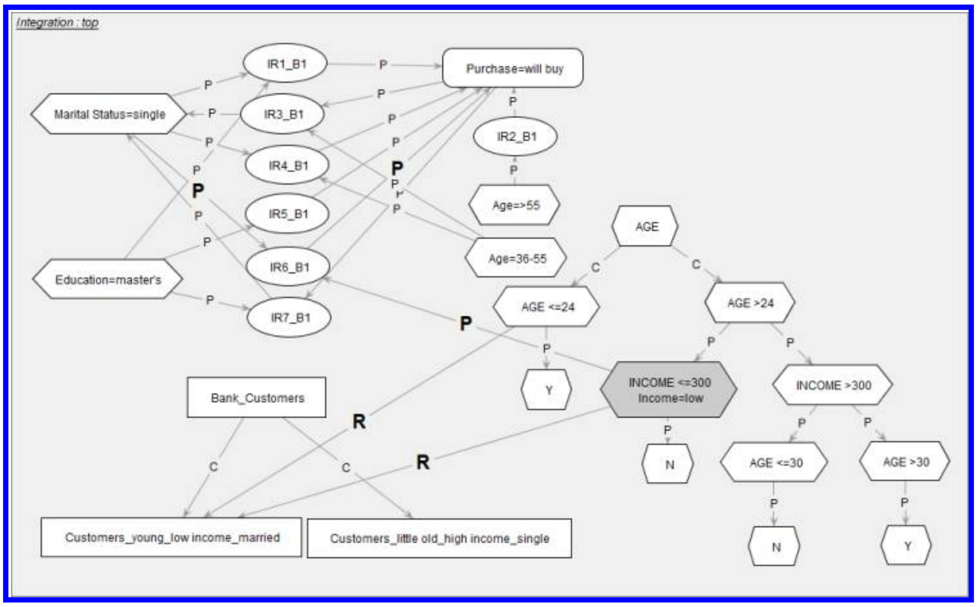


Fig. 21. MOT result of the integration of three MOT knowledge models.

KW. Figure 21 illustrates the knowledge model result from the integration of the three MOT knowledge models depicted in Figs. 11, 15 and 20.

In this example, the common AKU *Principle*, called "*INCOME* $\leq$ 300 *Income* $=$ *low*" (drawn in grey color in Fig. 21), links the MOT of the decision tree of risky customers (cf. Fig. 11) and the MOT of rules base of customers (cf. Fig. 15) using the *link P* in bold in Fig. 21. This new unified *Principle* is obtained after integrating the two principles "*INCOME* $\leq$ *300*" and "*Income* $=$ *low*" considered as synonyms using a taxonomy/ontology.

Note that before knowledge integration, we just have *isolated* fragment of knowledge indicating that customer with *Income* $\leq$ 300 is risky customer, no more (cf. Fig. 11). However, referring to this integrated model we have two benefits.

First, we can derive the new following knowledge: If the customer *income is* $\leq$ 300 *or is low* then this customer will buy products of the company if his marital status is single (cf., links of type *P* in bold that are the input/output of the rule *IR6_B1*). Obviously, this derived rule was neither available nor predictable before integration of the knowledge models, initially separated.

A second advantage is gained after integration; it is the presence of two new MOT links of type *link R* (in bold in Fig. 21). These links connect the *Principle* "*AGE* $<=$ *24*" and the *Principle* "*INCOME* $<=$ *300 Income* $=$ *low*" belonging to the two previous integrated MOT models with the *Concept* "*Customers_young_low income_married*" belonging to the MOT model of clusters of bank customers (cf. Fig. 20). The two new links indicate that the *Principles* raise as constraints to satisfy the *Concept* "*Customers_young_low income_married*". This means a new bank customer cannot belong to the group "*Customers_young_low income_married*" unless the condition *age* $\leq$ *24 and income* ≤ 300 is satisfied. These simple examples underline the importance of gathering knowledge in a KW for more consistent and well-founded decisions. Certainly, a case study is needed for better convincing; this is beyond the scope of this paper; nevertheless, it is among our ongoing work.

9. Conclusion and Perspectives

To be more competitive, organisations need an effective exploitation of knowledge to enhance their decision-making process for better-quality decisions. In order to construct an intelligent decision-support system, we have proposed the concept of KW along with a three-layer architecture, as a solution for integrating and storing knowledge of a given domain, and for sharing knowledge by decision-makers.

In this paper, we have focused on the first layer module called *Knowledge Normalisation Format* of this architecture, which aims to standardise and harmonise explicit knowledge initially formalised in heterogeneous formats (decision tree, AR, clustering). This module transforms knowledge expressed in different models into the MOT (Modeling with Object Types) semi-formal graphical language that we have elected as a pivot language considering its advantages in involving several stakeholders due to its diagrammatic aspects.

Furthermore, we have proposed a set of 15 rules to transform into MOT three knowledge models, namely, *decision trees*, *ARs* and *clustering*. To do so, we have designed four MMs, one for target MOT model and one MM for each of the three treated target knowledge models. Moreover, we have defined a set of matching rules based on structural correspondences between each source MM and the MOT target MM. We formalised these rules in ATL (Atlas Transformation Language) that is dedicated for model transformation.

Furthermore, we have implemented these rules under the Eclipse environment to generate MOT models automatically. In order to display graphically the obtained MOT model we have used the G-MOT editor, this task has required the generation of XML files from the XMI files that describe the MOT models.

Currently, we are developing an appropriate method for the integration of MOT knowledge models. It mainly consists in gathering knowledge, removing redundancy and inferring new knowledge for the decision-maker. A real-world case study would be very useful for proofing these research proposals; it is currently among our ongoing work.

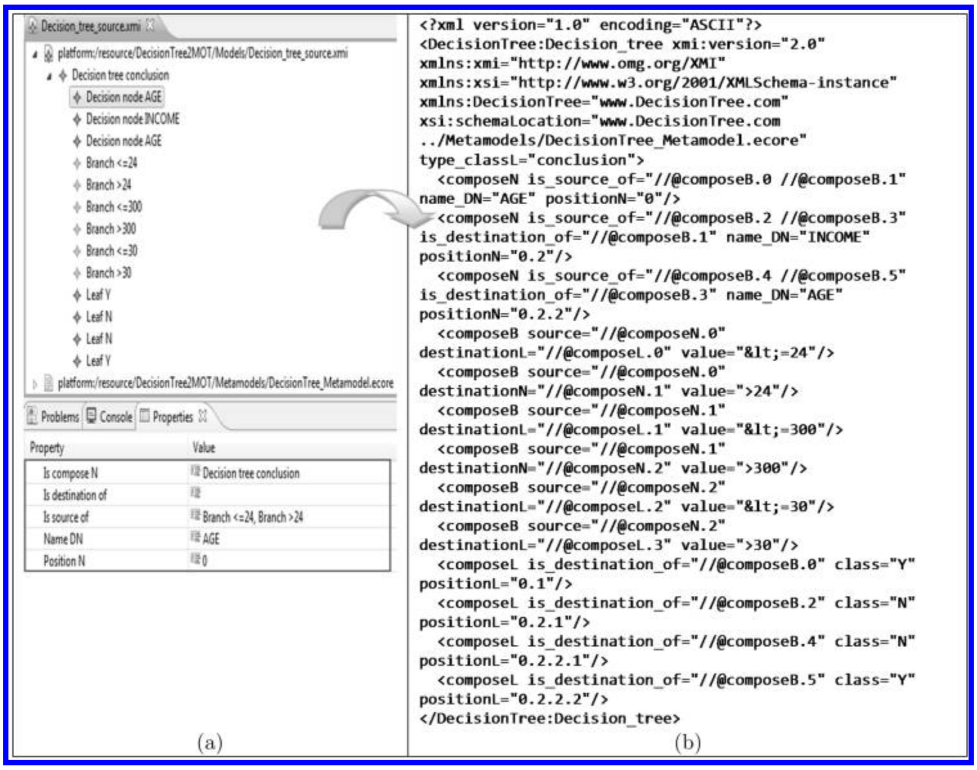
We are also working on defining methods and software tools to capture and express the tacit knowledge of experts into an explicit knowledge model in MOT.

Another future issue we should consider is the definition of a comprehensive graphical tool to display the KW model as it can help the KW Administrator managing and exploiting knowledge.

Appendix A

Under Eclipse, the *XMI* source model (cf. Fig. A.1(b)) of the DT shown in Fig. 7 is constructed by instantiating, in the properties editor (cf. Fig. A.1(a)), attributes of classes that compose the DT source MM (cf. Fig. 6).

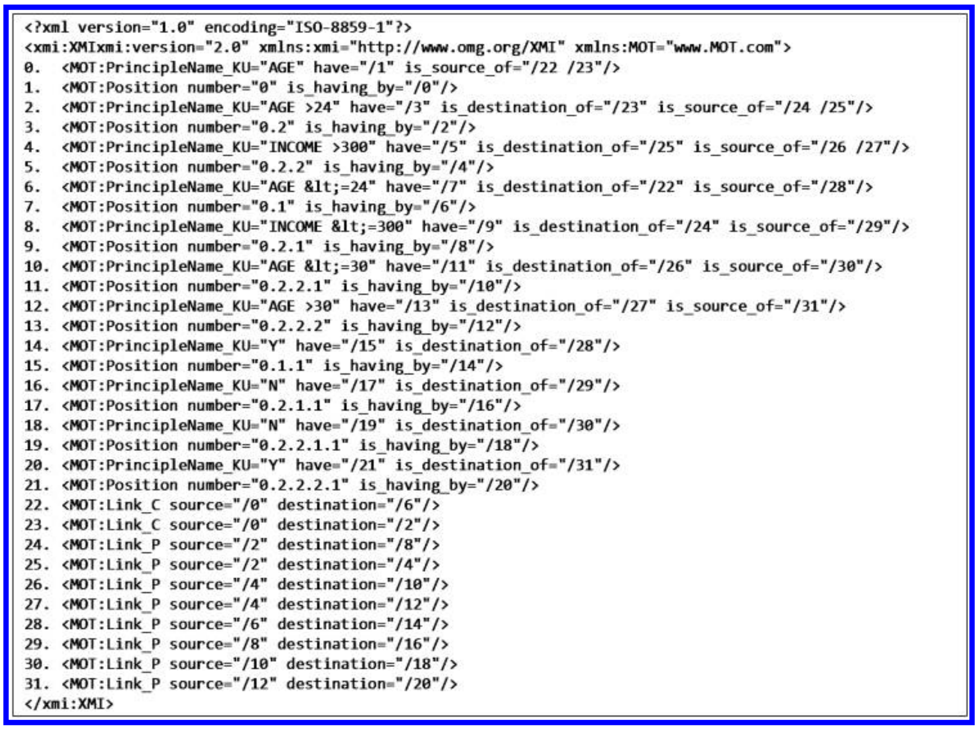
Figure A.2 shows the *XMI* target model of the DT of risky clients. This model is conforming to the MOT MM (cf., Fig. 2); it is generated based on our ATL transformation rules (cf., Sec. 5).



Property	Value
Is compose N	Decision tree conclusion
Is destination of	
Is source of	Branch <=24, Branch >24
Name DN	AGE
Position N	0

```
<?xml version="1.0" encoding="ASCII"?>
<DecisionTree:Decision_tree xmi:version="2.0"
xmlns:xmi="http://www.omg.org/XMI"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:DecisionTree="www.DecisionTree.com"
xsi:schemaLocation="www.DecisionTree.com
../Metamodels/DecisionTree_Metamodel.ecore"
type_classL="conclusion">
  <composeN is_source_of="//@composeB.0 //@composeB.1"
name_DN="AGE" positionN="0"/>
  <composeN is_source_of="//@composeB.2 //@composeB.3"
is_destination_of="//@composeB.1" name_DN="INCOME"
positionN="0.2"/>
  <composeN is_source_of="//@composeB.4 //@composeB.5"
is_destination_of="//@composeB.3" name_DN="AGE"
positionN="0.2.2"/>
  <composeB source="//@composeN.0"
destinationL="//@composeL.0" value="&lt;=24"/>
  <composeB source="//@composeN.0"
destinationN="//@composeN.1" value=">24"/>
  <composeB source="//@composeN.1"
destinationL="//@composeL.1" value="&lt;=300"/>
  <composeB source="//@composeN.1"
destinationN="//@composeN.2" value=">300"/>
  <composeB source="//@composeN.2"
destinationL="//@composeL.2" value="&lt;=30"/>
  <composeB source="//@composeN.2"
destinationL="//@composeL.3" value=">30"/>
  <composeL is_destination_of="//@composeB.0" class="Y"
positionL="0.1"/>
  <composeL is_destination_of="//@composeB.2" class="N"
positionL="0.2.1"/>
  <composeL is_destination_of="//@composeB.4" class="N"
positionL="0.2.2.1"/>
  <composeL is_destination_of="//@composeB.5" class="Y"
positionL="0.2.2.2"/>
</DecisionTree:Decision_tree>
```

Fig. A.1. The Sample Reflective Ecore Model Editor and the generated *XMI* file representing the decision tree of risky customers (cf. Fig. 7).

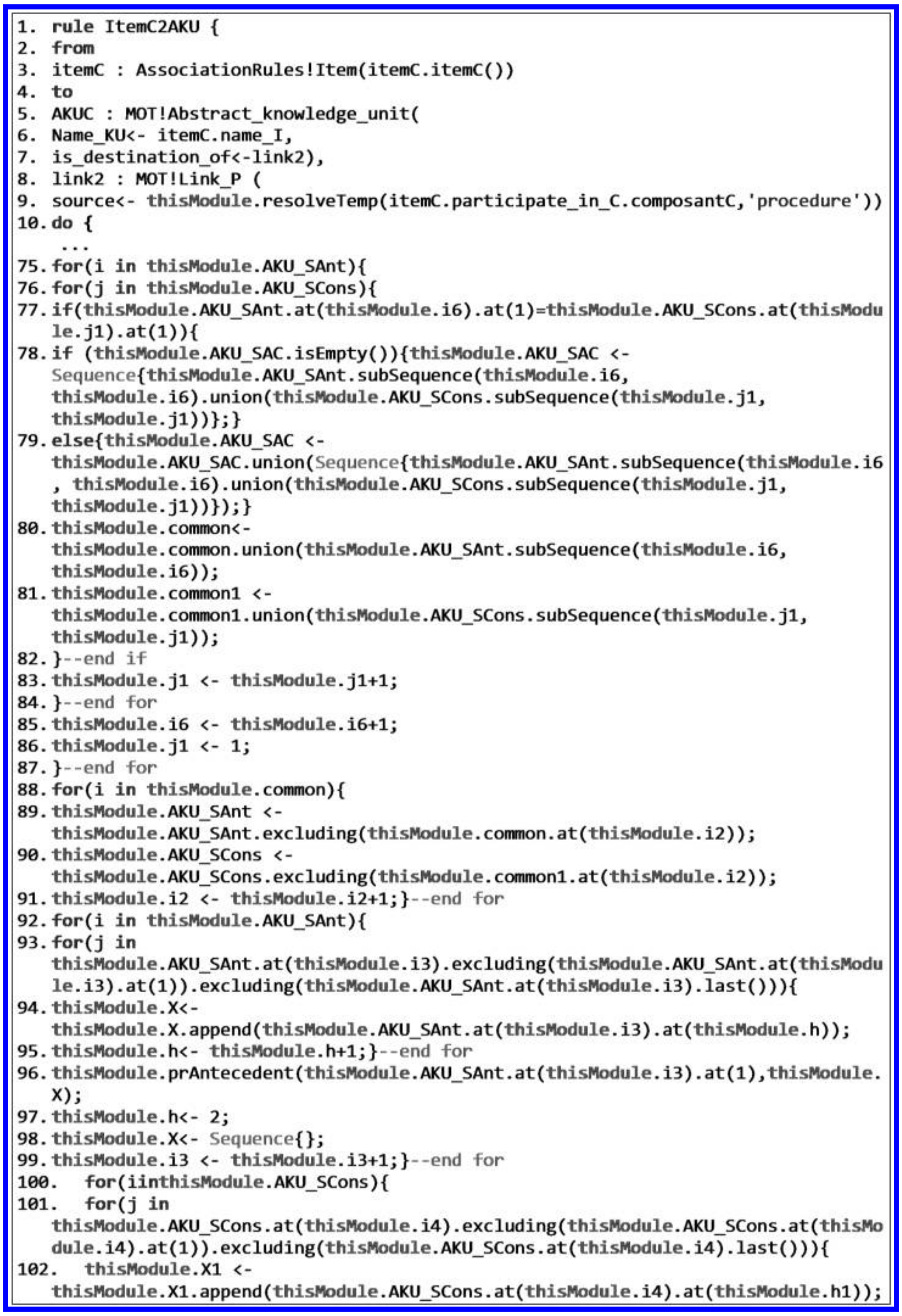


```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xmi:XMIxmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:MOT="www.MOT.com">
0.  <MOT:PrincipleName_KU="AGE" have="/1" is_source_of="/22 /23"/>
1.  <MOT:Position number="0" is_having_by="/0"/>
2.  <MOT:PrincipleName_KU="AGE >24" have="/3" is_destination_of="/23" is_source_of="/24 /25"/>
3.  <MOT:Position number="0.2" is_having_by="/2"/>
4.  <MOT:PrincipleName_KU="INCOME >300" have="/5" is_destination_of="/25" is_source_of="/26 /27"/>
5.  <MOT:Position number="0.2.2" is_having_by="/4"/>
6.  <MOT:PrincipleName_KU="AGE &lt;=24" have="/7" is_destination_of="/22" is_source_of="/28"/>
7.  <MOT:Position number="0.1" is_having_by="/6"/>
8.  <MOT:PrincipleName_KU="INCOME &lt;=300" have="/9" is_destination_of="/24" is_source_of="/29"/>
9.  <MOT:Position number="0.2.1" is_having_by="/8"/>
10. <MOT:PrincipleName_KU="AGE &lt;=30" have="/11" is_destination_of="/26" is_source_of="/30"/>
11. <MOT:Position number="0.2.2.1" is_having_by="/10"/>
12. <MOT:PrincipleName_KU="AGE >30" have="/13" is_destination_of="/27" is_source_of="/31"/>
13. <MOT:Position number="0.2.2.2" is_having_by="/12"/>
14. <MOT:PrincipleName_KU="Y" have="/15" is_destination_of="/28"/>
15. <MOT:Position number="0.1.1" is_having_by="/14"/>
16. <MOT:PrincipleName_KU="N" have="/17" is_destination_of="/29"/>
17. <MOT:Position number="0.2.1.1" is_having_by="/16"/>
18. <MOT:PrincipleName_KU="N" have="/19" is_destination_of="/30"/>
19. <MOT:Position number="0.2.2.1.1" is_having_by="/18"/>
20. <MOT:PrincipleName_KU="Y" have="/21" is_destination_of="/31"/>
21. <MOT:Position number="0.2.2.2.1" is_having_by="/20"/>
22. <MOT:Link_C source="/0" destination="/6"/>
23. <MOT:Link_C source="/0" destination="/2"/>
24. <MOT:Link_P source="/2" destination="/8"/>
25. <MOT:Link_P source="/2" destination="/4"/>
26. <MOT:Link_P source="/4" destination="/10"/>
27. <MOT:Link_P source="/4" destination="/12"/>
28. <MOT:Link_P source="/6" destination="/14"/>
29. <MOT:Link_P source="/8" destination="/16"/>
30. <MOT:Link_P source="/10" destination="/18"/>
31. <MOT:Link_P source="/12" destination="/20"/>
</xmi:XMI>
```

Fig. A.2. MOT Model generated in *XMI* format from the decision tree of risky customers (cf. Fig. 7).

Appendix B

In the rule *ItemC2AKU* (cf. Fig. B.1), we determine the three subsets S_{Ant}, S_{Cons} and S_{AC}, respectively, with the three vectors AKU_S_{Ant} (line 89), AKU_S_{Cons} (line 90) and AKU_S_{AC} (lines 78–79). On the contrary, we define rules of type *called rule prAntecedent*, *prConsequent*, *proc*, *prproc* and *prDoubleRole*. They take as a parameter the name of the AKU as well as its inbound links and/or its outbound links. These rules are invoked in lines 96, 106, 105, 120 and 121 of Fig. B.1 to apply, respectively, rules $\text{I}_{S_{\text{Ant}}}$2Pr, $\text{I}_{S_{\text{Cons}}}$2Pr, $\text{I}_{S_{\text{Cons}}}$2Proc, $\text{I}_{S_{\text{AC}}}$2PrProc and $\text{I}_{S_{\text{AC}}}$2Pr.



```
rule ItemC2AKU {
from
itemC : AssociationRules!Item(itemC.itemC())
to
AKUC : MOT!Abstract_knowledge_unit(
Name_KU<- itemC.name_I,
is_destination_of<-link2),
link2 : MOT!Link_P (
source<- thisModule.resolveTemp(itemC.participate_in_C.composantC,'procedure'))
do {
 ...
for(i in thisModule.AKU_SAnt){
for(j in thisModule.AKU_SCons){
if(thisModule.AKU_SAnt.at(thisModule.i6).at(1)=thisModule.AKU_SCons.at(thisModule.j1).at(1)){
if (thisModule.AKU_SAC.isEmpty()){thisModule.AKU_SAC <- Sequence{thisModule.AKU_SAnt.subSequence(thisModule.i6, thisModule.i6).union(thisModule.AKU_SCons.subSequence(thisModule.j1, thisModule.j1))};}
else{thisModule.AKU_SAC <- thisModule.AKU_SAC.union(Sequence{thisModule.AKU_SAnt.subSequence(thisModule.i6, thisModule.i6).union(thisModule.AKU_SCons.subSequence(thisModule.j1, thisModule.j1))});}
thisModule.common<- thisModule.common.union(thisModule.AKU_SAnt.subSequence(thisModule.i6, thisModule.i6));
thisModule.common1 <- thisModule.common1.union(thisModule.AKU_SCons.subSequence(thisModule.j1, thisModule.j1));
}--end if
thisModule.j1 <- thisModule.j1+1;
}--end for
thisModule.i6 <- thisModule.i6+1;
thisModule.j1 <- 1;
}--end for
for(i in thisModule.common){
thisModule.AKU_SAnt <- thisModule.AKU_SAnt.excluding(thisModule.common.at(thisModule.i2));
thisModule.AKU_SCons <- thisModule.AKU_SCons.excluding(thisModule.common1.at(thisModule.i2));
thisModule.i2 <- thisModule.i2+1;}--end for
for(i in thisModule.AKU_SAnt){
for(j in thisModule.AKU_SAnt.at(thisModule.i3).excluding(thisModule.AKU_SAnt.at(thisModule.i3).at(1)).excluding(thisModule.AKU_SAnt.at(thisModule.i3).last())){
thisModule.X<- thisModule.X.append(thisModule.AKU_SAnt.at(thisModule.i3).at(thisModule.h));
thisModule.h<- thisModule.h+1;}--end for
thisModule.prAntecedent(thisModule.AKU_SAnt.at(thisModule.i3).at(1),thisModule.X);
thisModule.h<- 2;
thisModule.X<- Sequence{};
thisModule.i3 <- thisModule.i3+1;}--end for
   for(iinthisModule.AKU_SCons){
   for(j in thisModule.AKU_SCons.at(thisModule.i4).excluding(thisModule.AKU_SCons.at(thisModule.i4).at(1)).excluding(thisModule.AKU_SCons.at(thisModule.i4).last())){
   thisModule.X1 <- thisModule.X1.append(thisModule.AKU_SCons.at(thisModule.i4).at(thisModule.h1));
```

Fig. B.1. Extract of the rule *ItemC2AKU* formalised in ATL.

```
thisModule.h1 <- thisModule.h1+1;}--end for
if(thisModule.AKU_SCons.at(thisModule.i4).last() ='action'){--or conclusion
thisModule.proc(thisModule.AKU_SCons.at(thisModule.i4).at(1),thisModule.X1);
}
else{thisModule.prConsequent(thisModule.AKU_SCons.at(thisModule.i4).at(1),thisModule.X1);}
thisModule.h1 <- 2;
thisModule.X1 <- Sequence{};
thisModule.i4 <- thisModule.i4+1;}--end for
for(i in thisModule.common){
for(j in Set{1}){
for(k in thisModule.AKU_SAC.at(thisModule.i5).at(j).excluding(thisModule.AKU_SAC.at(thisModule.i5).at(j).at(1)).excluding(thisModule.AKU_SAC.at(thisModule.i5).at(j).last())){
thisModule.X2 <- thisModule.X2.append(thisModule.AKU_SAC.at(thisModule.i5).at(j).at(thisModule.h2));
thisModule.h2 <- thisModule.h2+1;}--end for
for(k in thisModule.AKU_SAC.at(thisModule.i5).at(j+1).excluding(thisModule.AKU_SAC.at(thisModule.i5).at(j+1).at(1)).excluding(thisModule.AKU_SAC.at(thisModule.i5).at(j+1).last())){
thisModule.X3 <- thisModule.X3.append(thisModule.AKU_SAC.at(thisModule.i5).at(j+1).at(thisModule.h3));
thisModule.h3 <- thisModule.h3+1;}--end for
}--end for
if(thisModule.AKU_SAC.at(thisModule.i5).at(1).last() ='action'){
thisModule.prproc(thisModule.AKU_SAC.at(thisModule.i5).at(1).at(1),thisModule.X2,thisModule.X3);}
else{thisModule.prDoubleRole(thisModule.AKU_SAC.at(thisModule.i5).at(1).at(1),thisModule.X2,thisModule.X3);}
thisModule.h2 <- 2;
thisModule.X2 <- Sequence{};
thisModule.h3 <- 2;
thisModule.X3 <- Sequence{};
thisModule.i5 <- thisModule.i5+1;}--end for
}
}--end do
}--end rule
```

Fig. B.1. (*Continued*)

References

ATLAS, G (2006). ATL User Manual, version 0.7. Available at http://www.eclipse.org/atl/documentation/old/ATL_User_Manual[v0.7].pdf. Accessed on 29 October 2018.

Ayadi, R, Y Hachaichi, S Alshomrani and J Feki (2015). Decision tree transformation for knowledge warehousing. In *ICEIS 2015 — Proceedings of the 17th International Conference on Enterprise Information Systems*, Vol. 1, Barcelona, Spain, 27–30 April, pp. 616–623.

Ayadi, R, Y Hachaichi and J Feki (2013). Towards knowledge warehouses: Definition and architecture. In *7th Edition of the Conference on Advances in Decisional Systems*, Marrakech, Morocco, 25–27 May 2013, pp. 445–450 (in French).

Ayadi, R, Y Hachaichi and J Feki (2017a). Association rules transformation for knowledge integration and warehousing. In *Intelligent Systems Design and Applications — 17th*

International Conference on Intelligent Systems Design and Applications (ISDA 2017), Delhi, India, 14–16 December 2017, pp. 378–388.

Ayadi, R, Y Hachaichi and J Feki (2017b). MOT knowledge model integration rules for knowledge warehousing. In *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of 21st International Conference on KES-2017*, Marseille, France, 6–8 September 2017, pp. 544–553.

Basciani, F, J Di Rocco, D Di Ruscio, L Iovino and A Pierantonio (2016). Automated clustering of metamodel repositories. In *Advanced Information Systems Engineering: 28th International Conference*, S Nurcan, P Soffer, M Bajec and J Eder (eds.), CAiSE 2016, Ljubljana, Slovenia, 13–17 June 2016, pp. 342–358. Cham: Springer International Publishing.

Canas, AJ, KM Ford, JD Novak, P Hayes, TR Reichherzer and N Suri (2001). Online concept maps: Enhancing collaborative learning by using technology with concept maps. *The Science Teacher*, 68(4), 49–51.

Chauvet, V and C Ghetty (2003). *A Conceptual Approach to Knowledge Creation Through Virtual Communities*. Aix-Marseille (in French).

Collins, A and M Quillian (1969). Retrieval time from semantic memory. *Journal of Verbal Learning and Verbal Behavior*, 8(2), 240–247.

Davenport, TH and L Prusak (1998). *Working Knowledge: How Organizations Manage What They Know*. Boston, Massachusetts: Harvard Business School Press.

Dinarelli, M, A Moschitti and G Riccardi (2010). Hypotheses selection for re-ranking semantic annotations. In *2010 IEEE Spoken Language Technology Workshop*, pp. 407–411.

Doshi, D, H Jain, N Tiwari and B Jasuja (2015). Adaptive learning using data mining techniques. *International Journal of Advanced Research in Computer Science and Software Engineering*, 5(2), 583–585.

Drucker, PF (1993). *Post-Capitalist Society*. New York: HarperBusiness.

Dymond, A (2002). The knowledge warehouse: The next step beyond the data warehouse. In *Data Warehousing and Enterprise Solutions*. SAS Users Group International 27.

Erata, F, M Challenger and G Kardas (2015). *Review of Model-to-Model Transformation Approaches and Technologies*, Eindhoven, The Netherlands.

Giandini, R, C Pons and G Pérez (2009). A two-level formal semantics for the {QVT} language. In *Memorias de la {XII} Conferencia Iberoamericana de Software Engineering (CIbSE 2009)*, Medellin, Colombia, 13–17 April, pp. 73–86.

Hamad, MM and BA Qader (2014). Knowledge-driven decision support system based on knowledge warehouse and data mining for market management. *International Journal of Application or Innovation in Engineering and Management*, 3(1), 139–147.

Héon, M, J Basque and G Paquette (2010). Semantics validation of a semi-formal knowledge model with OntoCASE. In *Act of 21st Francophone Days of Knowledge Engineering*, Nimes, France, pp. 55–66 (in French).

Héon, M, G Paquette and J Basque (2009). Assisted methodology of an ontology design based on a semi-formal consensual conceptualization. In *IC 2009: 20th Francophone Days of Knowledge Engineering*, Hammamet, Tunisia, pp. 61–72 (in French).

Huang, K, YW Lee and RY Wang (1999). *Quality Information and Knowledge*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

Hussain, MS, HS Chandio, S Muhammad and HS Talpur (2012). Knowledge Warehouse Framework. *International Journal of Engineering Innovation and Research*, 1(3), 262–270.

Irfan, R and M Uddin-Shaikh (2010). Enhance knowledge management process for group decision making. In *Proc. 2010 Second International Conference on Computer Engineering and Applications — Volume 01*, pp. 66–70. New York: IEEE Computer Society.

Jouault, F, F Allilaire, J Bézivin and I Kurtev (2008). ATL: A model transformation tool. *Science of Computer Programming*, 72(1–2), 31–39.

Kerschberg, L (2001). Knowledge management in heterogeneous data warehouse environments. In *International Conference on Data Warehousing and Knowledge Discovery*, pp. 1–10. Berlin: Springer.

Koegel, M and J Helming (2016). What every Eclipse developer should know about EMF. Available at http://eclipsesource.com/blogs/tutorials/emf-tutorial/. Accessed on 29 October 2018.

Liebowitz, J and M Frank (2016). *Knowledge Management and E-Learning*. CRC Press. Retrieved from https://books.google.tn/books?id=asFxYdhLjEUC.

Mach, M (1995). The learning organization as a system of transformation of knowledge in value. *French Review of Management: The Paths of Knowledge of the Company*, 105, 43–55 (in French).

Michael, Y (1999). The knowledge warehouses reusing knowledge components. *Performance Improvement Quarterly*, 12(3), 132–140.

Nemati, HR, DM Steiger, LS Iyer and RT Herschel (2002). Knowledge warehouse: An architectural integration of knowledge management, decision support, artificial intelligence and data warehousing. *Decision Support Systems*, 33(2), 143–161.

Nonaka, I (1991). The knowledge creating company. *Harvard Businesses Review*, Nov–Dec, 96–104.

Nonaka, I and H Takeuchi (1995). *The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation*. New York: Oxford University Press.

OMG (2003). MDA Guide Version 1.0.1., J Miller and J Mukerji (eds.). Available at http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf. Accessed on 29 October 2018.

OMG (2011). Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Available at http://www.omg.org/spec/QVT/1.1. Accessed on 29 October 2018.

OMG (2012). Information technology — Object Management Group Object Constraint Language (OCL). Published by ISO/IEC 19507.

OMG (2014). Information technology — Object Management Group XML Metadata Interchange (XMI). Published by ISO/IEC 19509.

OMG (2015). About OMG. Available at https://www.omg.org/about/index.htm. Accessed on 23 May 2019.

Paquette, G (1996). Modeling with object types — A representation method for learning and task aids systems. *Educational Science and Techniques*, 3(1), 9–42 (in French).

Paquette, G (2002). *Knowledge and Skills Modeling: A Graphical Language for Designing and Learning*. Sainte-Foy: University of Quebec Press (in French).

Paquette, G (2010). *Visual Knowledge Modeling for Semantic Web Technologies: Models and Ontologies*. Hershey, PA: IGI Global.

Qing-Lan, H and H Zhi-Jun (2009). Research on cost control DSS based on knowledge warehouse. In *Proc. 6th International Conf. Fuzzy Systems and Knowledge Discovery*, Vol. 7, pp. 357–361. Piscataway, NJ, USA: IEEE Press.

Silwattananusarn, T and K Tuamsuk (2012). Data mining and its applications for knowledge management: A literature review from 2007 to 2012. *International Journal of Data Mining and Knowledge Management Process (IJDKP)*, 2(5), 13–24.

Suciu, I, C Fernandez and A Ndiaye (2012). How to acquire scientific knowledge for university to industry knowledge transfer. In *Proceedings of the Fourth IEEE International Conf. Information Process and Knowledge Management*, pp. 24–27.

Zaki, MJ and W Meira (2014). *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge: Cambridge University Press.

Zhang, H and Y Liang (2006). A knowledge warehouse system for enterprise resource planning systems. *Systems Research and Behavioral Science*, 23(2), 169–176.