Mapping CommonKADS Knowledge Models into PRR

Nicolas Prat
ESSEC Business School
Paris, France
prat@essec.edu

Jacky Akoka
CEDRIC-CNAM & Institut Telecom
/ TEM Research
Paris, France
akoka@cnam.fr

Isabelle Comyn-Wattiau
CEDRIC-CNAM & ESSEC Business School
Paris, France
wattiau@cnam.fr

Abstract—This paper aims at supporting the knowledge engineering process by proposing an approach to map CommonKADS knowledge models into specifications based on the Production Rule Representation (PRR) language. This approach starts by proposing a metamodel of CommonKADS knowledge models. We define the concept of inference group, required to perform the mapping transformations, and an algorithm that identifies inference groups automatically. We then proceed to the definition of transformation rules. The latter are applied to map CommonKADS knowledge models into a set of PRR production rulesets, combined with UML activity diagrams.

Keywords – CommonKADS, Model Driven Approach, Production Rule Representation, metamodel, mapping

I. INTRODUCTION

It is well accepted that the cross-fertilization between software engineering and knowledge engineering can significantly improve the design of knowledge-based and expert systems. The aim of this paper is to present an approach enabling the mapping of CommonKADS knowledge models into specifications based on the Production Rule Representation language (PRR). The following aspects mainly motivate our work:

- The CommonKADS methodology is a widely used knowledge engineering method. Even if CommonKADS is more generally suited to systems where knowledge plays an important role, it does not necessarily lead to the development of expert systems. It focuses more on knowledge engineering issues. CommonKADS knowledge models are therefore specified at the analysis level. It can be considered as the Computational Independent Model (CIM) level of the Model Driven Approach (MDA [8]).

- On the other hand, PRR represents a relatively recent standardization effort conducted by OMG [10]. PRR is situated at the Platform Independent Model (PIM) level of MDA. We assume that the knowledge system will be implemented as production rules. PRR represents this knowledge in the design phase.

- There exists a possible convergence between CommonKADS knowledge models and PRR, since both are based on UML. However, the two are not completely related. Therefore, a mapping of CommonKADS knowledge models into PRR is an open research problem. We argue that such a mapping can be very useful.

To perform such a mapping, we propose an algorithm aiming at the definition of “inferences groups” as well as several transformation rules.

The rest of the paper is organized as follows. Section II is devoted to a presentation of related work. We describe in Section III the main concepts of the PRR metamodel. Our CommonKADS metamodel is presented in Section IV. The algorithm and the transformations used to map CommonKADS knowledge models into PRR are described in Section V. Finally, Section VI is devoted to the conclusion and future research.

II. RELATED WORK

CommonKADS [13] is a widely referenced methodology for designing knowledge-based systems [2,5,7,14,15]. As mentioned in the previous section, there exists a possible convergence between CommonKADS knowledge models and PRR, since both are based on UML. However, the two are not completely related. [1] offers an approach enabling a mapping between the knowledge model and JESS, based on a UML profile. However, the knowledge model presented in their approach is less detailed than the one presented in this paper. Moreover, this mapping is situated at the Platform Specific Model (PSM) level of MDA. Finally, let us mention that the description of the design level in CommonKADS [13] is relatively limited, especially compared to the richness of its analysis level as well as its knowledge model. The design level consists mainly in the “direct” mapping of concepts of the analysis level (including knowledge model concepts) as design classes. This approach does not help much the designer, hence the interest of a bridge with PRR in order to switch to the design level. [16] proposes to combine OWL, SWRL and JESS to build a knowledge level modeling. [17] describes and compares several rule languages with the objective of modeling
business processes. [4] is a recent and detailed state-of-the-art on rule-based systems.

III. MAIN CONCEPTS OF THE PRR METAMODEL

The Production Rule Representation language (PRR) [10] has been proposed by OMG for high-level (tool-independent) representation of rules. Fig. 1 is an excerpt of the metamodel of PRR, with the main concepts relevant in this research.

Production rules are the central concept of PRR. Production rules may reference and modify classes (possibly defined in a UML [11] class diagram). Rules are grouped into rulesets. A 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, rule execution order is determined by the sequence of rules in the ruleset. In the inferencing mode, the inference engine controls rule execution order. PRR currently supports forward chaining only, and suggests RETE [3] as a possible algorithm. A PRR rule may have a priority. Priorities are used 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 be defined at the ruleset level or at the rule level. Rule variables are used for binding.

PRR rules can be represented formally, based on an extension of the Object Constraint Language (OCL) [9]. Imperative expressions are used to represent rules actions. Navigation call expressions are used to navigate within or between objects.

PRR is a powerful language for expressing rules, as illustrated in [12] in the domain of data warehousing and OLAP.

IV. MAIN CONCEPTS OF COMMONKADS KNOWLEDGE MODEL

CommonKADS knowledge models can be specified diagrammatically, and textually using the CommonKADS Modeling Language (CML). CML is a structured, semi-formal language. In [13], the syntax of CML is described, but the authors do not provide a metamodel of the CommonKADS knowledge model. Metamodels of the CommonKADS knowledge model have been proposed in the literature, but they are too succinct to be useful in our approach [1,6]. Our metamodel, defined as a UML class diagram, is presented in Fig. 2. It describes the main concepts of the CommonKADS knowledge model.

A knowledge model has three components: domain knowledge, inference knowledge and task knowledge. Domain knowledge represents the structure of the knowledge system, while inference knowledge 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. For example, the assessment task may be used in several domains. The mapping of inference and task knowledge with domain knowledge is performed through the concept of knowledge role.

The structure of domain knowledge is described in a domain schema. This schema is represented as a UML class diagram. 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 (respectively the consequent) indicate the minimum and maximum number of expressions of the antecedent class (respectively the consequent class) in an instance of the rule type. Besides implication rule types, constraint rule types may be defined. Constraint rule types are internal to a concept. They can be considered as integrity constraints defined on this concept.
The behavior of the knowledge system is represented with task knowledge and inference knowledge. Tasks, task methods, inferences and transfer functions are related through functional decomposition. 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 represented as a black box. Transfer functions are used to communicate with the external world (information flows between the system and the user, at the initiative of the system or of the user). The transfer function “obtain”, whereby the system asks the user to enter an information item, is frequently used. In this paper, we will focus on this transfer function.

Knowledge roles will be used to 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. A static knowledge role will generally map to an implication rule type in the domain schema.

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.

![Figure 2. Metamodel of the CommonKADS knowledge model](image-url)
V. MAPPING COMMONKADS KNOWLEDGE MODELS INTO PRR

A. Overview of our Approach

A specificity of our approach is to use PRR to represent knowledge in the design phase. Thus, our approach consists in mapping CommonKADS knowledge models into PRR. We assume that the knowledge system will be implemented as production rules in an expert system. In this context, the choice of PRR is appropriate.

Our approach is divided into two steps:

1. Before applying the mapping transformations, groups of inferences and transfer functions (more particularly, “obtain” transfer functions), are defined in the knowledge model. To this end, we propose an algorithm for defining such groups, called “inference groups”. This algorithm operates on the control structure of task methods, represented as UML activity diagrams.

2. The transformations are then applied to map the knowledge model to the design level. The result is a set of PRR production rule sets, combined with UML activity diagrams.

B. Inference Groups

Before applying the mapping transformations, we need to group inferences and transfer functions (more specifically “obtain” transfer functions, on which we focus in this paper) 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, UML activity diagrams). 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 be mapped into a single production ruleset in PRR. We present below the properties that must be satisfied by inference groups.

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 different linear groups of actions. Our concept of “linear group of actions” is a specialization of the concept of ActivityGroup defined in UML [11]. 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 research).

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, and control structures may not be used to combine the production rules within a production ruleset (as described in Section III, 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 of one (or a series of) inference(s), possibly preceded by one (or several) obtains.

Inference groups may not contain tasks, because tasks are too high level to be mapped directly into PRR. In CommonKADS, a task needs to be further decomposed, as explained previously. In addition, inference groups do not integrate actions on roles (stereotyped actions "<<WriteVariable>>", "<<AddVariableValue>>" and "<<RemoveVariableValue>>"), since these operations on variables (roles) are an important part of the global control structure of a task method, 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; and in CommonKADS, it is through inferences that rules are executed. The inference group may start with one (or several) obtain(s), which will be mapped using the concept of variable at the beginning of the PRR production ruleset.

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 will be merged into a single inference group, the output of the resulting inference group will be the output of the last inference of the inference group; the other outputs will no longer be represented specifically, 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.

C. The Inference-Group Building Algorithm

To identify inference groups within the activity diagram representing the control structure of a task method, 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 diagram 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 following action of a given action, activity edges are followed. Since an action may have several indirect successors, we need to find the immediate successor. To this end, we use the constraint that the following action of an action may not be the target of another action that is not already present 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).

In step 2, inference groups are identified within each linear group of actions. This is done by traversing the actions of linear groups of actions in reverse order. Each time an inference is met, a candidate inference group is identified. The preceding inferences are recursively incorporated into the candidate inference group (as long as property 3 is satisfied); the preceding obtains are then recursively incorporated into the candidate inference group (as long as property 3 is satisfied). A candidate inference group is defined as an inference group if it contains at least two actions.

D. The transformations

Once the inference groups have been identified, the knowledge model is mapped to the design level, using the transformations described below. The result is a set of PRR production rulesets, combined with one UML activity diagram for each task method.

At the design level, we replace dynamic and intermediate roles by their domain mapping. This is at the expense of reusability, but reusability in CommonKADS is performed primarily at the analysis level through the reuse of template knowledge models. Furthermore, domain mapping makes activity diagrams more readable. We also simplify activity diagrams at the design level by keeping only the variables on which actions are performed (<<WriteVariable>>, <<AddVariableValue>> or <<RemoveVariableValue>>).

The transformations described below focus on the non-immediate mappings between the analysis level and the design level. (For example, implicitly, a concept, defined as a UML class in the domain schema, is mapped into a class at the design level; an action <<AddVariableValue>> in an analysis activity diagram is mapped into the same action in the design activity diagram, etc…)

**Transformation T1**: An inference or an inference group is mapped into an action, represented internally as a production ruleset.

**Transformation T2**: The input and output dynamic roles of an inference or an inference group are mapped into input and output parameters (pins) of the action resulting from transformation T1.

**Transformation T3**: The input dynamic roles of an inference or an inference group are mapped into parameters of the production ruleset resulting from transformation T1.

**Transformation T4**: For each static knowledge role of an inference or an inference group, the rule type instances of the implication rule type referenced by the static knowledge role are mapped into production rules in the production ruleset resulting from transformation T1.

**Transformation T4.1**: Each antecedent or consequent of the rule type (i.e. of the rule type of transformation T4), is mapped into at least one rule variable in each of the production rules.

**Transformation T4.2**: 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.

**Transformation T4.3**: 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.

**Transformation T4.4**: For each consequent of the rule type, an action of the assert type is created in each of the production rules to record the fact that the production rule has modified the consequent.
Transformation T5: For each input dynamic role of an inference, the class referenced by the role is used at least once in a rule variable filter expression in the production rule set resulting from transformation T1.

Transformation T6: For each input dynamic role of an inference group, the class referenced by the role is used at least once (1) in a rule variable filter expression in the production rule set resulting from transformation T1 or (2) in the initialization of a variable in the production rule set resulting from transformation T1.

Transformation T7: Each “obtain” transfer function of an inference group is mapped into a variable in the production rule set resulting from transformation T1.

Transformation T8: Each “obtain” transfer function outside an inference group is mapped into an action.

VI. CONCLUSION

In this paper, we have presented an approach for mapping CommonKADS knowledge models into specifications based on the PRR language. We have introduced the concept of "inference group", and defined an algorithm for automatically identifying inference groups, before proceeding to mapping transformations. In order to map knowledge models at the PIM level, we had to complete the concepts of PRR with UML activity diagrams.

Due to space limitations, it was not possible to illustrate our approach with an application. However, we have successfully applied the approach (including the mapping transformations) to a company buyout example, which is an example of assessment task.

Further research will concentrate on the following issues:

- Application of the mapping transformations to other scenarios (besides the company buyout example).
- Refinement of the transformations. In particular, in the current transformations, we do not take into account the distinction between sequential or inferencing operational modes, neither the concept of rule priority in the inferencing mode.
- Formal representation of the transformations with the QVT (Query/View/Transformation) language associated with MDA.
- Mapping of CommonKADS knowledge models into other languages than PRR at the PIM level (All CommonKADS knowledge is not necessarily meant to be represented as production rules).

VII. REFERENCES