Expert Systems with Applications 39 (2012) 10420–10437

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

An MDA approach to knowledge engineering

Nicolas Prat [a], Jacky Akoka [b,*], Isabelle Comyn-Wattiau [c]

[a] ESSEC Business School, 1, Av. B. Hirsch, 95000 CERGY, France
[b] CEDRIC-CNAM & Institut Telecom/TEM Research, 2 Rue Conté, 75003 Paris, France
[c] CEDRIC-CNAM & ESSEC Business School, 2 Rue Conté, 75003 Paris, France

ARTICLE INFO

Keywords:
Knowledge engineering
Model-driven approach
Knowledge model
Production rule
CommonKADS
Production rule representation (PRR)

ABSTRACT

This paper proposes an MDA approach to knowledge engineering, centered on the CommonKADS knowledge model. The latter corresponds to the CIM level of MDA whereas PRR, which represents production rules and rulesets, corresponds to the PIM level. The paper explores the mapping between CommonKADS knowledge models and production rules and rulesets based on PRR. Mapping CommonKADS knowledge models into PRR is very useful, due to the fact that the CIM level remains relatively unexplored, despite its key role in MDA. This motivates our choice to focus on the CIM and PIM levels. Furthermore, the mapping between PIM and PSM (i.e. the implementation of production rules in specific rule-based systems) constitutes less of an issue. To map CommonKADS knowledge models into PRR production rules and rulesets, we propose and illustrate a set of transformations. To ease these transformations, we start by grouping elements of the CommonKADS knowledge models into so-called "inference groups". We propose and illustrate an algorithm that defines these inference groups automatically. The definition of transformations between models (CIM to PIM levels) requires a specific metamodel for CommonKADS as well as a dedicated metamodel for PRR. Unlike PRR, there is no published CommonKADS metamodel. This paper proposes a comprehensive CommonKADS knowledge metamodel. We describe and discuss an example, applying the whole approach.

1. Introduction

Fox (2011) defines knowledge engineering as "the engineering discipline that involves integrating knowledge into computer systems in order to solve complex problems normally requiring a high level of human expertise". The activity of knowledge engineering typically results in expert systems, with knowledge represented in the form of production rules. The latter enable to represent business rules in information systems. A fair number of rule engines support production rules, among which Jess, Ilog JRules, and JBoss. Although knowledge engineering is now an established academic discipline, practitioners often regret a lack of methodological support for the development of rule-based systems (Zacharias, 2008). This paper presents a contribution in the domain of knowledge engineering. More specifically, we provide methodological support for the mapping of knowledge models into production rules, for subsequent mapping into expert systems. One originality of our work is to combine the knowledge engineering method CommonKADS (Schreiber et al., 2000) with the model-driven approach (Object Management Group, 2003), which originated in the domain of software engineering.

CommonKADS is a widespread methodology, and the most influential academic method in knowledge engineering to date (Zacharias, 2009). The methodology proposes several models, with a strong focus on knowledge models. These models require further efforts before implementation.

In model-driven approaches, models drive the software development process. The latter distinguishes different abstraction levels: conceptual, logical and physical. Models of lower abstraction levels result from mapping models from upper levels. This distinction between abstraction levels is now a consensus and forms the core of information systems engineering. The Model-Driven Architecture (*MDA*) emerged more recently. It is a standard model-driven approach proposed by the Object Management Group (OMG). It facilitates the definition of guidelines, helping learners in the appropriation of the different design steps. MDA also facilitates tool support in the development of information systems, thus directly affecting productivity and maintainability. MDA presents the advantage of separating design aspects from architecture issues. To this end, it defines three abstraction levels (quite similarly to the traditional distinction between conceptual, logical and physical levels in information systems engineering). The Computation Independent Model (CIM) constitutes the more abstract level. It represents the context and purpose of the information system without any computational consideration. It focuses on the business and the conceptual considerations. The Platform Independent

* Corresponding author.
E-mail addresses: prat@essec.edu (N. Prat), akoka@cnam.fr (J. Akoka), wattiau@cnam.fr (I. Comyn-Wattiau).

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2012.02.010

Model (PIM) describes the behavior and structure of the information system, regardless of the implementation choices. Finally, at the third level, the Platform Specific Model (PSM) contains all required information allowing developers to build the code and to execute the resulting application. Applying MDA to the development of knowledge systems requires a mapping between MDA abstraction levels and knowledge models.

Gartner, Inc. (2010) considers that MDA has the potential for providing high payoff to companies choosing this development approach. Among the several benefits that this approach provides, let us mention portability, interoperability and reusability (Zhu, 2006). Although MDA primarily focuses on the development of transactional applications, it also has the potential to apply to the development of other types of applications. For example, MDA, and more generally model-driven approaches, provide major benefits to the development of data warehouses (Mazon & Trujillo, 2008; Prat, Akoka, & Comyn-Wattiau, 2006). Similarly to data warehousing, MDA also has great potential to apply in the development of rule-based systems and, more generally, in knowledge engineering. As Baumeister, Reutelshoefer, and Puppe (2011) point out, knowledge engineers often need to face the flexibility/productivity dilemma. MDA provides a means for solving this dilemma, by distinguishing between the three abstraction levels. Thanks to this distinction, after defining a model at the PIM level for instance, developers may generate several models at the PSM level, depending on the target expert system.

Some previous work has already combined MDA with knowledge engineering and knowledge systems. However it suffers from two major limitations:

1. It often focuses on rule languages (Wagner, Antoniou, Tabet, & Boley, 2004). Knowledge engineering involves the definition of rules, but extends far beyond this activity.
2. When the focus is not just rule languages but knowledge engineering, CommonKADS knowledge models are placed at the PIM level (Abdullah, Benest, Paige, & Kimble, 2007; Cañadas, Palma, & Tùnez, 2005). We argue that the PIM level is not the right level for CommonKADS knowledge models. These models provide all the tools required to analyze tasks in terms of knowledge involved at different granularities. Knowledge analysis aims at studying the tasks at a conceptual level. Thus, we consider CommonKADS knowledge models as the CIM level of MDA.

To provide methodological support for mapping knowledge models into production rules, the originality of our approach lies in combining CommonKADS knowledge models (CIM level of MDA) with production rules representation language (PRR) proposed by the Object Management Group (2009). OMG has defined PRR at the PIM level of MDA. Production rules constitute a very rich representation of a large body of knowledge. Many languages enable the definition of production rules. Admittedly, the model-driven approach forms one of the theoretical bases of CommonKADS. However, we argue that the more abstract levels, and more particularly the knowledge model, are the central contribution of the CommonKADS methodology. Thus, by combining this knowledge model with a standard for production rule representation (PRR), we combine the advantages of CommonKADS with a standard for production rule representation.

The paper contributes to methodological support in knowledge engineering in the following ways:

1. It proposes an MDA approach to knowledge engineering, centered on the CommonKADS knowledge model. The CommonKADS knowledge model corresponds to the CIM level of MDA; PRR, which represents production rules and rulesets, corresponds to the PIM level.
2. The paper explores the mapping between CommonKADS knowledge models (CIM level of MDA) and production rules and rulesets based on PRR (PIM level). There exists a possible convergence between CommonKADS and PRR, since both rely on the unified modeling language (UML: Object Management Group, 2010c). However, the two are not completely related. Therefore, mapping CommonKADS knowledge models into PRR constitutes an open research problem. We argue that such a mapping can be very useful. We concur with Gartner, Inc. (2010) that the CIM level remains relatively unexplored, despite its key role in MDA. This motivates our choice to focus on the CIM and PIM levels. Furthermore, the mapping between PIM and PSM (i.e. the implementation of production rules in specific rule-based systems) constitutes less of an issue (Object Management Group, 2009). To map CommonKADS knowledge models into PRR production rules and rulesets, we propose and illustrate a set of transformations. To ease these transformations, we start by grouping elements of the CommonKADS knowledge models into so-called "inference groups". We propose and illustrate an algorithm that defines these inference groups automatically.
3. The definition of transformations between models (CIM to PIM levels) requires a specific metamodel for CommonKADS as well as a dedicated metamodel for PRR. Unlike PRR, there is no published CommonKADS metamodel. This paper proposes a comprehensive CommonKADS knowledge metamodel. Complying with MDA, it represents the metamodel with UML.

The outline of the paper follows. Section 2 surveys related work. It encompasses reference models at the different levels of knowledge engineering. It synthesizes and compares previous related work referencing CommonKADS as well as MDA and PRR concepts. Section 3 presents the example used throughout the paper to illustrate our approach. Section 4 describes the main concepts of the two metamodels: CommonKADS and PRR metamodels. Section 5 presents our MDA-based knowledge engineering approach. This section situates the approach within the MDA framework and presents the transformations mapping CommonKADS knowledge models into PRR and activity models. Finally, Section 6 discusses the contributions and implications of this research, as well as future research opportunities.

2. Related work

In this section, we first present a brief review of research on rule modeling, especially with the PRR language. The second part is dedicated to knowledge engineering methods, with a particular accent on CommonKADS.

2.1. Rule modeling

Abdullah et al. (2007) propose a UML profile for modeling knowledge-based systems within the context of MDA. They define a mapping of the profile elements to Jess concepts. Thus, their mapping is concentrated on PIM to PSM transformation. Their choice of a specific UML profile is justified by the immaturity of PRR standardization work in 2007. They do not address the CIM viewpoint.

Several authors propose different meta-models to represent production rules. Lukichev and Wagner (2006) extend the UML metamodel with the concept of rule and describe an implementation. Milanovic et al. (2009) describe a bridge for transforming abstract and concrete syntax of Web rule languages based on a rule metamodel.

Wagner et al. (2004) have been among the first authors to attempt a link between MDA viewpoints and rule languages. Their

proposal, known as the Rule Markup Language (RuleML) Initiative consists of a general Web rule language framework and relates it to the MDA and UML. They consider that there is no rule language at the CIM level, which was true at that time.

Zur Muehlen and Indulska (2010) present and compare four rule-modeling languages, namely simple rule markup language (SRML), semantic web rules language (SWRL), production rule representation (PRR), and semantics of business vocabulary and business rule (SBVR) specifications. The main result of their comparison shows that no single language can be considered as internally complete with respect to the Bunge–Wand–Weber representation theory.

Martínez-Fernández, Martínez, and González-Cristóbal (2009) present a novelty by proposing to introduce business rules at all levels of MDA. They recommend the rules interchange format (RIF) for Web-oriented applications and PRR for UML based tools and provide automatic mappings between both standards. Prat, Comyn-Wattiau, and Akoka (2011) used PRR to represent aggregation knowledge in data warehouses.

Cabot, Pau, and Ravento (2010) suggest to consider the transformation from UML/OCL to SBVR. SBVR specifications are at the CIM level.

Diouf, Maabout, and Musumbu (2007) propose an approach that automatically generates some business rules. To achieve such a result, they combine concepts coming from Model Driven Architecture and Semantic Web using the Ontology Definition Metamodel.

2.2. CommonKADS

As pointed out by several authors (in particular Schreiber et al. (2000), Nabil, El-Korany, & Sharaf Eldin (2008) and Kingston (1998)) the CommonKADS (Knowledge Acquisition and Development Systems) development methodology is used to model expert systems in terms of three types of knowledge, namely, domain knowledge, inference knowledge, and task knowledge. Domain knowledge represents declarative knowledge. Inference knowledge represents the knowledge-based inferences performed during problem solving. Finally, task knowledge defines a procedural ordering on inferences. CommonKADS knowledge models can be specified diagrammatically, and textually using the CommonKADS modeling language (CML). CML is a structured, semi-formal language. Schreiber et al. (2000) describe the syntax of CML, but do not provide a CommonKADS knowledge metamodel. CommonKADS knowledge metamodels have been proposed in the literature, but they are too succinct to be useful in our approach. The CommonKADS methodology encompasses several models (Fig. 1). At the context level, three different models are required, namely organization, task and agent models. The organization model supports the analysis of an organization. The task model describes tasks that are performed in the organizational environment whereas the agent model describes capabilities, norms, preferences, and permissions of agents. At the concept level, CommonKADS considers knowledge and communication models. The knowledge model provides an implementation-independent description of knowledge involved in a task. The communication model represents the communicative transactions between agents. Finally, at the artefact level, the design model describes the structure of the system that needs to be constructed.

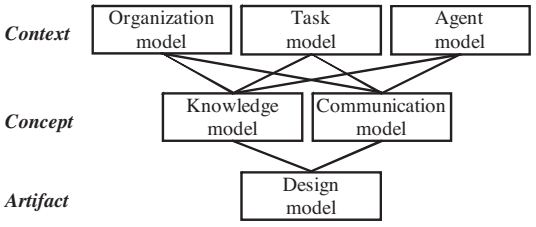

Fig. 1. CommonKADS models (Schreiber et al., 2000).

CommonKADS is currently one of the most used methodologies for knowledge-based systems development. As an example, Andrade, Ares, García, Rodríguez, and Suárez (2010) use CommonKADS to design a knowledge-based system that makes it possible to evaluate the knowledge maturity of an organization. They mention CommonKADS as the European *de facto* standard for knowledge-based system analysis and development. Hasan and Isaac (2011) illustrate the application of CommonKADS to web-based expert systems development. They combine the MAS-CommonKADS agent oriented methodology with Model-View-Controller Architecture and optimization strategies. Sigut, Piñeiro, González, and Torres (2007) describe a knowledge-based approach to supervised classifier design following the CommonKADS methodology. In this paper, the emphasis is put on Alzheimer's diagnosis. Another health example is proposed by Sutton and Patkar (2009) applying the CommonKADS methodology to the development of a knowledge-based system to diagnose breast cancer. Anya, Tawfik, Amin, Nagar, and Shaalan (2010) combine the CommonKADS methodology with the concept of activity landscape and context-aware modeling techniques in order to enrich and optimize the knowledge resource. Arguello, Des, Fernandez-Prieto, Perez, and Paniagua (2008) present a simulation framework aiming at combining the CommonKADS methodology and semantic Web technologies (OWL, SWRL, and OWL-S).

Cañadas et al. (2005) propose a meta-modeling approach which carries out automatic transformation from conceptual models at knowledge level to design models, and finally to code. However, they do not integrate the CIM viewpoint. Let us notice that CommonKADS is not considered as sufficiently adequate to tackle some knowledge engineering projects, characterized by the fact that knowledge is present at different levels of formalization (Baumeister et al., 2011). Bera, Nevo, and Wand (2005) present the knowledge requirements analysis (KRA) method that combines two methods: a knowledge engineering method (CommonKADS) and a process modeling method (EDPDT).

3. Example application

In this paper, we consider a simplified application in the area of company buyout. A buyout involves the acquiring company and the target company. Making a buyout decision is a knowledge-intensive task. Many criteria come into play. In particular, there should exist a good synergy between the two companies (in terms of marketing, human resources etc.), the acquired company should meet financial criteria (financial health), etc.

Fig. 2 shows the domain schema for this simplified application. The schema uses the notation of UML class diagrams, extended with the CommonKADS concept of rule type (represented as an ellipse). Section 4.1. will explain the concept of rule type. A candidate buyout is a couple acquirer-target. The attributes of Company and Target will serve to assess the selection criteria. This example considers only two types of criteria (financial health and market synergy). The truth value of a selection criterion indicates if this criterion is satisfied for the buyout under consideration. The buyout decision ("Buy!" or "Do not buy!") follows from the truth-value of the selection criteria.

As stated above, this example will serve as a case study to illustrate our approach. In the following section, we describe the two meta-models underlying our engineering process.

4. Description of the underlying metamodels

Our MDA approach allows knowledge engineers to map semi-automatically their CommonKADS conceptual knowledge models

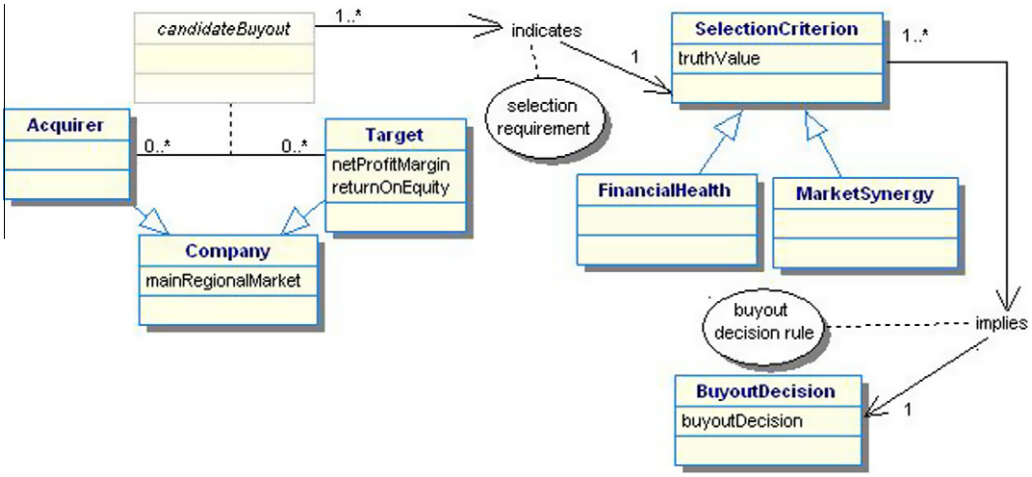

Fig. 2. Company buyout example: domain schema.

into PRR logical rules and rulesets. In this section, we describe the two main metamodels supporting this transformation process. We have defined a CommonKADS metamodel since the literature propositions were not sufficient to support our approach. On the contrary, we do not propose a PRR metamodel. We refer to the OMG PRR meta-model as a target since it contains the concepts needed to encompass CommonKADS transformation elements.

4.1. CommonKADS knowledge metamodel

Since our approach uses CommonKADS knowledge models in the CIM viewpoint, it required a MOF-compliant knowledge metamodel. We did not find a detailed, MOF-compliant CommonKADS knowledge metamodel in the literature. Therefore, our proposal comprises the definition of such a metamodel, based on the textual specification of the CommonKADS modeling language (CML). Fig. 3 exhibits the main concepts of the CommonKADS knowledge metamodel.

A knowledge model has three components: *domain knowledge*, *inference knowledge* and *task knowledge*. Domain knowledge represents the structure of the knowledge system, while inference and task knowledge represent its behavior. Another key difference is that inference and task knowledge are domain-independent, thus facilitating their reuse across several domains. The concept of knowledge role enables the mapping of inference and task knowledge with domain knowledge.

The *domain schema* describes the structure of domain knowledge. This schema uses the notation of UML class diagrams (as Fig. 2 shows for the company buyout example). The main elements of a domain schema are concepts, relations and rule types. Concepts and relations correspond to the UML concepts of class (without operations) and association, respectively. Rule types are specific to knowledge systems. They are prominently implication rule types, relating an antecedent concept to a consequent concept. The connection symbol specifies the semantics of the implication. The cardinalities of the antecedent (resp. the consequent) indicate the minimum and maximum number of expressions of the antecedent class (resp. the consequent class) in an instance of the rule type.

As an example, Fig. 2 has two rule types. Concerning the rule type "selection requirement", the connection symbol is "indicates". Instances of this rule type will have one or several expressions on the concept candidateBuyout in the antecedent, and exactly one expression on the concept SolutionCriterion in the consequent.

Besides implication rule types, constraint rule types may be defined. Constraint rule types are internal to a concept. They can be viewed as integrity constraints defined on this concept.

Knowledge bases are instances of domain schemas. They primarily contain rule type instances.

Appendix A.1 describes an example knowledge base for the company buyout example, with instances of the two rule types of Fig. 2.

Task knowledge and inference knowledge represent the behavior of the knowledge system. Functional decomposition relates tasks, task methods, inferences and transfer functions together. A *task* may be performed through one (or possibly several alternative) *task method*(s) (OR-decomposition). A task method is then specified by its functions (inferences, transfer functions or tasks), related together by a control structure (AND-decomposition). The tasks of task methods are in turn OR-decomposed into task methods, etc. Inferences and transfer functions appear at the lowest decomposition level. An *inference* carries out a primitive reasoning step. Its internal structure may itself be complex but, in the knowledge model, it is a black box. *Transfer functions* represent communication with the external world (information flows between the system and the user, at the initiative of the system or of the user). CommonKADS distinguishes four types of transfer functions. The transfer function "obtain", whereby the system asks the user to enter an information item, is frequently used. This paper focuses on this transfer function.

Knowledge roles connect task and inference knowledge to domain knowledge. *Dynamic knowledge roles* are the inputs or outputs of functions. These roles will generally map to concepts in the domain schema. In the control structure of a task method, *intermediate knowledge roles* may be used (i.e. roles that are not input or output roles of the task specified by the task method). *Static knowledge roles* specify the collection of domain knowledge used inside an inference to perform the inference. A static knowledge role generally maps to an implication rule type in the domain schema. We present the specification of the task knowledge and the inferences for the assessment task in Appendix B. This specification, adapted from Schreiber et al. (2000), is domain-independent. In the specification of the task method, the expression HAS-SOLUTION is predefined in CommonKADS.

Domain mappings perform the mapping of knowledge roles to the domain schema. In some cases, a role may map to a collection (a set or a list) of a concept.

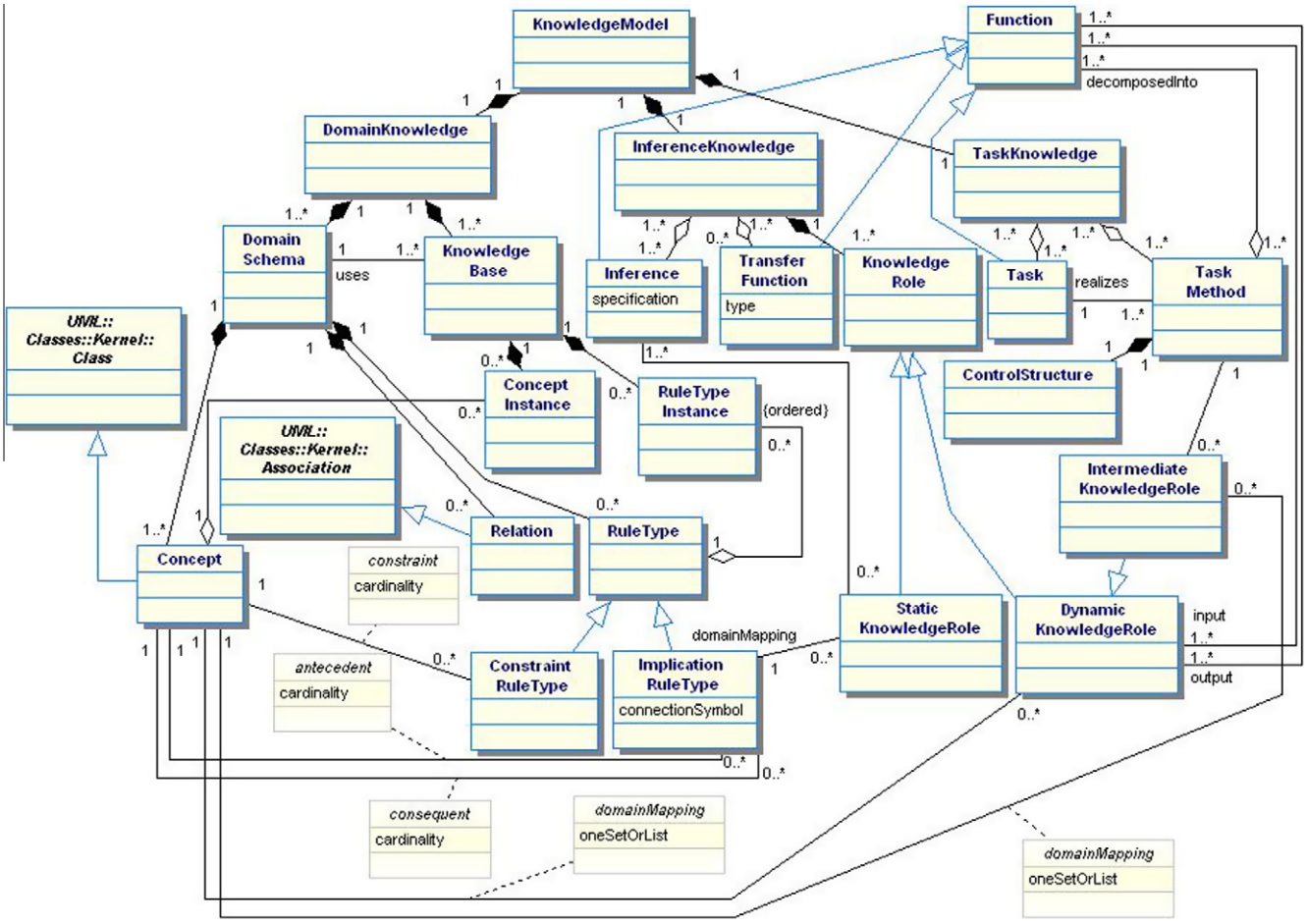

Fig. 3. CommonKADS knowledge metamodel.

In the company buyout example, the problem consists in assessing candidate buyouts according to selection criteria, in order to make a decision on the buyout. This problem is a specific case of assessment task. The knowledge roles used in the task knowledge and inferences of Appendix B are mapped into the domain schema of the company buyout example. Appendix A.2 describes (in table form) the knowledge roles and domain mappings. For example, the static knowledge role "requirements", from the inference "evaluate", maps into the rule type "selection-requirement" in the domain schema, whose rule type instances are in the knowledge base "measurement-system".

Fig. 4 details the *control structure* of task methods. This control structure may be complex. This paper considers the most commonly used elements of CommonKADS control structures. A control structure consists in a series of *statements*. A statement may be a *function call* (instantiation of a function), a *control loop*, a *conditional statement* or a *role operation*. CommonKADS uses three types of control loops: *until loops*, *while loops* and *for loops*. Control loops and conditional statements are made of several elements, which contain statements or conditions (the "for each" part of for loops has a specific structure, and refers to a dynamic role). CommonKADS widely uses two specific types of conditions: *has-solution* and *new-solution*, each referring to a function call. Among the possible role operations, *simple role assignment* assigns a dynamic role (right hand side) to another role (left hand side); *role addition* adds a role to a collection role; similarly, *role deletion* deletes a role from a collection role.

4.2. PRR metamodel

The Object Management Group (2009) has proposed the production rule representation language (PRR) for high-level (tool-independent) representation of rules. Fig. 5 is an excerpt of the PRR metamodel, with the main concepts relevant in this research (some classes of the PRR metamodel come from the UML::Classes::Kernel package).

Production rules are the central concept of PRR. A production rule may reference and modify classes, possibly defined in a UML (Object Management Group, 2010c) class diagram. A production ruleset is a collection of rules with a particular mode of execution (operational mode). The mode of execution is either sequential or inferencing. In the *sequential* mode, the order of rules in the ruleset determines their execution order. In the *inferencing* mode, the inference engine controls the execution order. PRR currently supports forward chaining only. A PRR rule may have a priority. The inference engine uses priorities for conflict resolution, i.e. to choose the rule to execute when there are several candidate rules.

A production rule is typically represented as *if [condition] then [action-list]*. An action may be the invocation of an operation associated with a class, the assignment of a value to an expression, or an action that updates the state of the rule engine (e.g. an action that asserts – creates – a new object).

Variables may appear at the ruleset level or at the rule level. The inference engine uses rule variables for binding.

An extension of the Object Constraint Language (OCL) (refer to Object Management Group, 2010a) enables the formal representation of PRR rules. Imperative expressions represent rules actions.

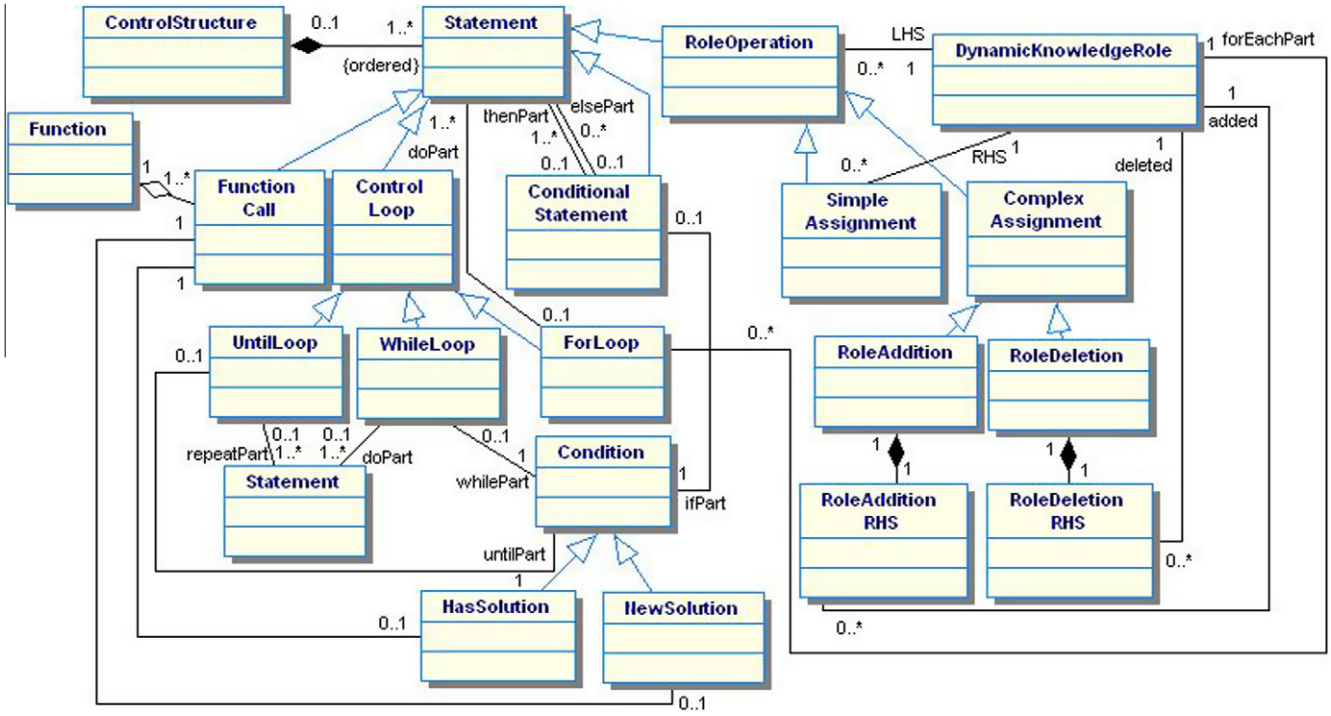

Fig. 4. CommonKADS knowledge metamodel: control structure of task methods.

Navigation call expressions represent navigation within or between objects.

The two metamodels described in this section are the root foundations of our approach detailed in the following section.

5. MDA-based knowledge engineering

After a brief overview of the approach, we detail the three main steps of our approach: defining the activities, identifying the inference groups and defining transformations for obtaining the PRR rules and rulesets.

5.1. Overview of the approach

Based on MDA, this paper provides methodological support for the mapping of knowledge models into production rules, for subsequent mapping into expert systems. The originality of the approach lies in combining CommonKADS knowledge models (CIM viewpoint of MDA) with PRR (PIM viewpoint).

At the analysis level (corresponding to the CIM viewpoint of MDA), CommonKADS defines other models besides the knowledge model (Fig. 1): the organization, task, agent and communication models. This paper considers the knowledge and communication models as given (the CommonKADS methodology clearly explains how to build the organization, task and agent models, and to derive the knowledge and communication models from these models). The communication model specifies the information exchange procedures to realize the knowledge transfer between agents. In the present paper, the focus is on the specification of knowledge. The paper does not consider the specification of dialogue, and assumes that there are only two agents (the knowledge system and its human user). Consequently, in the CIM viewpoint, our approach only considers the knowledge model.

Although PRR plays a central role in the PIM viewpoint, other models are necessary in this viewpoint. The PRR specification combines PRR with UML classes; it also suggests to combine PRR with UML activities (without explaining how). Consequently, in our approach, UML class and activity models complement PRR in the PIM viewpoint.

This paper focuses on CIM and PIM, and on the mapping between these two viewpoints. Moving from PIM to PSM and code is less of an issue, especially since several rule engine vendors participate in the specification of PRR.

In line with the recommendation of the Object Management Group (2003, 2010b), our approach complies with MOF (meta-object facility) and relies on three meta-layers: M3 (MOF, the meta-metamodel), M2 (metamodels), and M1 (models). In particular, the approach defines the CommonKADS metamodel (developed in Section 4.1) as an instance of the MOF.

Fig. 6 illustrates the knowledge engineering approach, combining MDA viewpoints and meta-layers. This schema draws from (de Sainte Marie & Taylor, 2007). Our approach focuses on the models in the gray rounded rectangle. PR-RIF stands for production rule interchange format.

Using the notation of UML activities (Object Management Group, 2010c), Fig. 7 illustrates the steps of CIM to PIM transformation in our approach. Defining UML classes (in the PIM viewpoint) from CommonKADS knowledge models (CIM) is relatively straightforward: the CommonKADS notions of concept and relation correspond respectively to the UML notions of class and association (Schreiber et al., 2000). Consequently, this paper does not detail the "Define Classes" process, and focuses on the other three processes. It presents the transformations of the three processes, and illustrates systematic application to the company buyout example. The mapping transformations are presented using natural language. They may also be specified with the QVT language (query/view/transformation) associated with MDA (Object Management Group, 2011). For space reasons, QVT specification is shown for two transformations only.

5.2. Defining activities

Although PRR is central to the PIM viewpoint, the PRR model alone is insufficient to represent the behavior of a knowledge system, expressed in the CommonKADS knowledge model (CIM

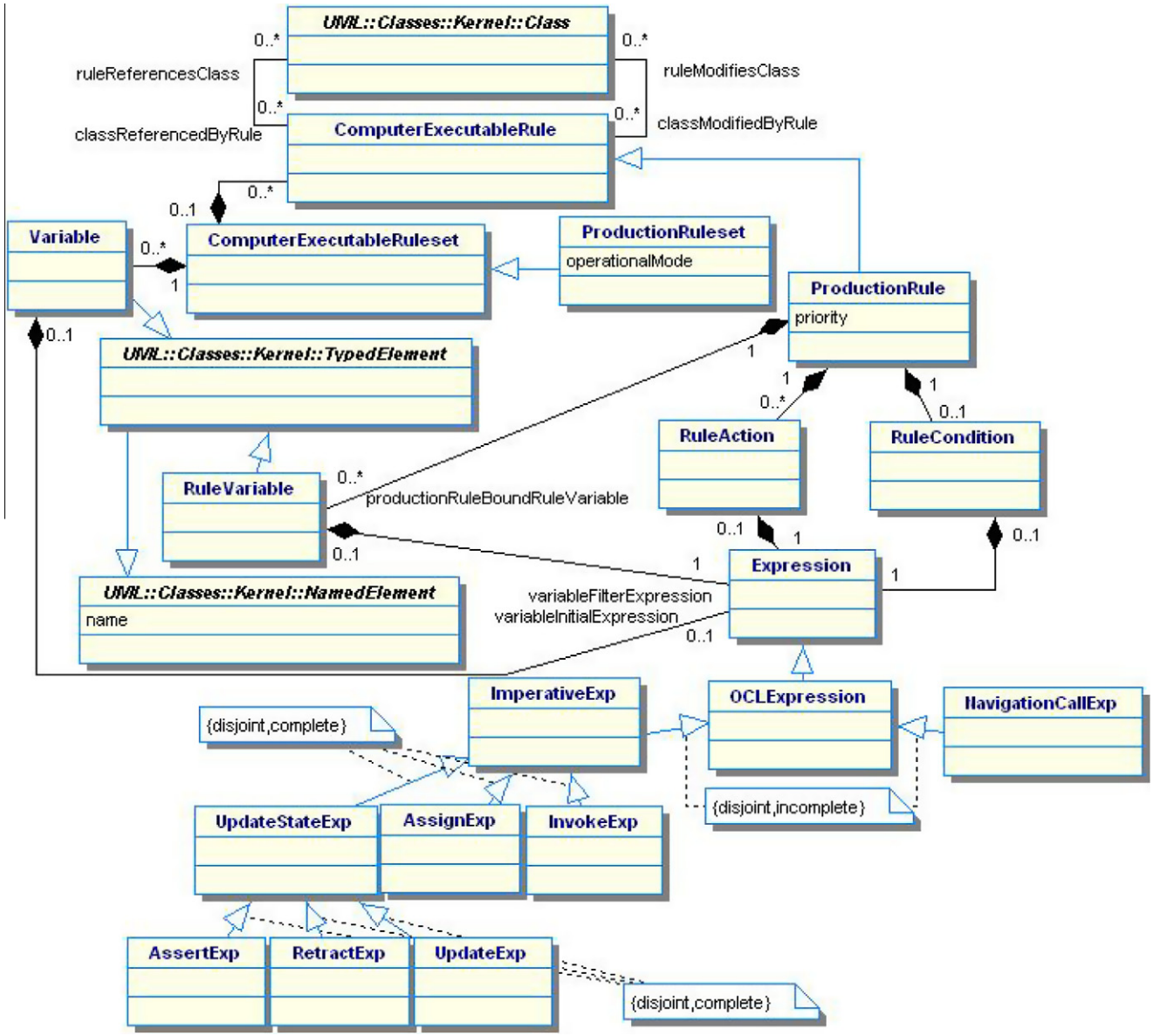

Fig. 5. Main concepts of the PRR metamodel (Object Management Group, 2009).

viewpoint). Therefore, our approach starts by mapping the CommonKADS knowledge model into a UML activity model. This activity model will be the starting point of the subsequent process ("Identify Inference Groups").

This paper assumes familiarity with UML2 activity models (for more details, the reader is referred to Object Management Group, 2010c). Taking advantage of UML extensibility, we specialize the UML concept of Action by defining the following stereotypes: ≪Inference≫, ≪Task≫, ≪ProductionRuleset≫, ≪Obtain≫, ≪Present≫, ≪Receive≫, and ≪Provide≫. The last four stereotypes correspond to the four types of transfer functions in CommonKADS. The stereotype ≪ProductionRuleset≫ will be used in the process "Define Production Rules and Rulesets".

CommonKADS represents the control structure of task methods with pseudo-code, and suggests UML activity diagrams as an alternative notation. This constitutes a good starting point for mapping CommonKADS knowledge models into UML activity models. However, at the time of publication of the CommonKADS methodology, UML activity diagrams lacked some concepts important in the representation of CommonKADS control structures. Since then (especially since UML2), the Object Management Group has significantly improved and extended the UML activity metamodel. Our approach uses these extensions (e.g. the concept of loop).

The process "Define Activities" consists in the following transformations. When a transformation applies to the company buyout example, we illustrate its application, and Fig. 9 presents the UML activity model resulting from all transformations.

Transformation A1: Each task method in the CommonKADS knowledge model becomes an activity in the UML activity model.

In the company buyout example, the task method assessment-method becomes a UML activity with the same name.

Transformation A2: Each input dynamic role of the task realized by the task method becomes an input parameter of the activity.

Fig. 8 expresses this transformation as a QVT relation. The source model (kad) is a CommonKADS knowledge model (instance of the CommonKADS knowledge metamodel). Similarly, the target model (act) is a UML activity model. Under the model names, C and E stand respectively for "checkonly" and "enforced" (Object Management Group, 2011). The "when" part indicates that transformation A2 may only follow transformation A1. The "where" part specifies precisely how the name of the activity parameter node is formed. The UML specification does not formally distinguish between input and output activity parameter nodes, but some tools (e.g. Visual Paradigm©) do. We also make this distinction.

In the company buyout example, the assessment task has one input dynamic role, called "case-description". This role maps to the concept of candidateBuyout in the CommonKADS domain schema. The input dynamic role becomes the input parameter "case-description: candidateBuyout" in the activity.

Transformation A3: Each output dynamic role of the task realized by the task method becomes an output parameter of the activity.

In the example, the assessment task has one output dynamic role, called "decision". This role maps to the concept of BuyoutDecision in the CommonKADS domain schema. The output dynamic role becomes the output parameter "decision: BuyoutDecision" in the activity.

Transformation A4: Each statement of the control structure of the task method is mapped through transformations A5, A8, A9, A10, A11 or A12, depending on the nature of the statement (function call, until loop, while loop, for loop, conditional statement, simple role assignment, role addition, and role deletion).

In the example, the control structure of the task method (Appendix B) consists in a function call (which will be mapped through transformation A5) followed by an "until" loop (mapped through transformation A8).

Transformation A5: Each function call in the control structure of the task method becomes an action of the activity. The stereotype of the action (≪Inference≫, ≪Obtain≫, ≪Present≫, ≪Receive≫, ≪Provide≫, or ≪Task≫) follows from the nature of the function called.

In our example, the control structure of the task method starts with a function call (instantiation of the inference "specify"). This function call becomes the action Specify (with stereotype ≪Inference≫) in the activity.

Transformation A5 will also be called recursively by transformation A8, when mapping the "repeat" part of the "until" loop. The actions Obtain (with stereotype ≪*Obtain*≫) and Evaluate (stereotype ≪*Inference*≫) will be defined then.

Transformation A6: Each input dynamic role of the function of transformation A5 becomes an input object pin of the action.

For the function call "specify", the inference "specify" has one input dynamic role ("case-description") in the CommonKADS knowledge model. This input dynamic role becomes the input pin "case-description: candidateBuyout" of the action Specify (following the same naming convention as in transformation A2). Similarly, the action Obtain has the input object pin "norms: Set of SelectionCriterion" and the action Evaluate has the input pins "norm: SelectionCriterion" and "case-description: candidateBuyout".

Transformation A7: Each output dynamic role of the function of transformation A5 becomes an output object pin of the action.

Fig. 9. shows the resulting output object pins for the example (the principle is the same as in transformation A6).

Transformation A8: Each ''until'' (resp. ''while'') loop in the control structure of the task method becomes a test-last (resp. test-first) loop in the activity. The ''repeat'' (resp. ''do'') part is mapped into the body part of the loop (inside this part, the statements are mapped recursively depending on their nature). The ''until'' (resp. ''while'') part is mapped into the test part of the loop.

In the example, there is an "until" loop in the control structure. This loop becomes a test-last loop in the activity (which we represent with the stereotype ≪*Until*≫). The "until" part is mapped into the test part of the loop. It consists in a "has-solution" condition. The three statements of the "repeat" part of the loop are recursively mapped.

Transformation A9: Each ''for'' loop in the control structure of the task method becomes an iterative expansion region in the activity. The ''for each'' part of the ''for'' loop becomes an input expansion node of the expansion region. Inside the ''do'' part, the statements are mapped recursively depending on their nature.

Transformation A10: Each conditional statement in the control structure of the task method is mapped by defining a decision node and a merge node in the activity. These nodes delimit the conditional statement in the activity. The decision node has one outgoing edge for the ''then'' part, and one for the ''else'' part. Inside the ''then'' part and the ''else'' part, the statements are mapped recursively depending on their nature.

Transformation A11: Each simple role assignment in the control structure of the task method becomes a ≪WriteVariable≫ action in the activity. The right hand side of the assignment becomes an input object pin of the action. The role referenced by the left hand side becomes a variable of the activity.

Transformation A12: Each role addition (resp. deletion) in the control structure of the task method becomes an ≪AddVariableValue≫ (resp. ≪RemoveVariableValue≫) action in the activity. The ''added'' (resp. ''deleted'') role of the right hand side of the assignment becomes an input object pin of the action. The role referenced by the left hand side becomes a variable of the activity.

In the example, there is no role addition or deletion contained directly in the control structure of the task method, but there is one role addition in the "repeat" part of the "until" loop. The left hand side of this role addition is the role evaluation-results. This role becomes a variable in the activity, and is used to name the action of role addition ("Add to evaluation-results"). The "added" role of the right hand side (norm-value) becomes an input object pin of the action, following the same naming conventions as in previous transformations.

Fig. 9 describes the UML activity model obtained in our example case. This figure is a faithful mapping of the task control structure represented in Appendix B. For the sake of simplicity, we allow multiple actions or structured nodes to connect to the same input parameter or output object pin. In our case, the input parameter is used both by the loop and by the action Specify. To avoid this, we could use a fork in the UML activity model, but this would make the model more complex.

5.3. Identifying inference groups

Before proceeding to the definition of production rules and rulesets, we need to group inferences and transfer functions (in the UML activity model) into what we will call "inference groups". The rationale for this is to prepare for the mapping of inferences into PRR production rulesets. Inferences in CommonKADS are often more fine-grained than PRR can represent with a single production ruleset. It is therefore advisable, when possible, to group inferences and then map them into a single PRR production ruleset. This way, the sequencing of the inferences inside the group will be represented inside the PRR production ruleset, instead of having to represent this sequencing with a complementary formalism (in our case, the UML activity model). Furthermore, "obtain" transfer functions can in certain cases be mapped using variables in PRR production rulesets. However, to this end, they need to be grouped with an inference (or inferences) following them. The resulting inference group will become a single production ruleset in PRR.

Our problem is similar to the issue addressed by Biton, Cohen-Boulakia, Davidson, and Hara (2008). These authors focus on the definition of user views in scientific workflows, in order to query and manage provenance. A user view is a grouping of modules in a workflow, making the management and querying of provenance easier. The authors describe the properties that user views should satisfy; they present an algorithm that takes as input a workflow specification, and automatically builds a user view with the required properties. Similarly to this paper, we define the properties that inference groups should satisfy, and present an algorithm that automatically identifies inference groups in each of the activities of the UML activity model.

5.3.1. Properties of inference groups

By analogy with Biton et al. (2008), we define an inference group as a "grouping of elements of a CommonKADS knowledge model, enabling an easier automatic rule mapping". These inference groups can be characterized by the following properties.

Property 1. *An inference group is a group of two or more consecutive actions within the same control structure (i.e. inside the same "linear group of actions").*

In our approach, we define a "linear group of actions" as one (or possibly more) action (s) not separated by a control structure. For example, actions separated by a decision node are in different control structures; actions in the setup part, the test part and the body part of an ≪Until≫ loop each belong to a different linear group of actions. Our concept of "linear group of actions" is a specialization of the concept of ActivityGroup defined in UML. Within a "linear group of actions", actions are linearly ordered. This follows from the fact that (1) these groups do not span across multiple control structures, and (2) we do not manage parallelism between actions (this could be the object of a future extension).

We require actions of an inference group to be within the same control structure since the idea of defining an inference group is to map this group into a single PRR production ruleset. Moreover, control structures may not be used to combine the production rules within a production ruleset (as described in Section 4, production rules in a production ruleset may only be executed sequentially or in forward-chaining mode).

Property 2. *The actions of an inference group may only be inferences or "obtains". An inference group consists in one (or a series of) inference(s), possibly preceded by one (or several) obtain(s).*

Inference groups may not contain tasks, since tasks are too high level to be mapped directly into PRR. A task needs further decomposition, as explained previously. In addition, inference groups do not integrate actions on roles (stereotyped actions ≪WriteVariable≫, ≪AddVariableValue≫ and ≪RemoveVariableValue≫), since these actions are an important part of the global control structure, and therefore we avoid merging them with other actions into an inference group.

An inference group needs to contain at least one inference since a PRR production ruleset, into which the inference group will be mapped, is a collection of rules. Moreover, in CommonKADS, it is through inferences that rules are executed. The inference group may start with one (or several) obtain(s), which we will map using the concept of variable at the beginning of the PRR production ruleset.

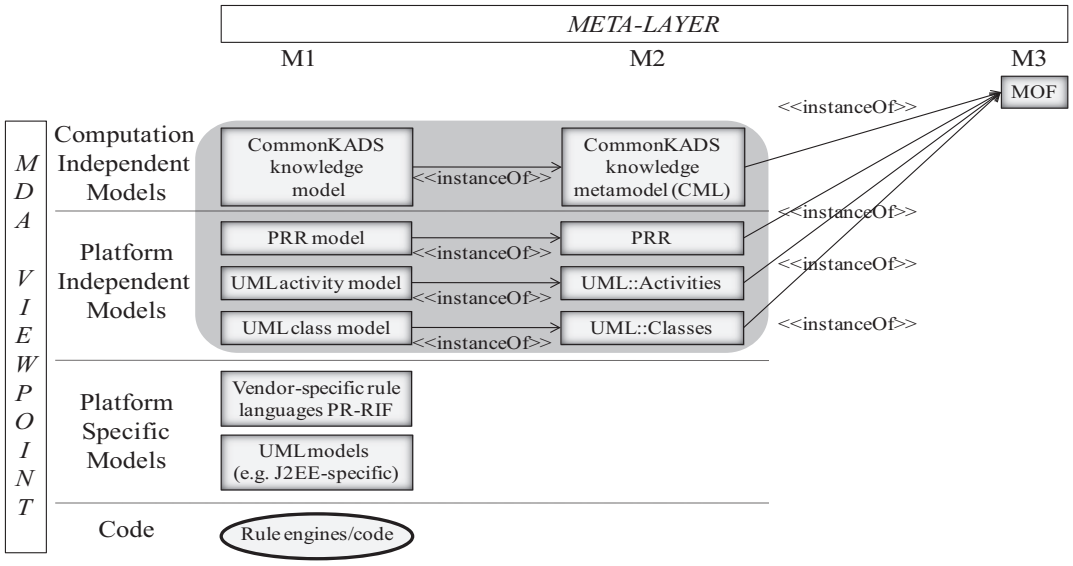

Fig. 6. MDA viewpoints and meta-layers.

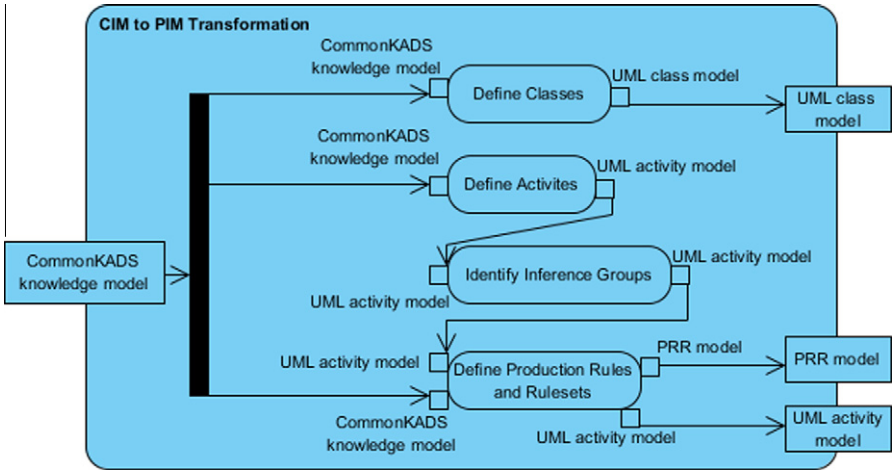

Fig. 7. CIM to PIM transformation.

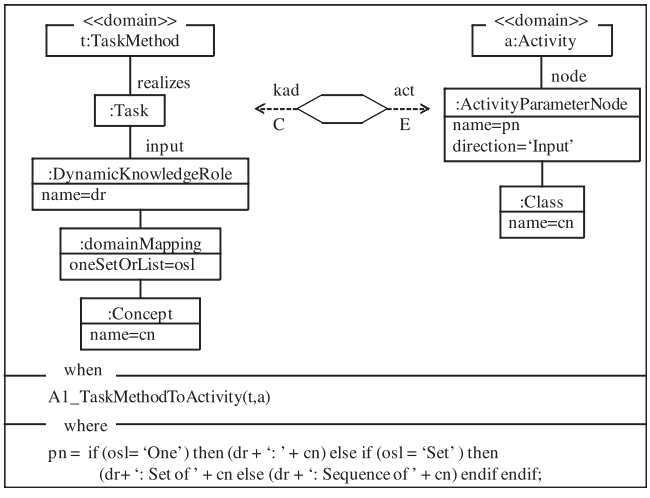

Fig. 8. QVT representation for transformation A2.

Property 3. *In an inference group, the outputs of "obtains" and of intermediary inferences are not used as inputs to actions or structured activity nodes outside the inference group.*

When "obtains" and inferences are merged into a single inference group, the output of the resulting inference group is the output of the last inference of the inference group; the other outputs are no longer represented explicitly, they will be internal to the inference group. Consequently, the outputs of an "obtain" or of an intermediary inference should not be needed outside the inference group.

Before presenting our algorithm for automatically defining inference groups satisfying the three properties above, we illustrate with the example of Fig. 10. In the latter, based on the control structure (decision node and merge node), four linear groups of actions are identified (the actions of these groups are represented with different colors). Within these linear groups of actions, there are two inference groups (their actions are surrounded). The first inference group stops at Inference2 since the next action is a task. Due to property 3, it is not possible to merge Inference3 and

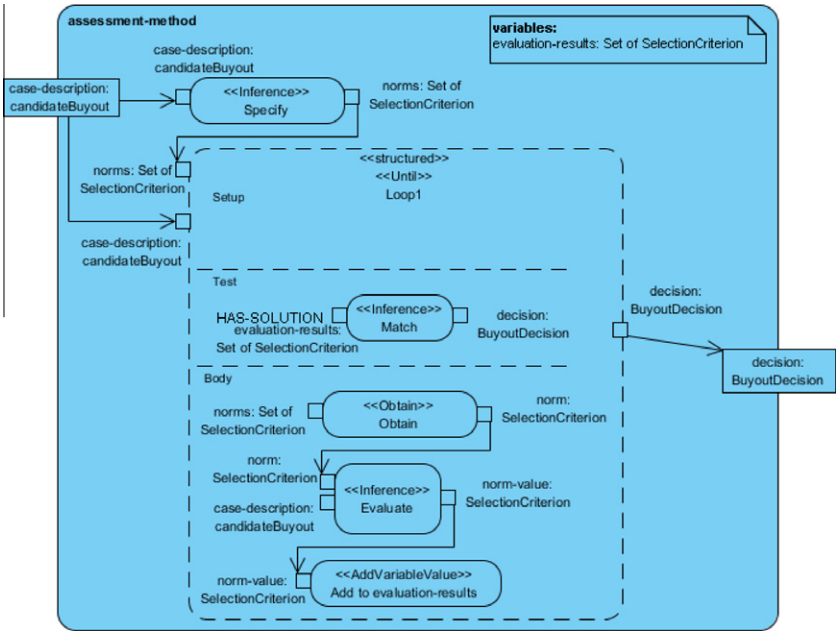

Fig. 9. Company buyout example: UML activity model resulting from the transformations of the "Define Activities" process.

Inference4 into an inference group (one output of Inference3, obj7, is used as input to Inference7).

5.3.2. The algorithm

We represent the algorithm with a pseudo-code notation. We could also represent it with QVT, but this would be at the expense of readability. To identify inference groups within an activity of the UML activity model, the algorithm proceeds in two steps:

1. Grouping of actions into linear groups of actions.
2. Identification of inference groups within each of the linear groups of actions.

In step 1, the identification of linear groups of actions proceeds by identifying the first action of each group, and then recursively following the links (activity edges) to find the next action, until the last action of the group is met. Initially, all actions of the activity are unmarked; actions are marked as they are incorporated into linear groups of actions.

An action is the first action of a linear group of actions if it is the target of an activity edge from a control node (e.g. a decision node), or if it is not the target (directly or through pins) of another action. Similarly, an action is the last action of a linear group of actions if it is the source of an activity edge to a control node (e.g. a merge node), or if it is not the source (directly or through pins) of another action. To find the next action of an action, the algorithm proceeds by following activity edges. Since an action may have several indirect successors, we need to find the immediate successor. To this end, we use the constraint that the next action to an action may not be the target of another action not already in the linear group of actions. Finally, in line with the definition of linear group of actions, we constrain an action and its immediate successor not to be separated by a control node (there should not exist a path of edges between an action and its next action such that this path contains a control node).

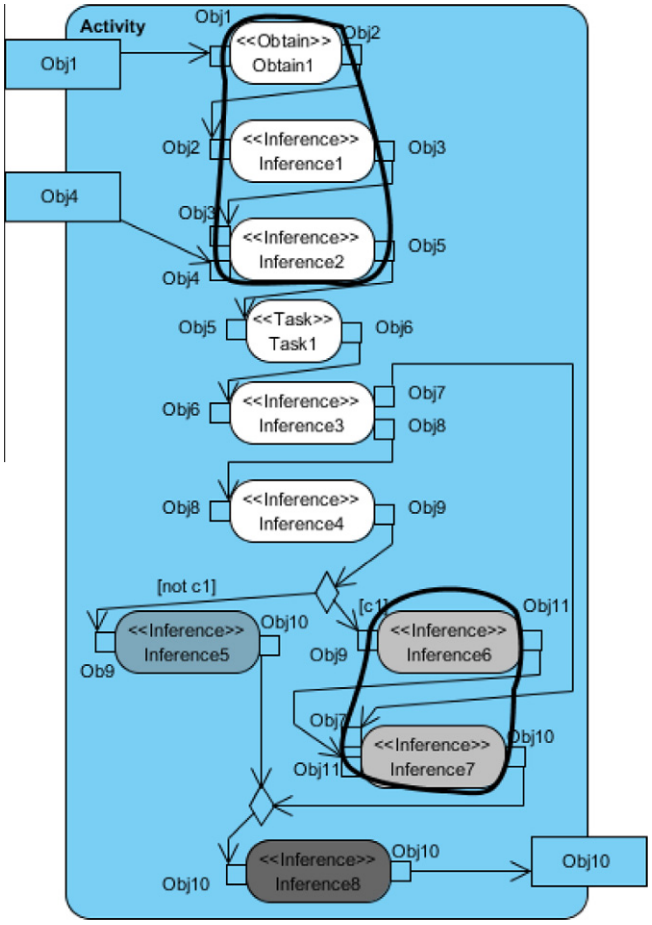

Fig. 10. Identifying inference groups: example.

```
– Step 1: Group actions into linear groups of
  actions.
for all al:Action do
  al.unmarked = true
end for
for all al:Action such that (al.unmarked = true) and
  ((∃ an ActivityEdge from a ControlNode to al) or (∄
  an ActivityEdge from an Action to al and ∄ an
  ActivityEdge from an OutputPin of an Action to an
  InputPin of al)) do
  gl:LinearGroupOfActions = new
  LinearGroupOfActions
gl.add (al)
currentAction = al
al.unmarked = false
while ∃ a2:Action such that (∃ an ActivityEdge from
  currentAction to
a2 or ∃ an ActivityEdge from an OutputPin of
  currentAction to an
InputPin of a2) and (∄ an ActivityEdge from an
  Action ∉ gl to a2) and
(∄ an ActivityEdge from an OutputPin of an Action ∉
  gl to an InputPin
of a2) and (∄ a path of ActivityEdge and
  ActivityNode from
currentAction to a2 such that this path contains a
  ControlNode) do
  gl.add (a2)
  currentAction = a2
  a2.unmarked = false
if (∃ an ActivityEdge from currentAction to a
  ControlNode) or (∄ an ActivityEdge
from currentAction to an Action and ∄ an
  ActivityEdge
from an OutputPin of currentAction to an InputPin of
  an Action) then
    break – from while loop
  end if
end while
end for
```

Step 2 identifies inference groups within each linear group of actions. To this end, the algorithm traverses the actions of linear groups of actions in reverse order. Each time it meets an inference, it identifies a candidate inference group. The preceding inferences are recursively incorporated into the candidate inference group (as long as Property 3 is satisfied – note that in UML terminology, ExecutableNode is a generic term for actions and structured activity nodes); the preceding obtains are then recursively incorporated into the candidate inference group (as long as Property 3 is satisfied). A candidate inference group becomes an inference group if it contains at least two actions.

```
– Step 2: Identify inference groups.
for all gl:LinearGroupOfActions do
  currentAction = gl.last ()
  while true do
    while currentAction <> ∅ and currentAction is
    not an Inference do
      currentAction = currentAction.predecessor ()
    end while
    if currentAction = ∅ then
      break– from while loop
    end if
    candidateInferenceGroup.add (currentAction)
  while (currentAction.predecessor () is an
    Inference) and (∄ an
  ActivityEdge from an OutputPin of
    currentAction.predecessor () to an
  InputPin of an ExecutableNode ∉
    candidateInferenceGroup) do
    currentAction = currentAction.predecessor ()
    candidateInferenceGroup.add (currentAction)
  end while
  while (currentAction.predecessor () is an Obtain)
    and (∄ an
  ActivityEdge from an OutputPin of
    currentAction.predecessor () to an
  InputPin of an ExecutableNode ∉
    candidateInferenceGroup) do
    currentAction = currentAction.predecessor ()
    candidateInferenceGroup.add (currentAction)
  end while
    if candidateInferenceGroup.size () > 1 then
      il:InferenceGroup = new InferenceGroup
      il = candidateInferenceGroup
    end if
  currentAction = currentAction.predecessor ()
  end while
  end for
```

Applying this algorithm to the company buyout example (activity model of Fig. 9.), we get three linear groups of actions. The first one contains the inference Specify. The second and third one correspond respectively to the test and body part of the ≪Until≫ loop node. Within these linear groups of actions, there is one inference group, which consists in Obtain and the inference Evaluate.

5.4. Defining PRR rules and rulesets

Based on the identification of inference groups, our approach then defines PRR rules and rulesets. We describe the transformations below and apply them to the example (we provide in Appendix A.3. examples of PRR rules and rulesets resulting from these transformations).

> **Transformation P1:** Each inference in the UML activity model becomes a production ruleset in the PRR model.

In the example, we get the production rulesets "specify" and "match", corresponding to the inferences (actions with stereotype ≪*Inference*≫) Specify and Match in the activity model.

> **Transformation P2:** Each inference group in the UML activity model becomes a production ruleset in the PRR model.

The inference group Evaluate (composed of the actions Obtain and Evaluate) is mapped into the production ruleset "evaluate".

Transformation P3: Each input object pin of each inference becomes a parameter of the corresponding production ruleset.

In the example, the input object pin of the inference Match is "evaluation-results: Set of SelectionCriterion". This input object pin becomes the parameter of the production ruleset "match", as shown in Appendix A.3. (For space reasons, Appendix A.3. does not show the specification of the production ruleset "specify". The paper focuses on the other production rulesets.)

Transformation P4: Each input object pin of each inference group becomes a parameter of the corresponding production ruleset. (The inputs of an inference group are the inputs coming from outside the inference group, i.e. not provided by actions inside the inference group.)

The inference group Evaluate is composed of the actions Obtain and Evaluate. The inputs of the inference group are the inputs coming from outside the inference group, i.e. "norms: set of SelectionCriterion" and "case-description: candidateBuyout" (refer to Fig. 9). These two input pins become the parameters of the production ruleset "evaluate", as shown in Appendix A.3.

Transformation P5: For each static knowledge role of each inference, the rule type instances of the implication rule type referenced by the static knowledge role become production rules in the production ruleset corresponding to the inference.

The inference Match has the static knowledge role "decision-knowledge" (as shown in Appendix B). This knowledge role references the implication rule type "buyout-decision-rule" (domain mapping). Consequently, the instances of this rule type (the last three rule type instances shown in Appendix A.1) become production rules in the production ruleset "match" (only the first production rule, "implies1", is shown in Appendix A.3).

Transformation P6: For each static knowledge role of each inference in each inference group, the rule type instances of the implication rule type referenced by the static knowledge role become production rules in the production ruleset corresponding to the inference group. (The static knowledge roles of the inference group are the union of the static knowledge roles of the inferences constituting the inference group.)

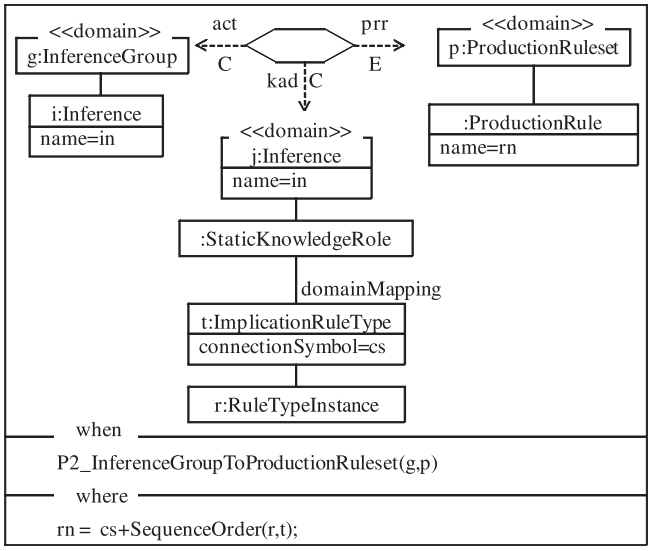

Fig. 11. QVT representation for transformation P6.

Fig. 11 details the QVT representation for transformation P6. This transformation uses two source models: the UML activity model (act) and the CommonKADS knowledge model (kad). The target is the PRR model. The "where" clause assumes that the function SequenceOrder has been defined. This function returns the sequence order of a rule type instance within the rule type of which it is an instance.

For the inference group Evaluate, the static knowledge role is "requirements", corresponding to the rule type "selection-requirement". The instances of this rule type (the first four rule type instances of Appendix A.1) become production rules of the production ruleset "evaluate" (only the first production rule, "indicates1", is shown).

Transformation P7: Each antecedent of the rule type (i.e. of the rule type of transformation P5 or P6), is mapped into at least one rule variable in each of the production rules.

The same antecedent may be mapped into several variables when this antecedent has specializations in the domain schema.

Fig. 2 points to the fact that the rule type "buyout-decision-rule" has one antecedent (SelectionCriterion). In rule "implies1", the antecedent of this rule type is mapped into the two rule variables ?financialHealth and ?marketSynergy, corresponding to the two specializations of SelectionCriterion in the domain schema. The rule type "selection-requirement" has the antecedent candidateBuyout. In rule "indicates1", the antecedent of this rule type is mapped into the variable ?candidateBuyout.

Transformation P8: Each consequent of the rule type (i.e. of the rule type of transformation P5 or P6), is mapped into zero, one or several rule variable(s) in each of the production rules. At least one rule variable is defined when the consequent is distinct from the antecedent.

The same consequent may be mapped into several variables when this consequent has specializations in the domain schema.

Fig. 2 indicates that the rule type "buyout-decision-rule" has one consequent (BuyoutDecision), distinct from the antecedent. In rule "implies1", this consequent is mapped into the variable ?buyoutDecision. The rule type "selection-requirement" has the consequent SelectionCriterion, distinct from the antecedent. In rule "indicates1", the consequent is mapped into ?financialHealth.

Transformation P9: For each antecedent of the rule type, there are at least as many navigation call expressions in each of the production rules as indicated by the cardinality between the antecedent and the rule type. These navigation call expressions may be in the filter of the rule variable (or variables) representing the antecedent, or in the rule condition of the production rule.

In our example, the condition of rule "implies1" contains two navigation call expressions, defined on the two rule variables representing the antecedent (?financialHealth and ?marketSynergy). Similarly, the condition of rule "indicates1" has two navigation call expressions, defined on the rule variable ?candidateBuyout representing the antecedent. This is consistent with the cardinality between the antecedent and the CommonKADS rule type, which is $1\ldots*$ for both rule types (as Fig. 2 indicates).

Transformation P10: For each consequent of the rule type, there are at least as many rule actions in each of the production rules as indicated by the cardinality between the consequent and the rule type. These rule actions are of the assign type; they modify the class referenced by the consequent, or one of its subclasses.

In our example, in rule "implies1", the action "?buyoutDecision.buyoutDecision="Buy"" assigns a new value to a property of ?buyoutDecision, representing the consequent. Similarly, in rule "indicates1", the action "?financialHealth.truthValue="true"" assigns a new value to a property of ?financialHealth. In each rule, there is only one assign action, in line with the cardinality between the consequent and the rule type, which is 1 for both rule types.

Transformation P11: For each input object pin of each inference, this object is used at least once in a rule variable filter expression in the production ruleset resulting from transformation P1.

The input object pin of the inference Match is "evaluation-results: Set of SelectionCriterion". This corresponds to the input parameter "evaluation-results" in the production ruleset "match". This parameter is then used (twice) in rule "implies1" in a rule variable filter expression.

Transformation P12: For each input object pin of each inference group, this object is used at least once (1) in a rule variable filter expression in the production ruleset resulting from transformation P2 or (2) in the initialization of a variable in the production ruleset resulting from transformation P2.

For the inference group Evaluate, the input object pins are "norms: Set of SelectionCriterion" and "case-description: candidateBuyout". This corresponds, respectively, to the input parameters "norms" and "case-description" in the production ruleset "evaluate". The parameter "norms" is used in the initialization of the variable "chosen-criterion"; the parameter "case-description" is used in rule "indicates1" in a rule variable filter expression.

Transformation P13: Each ''obtain'' transfer function of each inference group becomes a variable in the production ruleset resulting from transformation P2.

In our example, in the inference group Evaluate, the "obtain" is mapped into the variable "chosen-criterion", which will be initialized by prompting the user at execution time.

Transformation P14: The stereotype of each inference is changed to ≪ProductionRuleset≫.

In the example, the stereotype of actions Specify and Match, in the activity model, is changed from *≪Inference≫ to ≪ProductionRuleset≫*.

Transformation P15: For each inference group, the actions of the inference group are replaced by a single action, with stereotype ≪ProductionRuleset≫. The input object pins of this action are the inputs pins coming from outside the inference group, i.e. not provided by actions inside the inference group. The output object pins of this action are the outputs of the last inference of the inference group.

In the example, the inference group Evaluate includes the actions Obtain and Evaluate (stereotyped as *≪Obtain≫* and *≪Inference≫* respectively). These actions are replaced by the single action Evaluate, stereotyped as *≪ProductionRuleset≫*. The inputs pins of this action are the inputs coming from outside the inference group, i.e. "norms: set of SelectionCriterion" and "case-description: candidateBuyout". The output pin is "norm-value: SelectionCriterion" (output of inference Evaluate).

As a preliminary conclusion, let us sum up the results obtained so far. We have shown the feasibility of deriving PRR rules and rulesets starting from CommonKADS knowledge models. The main finding is based on the concept of inference group allowing us to define a common granularity level between the two models. Using this concept, we provided a set of transformations enabling to map systematically CommonKADS knowledge models into PRR. A first step of the validation process has been achieved by applying the approach to a simple example.

6. Conclusion

We conclude this paper by summarizing our main contributions and deriving implications for research and practice as well as further research opportunities.

6.1. Main contributions

This paper contributes to methodological support in knowledge engineering. More specifically, it provides support for mapping knowledge models into production rules, for subsequent mapping into expert systems. Our approach is based on MDA and bridges the gap between CommonKADS knowledge models (defined at the CIM level of MDA) and PRR production rules (defined at the PIM level).

The main limitation of this work lies in restricting the CIM level to the knowledge model, and focusing on the PRR model at the PIM level. The CIM level does not consider the communication model, which completes the knowledge model in CommonKADS. This paper assumes simple communication, which justifies leaving aside the communication model. More complex communication (e.g. in the case of multi-agent systems) would require considering also the communication model, as well as other transfer functions (besides the "obtain" transfer function considered in this paper). At the PIM level, our approach is PRR-driven: the UML class and activity models just serve as complements to the PRR model. Research should investigate other formalisms as alternatives to PRR, for example in cases where the target system for the knowledge model is not a rule-based system.

Having acknowledged these limitations, the paper contributes knowledge engineering in the following ways:

1. It proposes an instantiation of the three MDA levels applied to knowledge engineering, elaborating on previous research that has combined MDA with CommonKADS or rule languages. Our approach focuses on the CIM and PIM levels, the most crucial levels in MDA. It situates the CommonKADS knowledge model at the CIM level of MDA. We target the knowledge systems whose implementation relies on production rules. PRR represents this knowledge at the PIM level.
2. It proposes a complete UML metamodel to represent CommonKADS knowledge models and enable their subsequent mapping into PRR production rules and rulesets. CommonKADS knowledge models revolve around the central concepts of domain knowledge, inference knowledge and task knowledge. For our MDA approach to knowledge engineering, we needed to represent precisely the concepts of CommonKADS knowledge models, but a complete UML metamodel for CommonKADS knowledge models was missing from the literature.
3. The mapping between the CIM and PIM levels of MDA requires the definition of model transformation rules. To this end, the paper defines the concept of inference group as a logical set of CommonKADS inferences. The role of these inference groups is to facilitate the mapping between CommonKADS inferences and PRR rulesets. The paper defines the properties that inference groups should possess, and proposes an algorithm that automatically defines inference groups conforming to these properties. Grouping inferences into inference groups prepares CommonKADS knowledge models for mapping into PRR production rules and rulesets. The paper proposes a set of transformations that enable developers to perform semi-automatic mapping of CommonKADS knowledge models into PRR specifications. The transformations complete the definition of PRR rules and rulesets with UML activity diagrams, thus enabling a richer mapping of the concepts of CommonKADS knowledge models. The paper describes and discusses an example, applying the whole approach.

6.2. Implications for research and practice

This work contributes to research in the field of knowledge engineering, by bridging the gap between two standards: CommonKADS and PRR. As literature and practice suggest, CommonKADS is a *de facto* standard among knowledge engineering methods; PRR is the recommended OMG standard for the representation of production rules. As a byproduct of our approach, this work also contributes to the evolution of CommonKADS, by presenting how to represent the control structure of CommonKADS task methods with UML 2 activity diagrams. Although from the origin, CommonKADS has proposed UML activity diagrams, as a means to represent the control structure of task methods, the way to achieve this was unclear. With the new concepts of UML 2 activity diagrams, it becomes much easier to represent the control structure of CommonKADS task methods, and this paper shows how to achieve this.

Practitioners often report a lack of methodological support for the development of rule-based systems. This paper contributes to increased methodological support, easing the mapping between knowledge models and production rules. The fact that this work relies on MDA and other recognized standards (e.g. UML) should facilitate adoption by practitioners, as well as tool support.

6.3. Further research opportunities

Among the possible further extensions and refinements of this work, let us mention the following:

- The mapping transformations defining PRR production rules and rulesets could be refined by applications to other scenarios, besides the example used in this paper. In particular, the transformations could distinguish between the different operational modes (sequential and inferencing), and make use of the PRR concept of rule priority.
- The mapping transformations defining activities could also be refined, e.g. concerning the mapping of "until", "while" and "for" loops.
- Further research could consider other models at the CIM and PIM levels, and define new transformations. More specifically, considering the CommonKADS communication model as a complement to the knowledge model would be useful for multi-agent systems.
- Mapping PRR rules and rulesets into CommonKADS knowledge models (reverse engineering) constitutes another promising research direction.
- Finally, UML activity models could complete the PRR standard, more particularly to facilitate orchestration between different PRR rulesets. The Object Management Group (2009) mentions this research direction but does not specify how UML activity models could complement PRR. In this paper, we use UML activity models as a complement to PRR in order to enable more

complete mapping of the concepts of CommonKADS knowledge models. Therefore, our work constitutes a good starting point for augmenting the PRR standard with UML activity models.

Acknowledgements

The authors would like to thank William Starbuck for his helpful comments on an earlier version of this paper.

Appendix A. Company buyout example: knowledge base, knowledge roles, and examples of PRR rules and rulesets

A.1. Knowledge base

```
KNOWLEDGE-BASE measurement-system;
  USES:
  selection-requirement FROM buyout-schema,
  buyout-decision-rule FROM buyout-schema;
  EXPRESSIONS:
  candidateBuyout.target.netProfitMargin > 0.15 AND
  candidateBuyout.target.returnOnEquity > 0.33
    INDICATES
  FinancialHealth.truthValue = true;
  candidateBuyout.target.netProfitMargin < = 0.15 OR
  candidateBuyout.target.returnOnEquity < = 0.33
    INDICATES
  FinancialHealth.truthValue = false;
  candidateBuyout.target.mainRegionalMarket! =
  candidateBuyout.acquirer.mainRegionalMarket
    INDICATES
  MarketSynergy.truthValue = true;
  candidateBuyout.target.mainRegionalMarket =
  candidateBuyout.acquirer.mainRegionalMarket
    INDICATES
  MarketSynergy.truthValue = false;
  FinancialHealth.truthValue = true AND
  MarketSynergy.truthValue = true
    IMPLIES
  BuyoutDecision.buyoutDecision = 'Buy!';
  FinancialHealth.truthValue = false
    IMPLIES
  BuyoutDecision.buyoutDecision = 'Do not buy!';
  MarketSynergy.truthValue = false
    IMPLIES
  BuyoutDecision.buyoutDecision = 'Do not buy!';
END KNOWLEDGE-BASE measurement-system;
```

A.2. Knowledge roles

Knowledge roles

KNOWLEDGE ROLE	TYPE	DOMAIN MAPPING
case-description	DYNAMIC	candidateBuyout
decision	DYNAMIC	BuyoutDecision
norm	DYNAMIC	SelectionCriterion
norm-value	DYNAMIC	SelectionCriterion
norms	DYNAMIC	SET-OF SelectionCriterion
evaluation-results	DYNAMIC	SET-OF SelectionCriterion
requirements	STATIC	selection-requirement FROM measurement-system
decision-knowledge	STATIC	buyout-decision-rule FROM measurement-system

A.3. Examples of PRR rules and rulesets

```
Ruleset match (in evaluation-results:Set
  (SelectionCriterion))
Rule implies1
RuleVariable:
?financialHealth = evaluation-results → any
  (c1:SelectionCriterion|
    c1.OCLIsTypeOf (FinancialHealth))
?marketSynergy = evaluation-results → any
  (c2:SelectionCriterion|
    c2.OCLIsTypeOf (MarketSynergy))
  ?buyoutDecision = BuyoutDecision → any ()
Condition:
  ?financialHealth.truthvalue = true and
  ?marketSynergy.truthValue = true
Action:
  ?buyoutDecision.buyoutDecision= ''Buy''
...
Ruleset evaluate (in norms:Set
  (SelectionCriterion), case-
  description:candidateBuyout)
Variable:
chosen-criterion: SelectionCriterion
—This variable stores the criterion chosen by the
  —user. Upon implementation, the
—variable will be initialized with the appropriate
  —function, prompting the user to
—choose a criterion among the selection criteria.
  —(The selection criteria are
—provided through the first parameter of the
  —production ruleset).
Rule indicates1
RuleVariable:
?candidateBuyout = candidateBuyout → any
  (c:candidateBuyout|c = case-description)
?financialHealth = FinancialHealth → any
  (f:FinancialHealth|
    f.OCLAsType (SelectionCriterion)=chosen-
  criterion. OCLAsType (SelectionCriterion))
Condition:
  ?candidateBuyout.netProfitMargin > 0.15
  and
  ?candidateBuyout.returnOnEquity > 0.33
Action:
  ?financialHealth.truthValue = true
...
```

Appendix B. Task knowledge and inferences for the assessment task (adapted from Schreiber et al., 2000)

```
TASK assessment;
  ROLES:
    INPUT: case-description: "the case to be assessed";
    OUTPUT: decision: "the result of assessing the case";
END TASK assessment;
TASK-METHOD assessment-method;
  REALIZES: assessment:
  DECOMPOSITION:
    INFERENCES: specify, evaluate, match;
    TRANSFER-FUNCTIONS: obtain;
  ROLES:
    INTERMEDIATE:
      norms: "the full set of assessment norms";
      norm: "a single assessment norm";
      norm-value: "truth value of a norm for this case";
      evaluation-results: "list of evaluated norms";
    CONTROL-STRUCTURE:
      specify (case-description → norms);
      REPEAT
        obtain (norms → norm);
        evaluate (case-description, norm → norm-value);
        evaluation-results:= norm-value ADD evaluation-
  results;
      UNTIL
        HAS-SOLUTION match (evaluation-results →
  decision);
      END REPEAT
END TASK-METHOD assessment-method;
INFERENCE specify;
  ROLES:
    INPUT: case-description;
    OUTPUT: norms;
  SPECIFICATION:
    "Specifies the norms applicable to the case description.
    The way this specification is performed is left open at the
    analysis level.
    (The inference has no static knowledge role)."
END INFERENCE specify;
INFERENCE evaluate;
  ROLES:
    INPUT: norm, case-description;
    OUTPUT: norm-value;
    STATIC: requirements;
  SPECIFICATION:
    "Establishes the truth value of the norm for the case
    description."
END INFERENCE evaluate;
INFERENCE match;
  ROLES:
    INPUT: evaluation-results;
    OUTPUT: decision;
    STATIC: decision-knowledge;
  SPECIFICATION:
    "Makes a final decision based on the evaluation results."
END INFERENCE match;
```

References

Abdullah, M., Benest, I., Paige, R., & Kimble, C. (2007). Using unified modeling language for conceptual modeling of knowledge-based systems. In *Proceedings of the 26th international conference on conceptual modeling (ER 2007), Auckland, New Zealand, November* (pp. 438–453).

Andrade, J., Ares, J., García, R., Rodríguez, S., & Suárez, S. (2010). A knowledge-based system for knowledge management capability assessment model evaluation. *WSEAS Transactions on Computers, 9*(5), 506–515.

Anya, O., Tawfik, H. Amin, S., Nagar, A. & Shaalan, K. (2010). Context-aware knowledge modelling for decision support in e-health. In *Proceedings of the 2010 international joint conference on neural networks (IJCNN), Barcelona, Spain, July* (pp. 1–7).

Arguello, M., Des, J., Fernandez-Prieto, M. J., Perez, R., & Paniagua, H. (2008). Enabling reasoning on the Web: Introducing a test-bed simulation framework. In *Proceedings of the second UKSIM European symposium on computer modeling and simulation, Liverpool, England, September* (pp. 469–475).

Baumeister, J., Reutelshoefer, J., & Puppe, F. (2011). KnowWE: A semantic Wiki for knowledge engineering. *Applied Intelligence, 35*(3), 323–344.

Bera, P., Nevo, D., & Wand, Y. (2005). Special theme of research in information systems analysis and design – I. Unraveling knowledge requirements through business process analysis. *Communications of the Association for Information Systems* 16, Article 41, 814–830.

Biton, O., Cohen-Boulakia, S., Davidson, S. B., & Hara, C. S. (2008). Querying and managing provenance through user views in scientific workflows. In *Proceedings of the 24th international conference on data engineering (ICDE 2008), Cancun, Mexico, April* (pp. 1072–1081).

Cabot, J., Pau, R., & Ravento, R. (2010). From UML/OCL to SBVR specifications: A challenging transformation. *Information Systems, 35*(4), 417–440.

Cañadas, J., Palma, J., & Tùnez, S. (2005). From knowledge level to symbolic level. A metamodeling approach. In *Proceedings of the workshop on knowledge engineering and software engineering (KESE 2005), Koblenz, Germany, September* (pp. 13–24).

de Sainte Marie, C., & Taylor, J. (2007). Production rule standards – OMG PRR presentation. <http://www.edmblog.com/weblog/PRR at OMG Brusselsv3.pdf>.

Diouf, M., Maabout, S., & Musumbu, K. (2007). Mergin model driven architecture and semantic web for business rules generation. In *Proceedings of RR 2007, LNCS* (Vol. 4524, pp. 118–132).

Fox, J. (2011). Formalizing knowledge and expertise: where have we been and where are we going? *The Knowledge Engineering Review, 26*(1), 5–10.

Gartner, Inc., 2010. Hype cycle for application development, 2010, publication number G00205120.

Hasan, S. S., & Isaac, R. K. (2011). An integrated approach of MAS-CommonKADS, Model–View–Controller and web application optimization strategies for web-based expert system development. *Expert Systems with Applications, 38*(1), 417–428.

Kingston, J. K. C. (1998). Designing knowledge based systems: The CommonKADS design model. *Knowledge-Based Systems, 11*(5-6), 311–319.

Lukichev, S., & Wagner, G. (2006). UML-based rule modeling with Fujaba. In H. Giese & B. Westfechtel (Eds.), *Proceedings of the 4th international Fujaba Days 2006* (pp. 31–35). Germany: University of Bayreuth.

Martínez-Fernández, J., Martínez, P., & González-Cristóbal, J. (2009). Towards an improvement of software development processes through standard business rules. In *Proceedings of the international symposium on rule interchange and applications (RuleML 2009), Las Vegas, NV, USA, November* (pp. 159–166).

Mazon, J. N., & Trujillo, J. (2008). An MDA approach for the development of data warehouses. *Decision Support Systems, 45*(1), 41–58.

Milanovic, M., Gaševi, D., Giurca, A., Wagner, G., Lukichev, S., & Devedži, V. (2009). Model transformations to bridge concrete and abstract syntax of web rule languages. *ComSIS, 6*(48), 2.

Nabil, D., El-Korany, A., & Sharaf Eldin, A. (2008). Towards a suite of quality metrics for KADS-domain knowledge. *Expert Systems with Applications, 35*(3), 654–660.

Object Management Group. (2003). MDA Guide version 1.0.1., document number omg/03-06-01. <http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf>.

Object Management Group. (2009). Production rule representation (PRR) version 1.0 specification, document number formal/2009-12-01. <http://www.omg.org/spec/PRR/1.0/PDF>.

Object Management Group. (2010a). Object constraint language (OCL) specification, document number formal/2010-02-01. <http://www.omg.org/spec/OCL/2.2/PDF>.

Object Management Group. (2010b). Unified modeling language (UML) version 2.3. Infrastructure specification, document number formal/2010-05-03. <http://www.omg.org/spec/UML/2.3/Infrastructure/PDF>.

Object Management Group. (2010c). Unified Modeling Language (UML) version 2.3. Superstructure specification, document number formal/2010-05-05. <http://www.omg.org/spec/UML/2.3/Superstructure/PDF>.

Object Management Group. (2011). Query/View/Transformation (QVT) version 1.1 specification, document number formal/2011-01-01. <http://www.omg.org/spec/QVT/1.1/PDF/>.

Prat, N., Akoka, J., & Comyn-Wattiau, I. (2006). A UML-based data warehouse design method. *Decision Support Systems, 42*(3), 1449–1473.

Prat, N., Comyn-Wattiau, I., & Akoka, J. (2011). Combining objects with rules to represent aggregation knowledge in data warehouse and OLAP systems. *Data & Knowledge Engineering, 70*(8), 732–752.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., et al. (2000). *Knowledge engineering and management: The CommonKADS methodology*. Cambridge, MA, USA: MIT Press.

Sigut, J., Piñeiro, J., González, E., & Torres, J. (2007). An expert system for supervised classifier design: Application to Alzheimer diagnosis. *Expert Systems with Applications, 32*(3), 927–938.

Sutton, D., & Patkar, V. (2009). CommonKADS analysis and description of a knowledge based system for the assessment of breast cancer. *Expert Systems with Applications, 36*(2), 2411–2423.

Wagner, G., Antoniou, G., Tabet, S., & Boley, H. (2004). The abstract syntax of RuleML – towards a general Web rule language framework, National Research Council, Canada. <http://www.nparc.cisti-icist.nrc-cnrc.gc.ca/npsi/ctrl?action=rtdoc&an=891415&lang=en>.

Zacharias, V. (2008). Development and verification of rule based systems – A survey of developers. In *Proceedings of the international symposium on rule representation, interchange and reasoning on the web (RuleML 2008), Orlando, FL, USA, October* (pp. 6–16).

Zacharias, V. (2009). The agile development of rule bases. In W. Wojtkowski, G. Wojtkowski, M. Lang, K. Conboy, & C. Barris (Eds.). *Information systems development – challenges in practice, theory and education* (Vol. 1, pp. 93–104). New-York, NY, USA: Springer.

Zhu, L. (2006). Model-driven architecture. In I. Gorton (Ed.), *Essential software architecture* (pp. 197–215). Berlin/Heidelberg, Germany: Springer.

Zur Muehlen, M., & Indulska, M. (2010). Modeling languages for business processes and business rules: a representational analysis. *Information Systems, 35*(4), 379–390.