Int. J. Appl. Math. Comput. Sci., 2011, Vol. 21, No. 1, 41–55
DOI: 10.2478/v10006-011-0003-4

amcs

DEFINING THE SEMANTICS OF RULE–BASED WEB APPLICATIONS THROUGH MODEL–DRIVEN DEVELOPMENT

JOAQUÍN CAÑADAS *, JOSÉ PALMA **, SAMUEL TÚNEZ *

* Department of Languages and Computation
University of Almeria, Ctra. Sacramento s/n, 04120 Almeria, Spain
e-mail: {jjcanada,stunez}@ual.es

**Department of Information and Communication Engineering
University of Murcia, Campus de Espinardo, 30100 Murcia, Spain
e-mail: jtpalma@um.es

Rule languages and inference engines incorporate reasoning capabilities to Web information systems. This paper presents an approach for the specification and development of Web applications performing the usual functionalities of data management and incorporating a rule engine for reasoning capabilities. The proposed approach is based on the definition of a high-level representation of the semantics of rule-based applications through a formalism for conceptual modeling combining lightweight ontologies and production rules. These models are used as the source for a model-driven method that applies several transformations to conceptual models generating the rule-based Web application code in an automatic process. As a result, the rule-based Web application embeds a rule engine suitable for deducing information by applying an inference process. The structure of the information managed by the Web application is based on ontology classes, whereas the logical expressions applied in reasoning are obtained from production rules of the model. A rule-based Web application has been developed and evaluated using a supporting tool that implements the ideas presented in this paper.

Keywords: model-driven development, rule-based systems, Web applications.

1. Introduction

The design of rule languages and inference engines to provide Web applications with reasoning capabilities is an important Semantic Web research topic. Several proposals have been defined, although there is no established standard yet (Eiter *et al.*, 2008). Rules are also used in software engineering since business rules have proven their usability for modeling business logic as part of specifications of software systems. Today, both points of view have merged, favoring the widespread adoption of rule-based systems and business rules in the implementation of complex decision-making processes (Object Management Group, 2008).

Each rule formalism enables the representation of different semantics and expressiveness in relation to the underlying logic supported, the reasoning methods and the inference process that can be applied (Paptaxiarhis *et al.*, 2009). However, in spite of that powerful semantic expressiveness, there is little methodological support to assist developers in implementing rule-based systems from rule models using proper technologies such as reasoners or rule engines. It becomes necessary to define methodologies and systematic approaches to map those rich expressive rule models to rule-based systems. This paper focuses on this goal.

Several rule types, such as integrity rules, derivation rules, production rules and reaction rules, have been identified (Wagner, 2002). In this paper, we are concerned with production rules. These (IF-condition-THEN-action) have been successfully applied both in knowledge-based systems and information systems since they enable a declarative representation of domain expert knowledge and business logic. Rule engines deal with rule sets and execute inference methods for firing the right rules to deduce information and obtain new results (Brachman and Levesque, 2004).

This paper addresses the development of rule-based systems embedded in Web applications. The integration of rule engines enhances system functionality, providing reasoning and inference capabilities to deduce new infor-

mation and reach intelligent conclusions from managed data. A model-driven approach to automate the development of rule-based Web applications is presented. Conceptual models combined with rules are used to define rule-based Web application semantics. Then, a set of model transformations are applied to convert those models into code. An important advantage of Model-Driven Development (MDD) is that it enables the definition and automatic execution of transformations between models and from models to code, substantially reducing the development time. On the contrary, since models do not represent all aspects of a rule-based Web application, many elements must be predefined. Hence, in this work the functionality for the rule-based Web application is predefined to enable end users to create, retrieve, update and delete instances (CRUD). In contrast to current tools for automatic generation of CRUD systems that perform those functions on relational databases, our contribution executes CRUD operations on the rule engine working memory, enabling the inference engine to execute a forward-chaining inference mechanism to drive the reasoning process.

To put our proposal into practice, we developed a support tool using MDD tools provided by the Eclipse Modeling Project[1]. The resulting Web application, based on the Model-View-Controller (MVC) architectural pattern, embeds the Jess rule engine (Friedman-Hill, 2003) to provide inference features. The proposed approach materializes InSCo (del Águila *et al.*, 2006), a methodology intertwining knowledge engineering and software engineering approaches for hybrid intelligent information systems development.

The rest of this paper is organized as follows. Sections 2 and 3 introduce the main technologies applied, model-driven development, and rule-based systems and rule modeling languages, respectively. Next, the rule-based modeling approach for specifying Web applications is described in Section 4. After that, the model-driven method for rule-based Web application development is explained in Section 5. The development process is evaluated and a case study is presented in Section 6. Related work is reviewed in Section 7. Finally, the main conclusions and future work are summarized.

2. Model-driven development

Model-Driven Architecture (MDA) (Object Management Group, 2003a) refers to a software development approach that uses models as first class entities, enabling the definition and automatic execution of transformations between models and from models to code. The term MDA was initially coined by the Object Management Group (OMG) and involved the use of several standards proposed by the OMG. In general, the terms Model-Driven Development (MDD) (Mellor *et al.*, 2003) and Model-Driven Engineering (MDE) (Schmidt, 2006) are used to refer to this software development approach.

The creation of metamodels for specifying modeling languages is a basic task in MDA/MDD. A metamodel defines a modeling language through the concepts, relationships, and integrity constraints available in the language.

The first model type that MDA defines is called a Platform Independent Model (PIM), a highly abstract model that is independent of any implementation technology. A PIM is transformed into one or more Platform Specific Models (PSMs). A PSM specifies the system in terms of implementation constructs that are available in a specific implementation technology. For example, a relational database PSM is a model of the system in terms of “table", “column", “foreign key", and so on. Finally, the last step in the development is transforming the PSM into code.

Model transformation consists of converting one model into another on a different abstraction level: from a PIM to a PSM and from a PSM to code, as shown in Fig. 1. Transformations between models are called model-to-model (M2M) transformations, and model to code transformations are called model-to-text (M2T). The main advantage of this approach to software development is that MDD tools enable these transformations to be specified and automatically executed, using MDA/MDD supporting languages and tools.

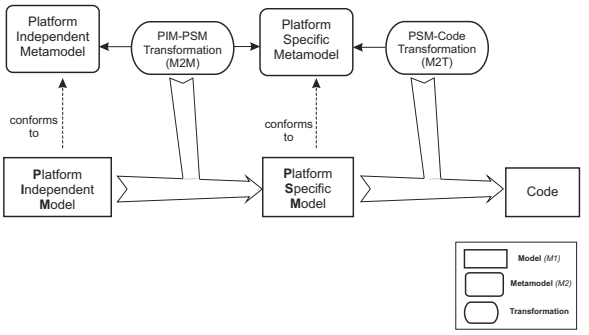

Fig. 1. Basic steps in the MDA/MDD development process.

This approach is currently being applied in many domains, such as embedded systems (Karsai *et al.*, 2008), Web engineering (Moreno *et al.*, 2008), ontology engineering (Gasevic *et al.*, 2006), and so forth. However, it has some limitations because, being relatively new, MDD supporting tools are not mature enough, and it also introduces some rigidity since model writing is not as flexible as source code writing.

3. Rule-based systems and rule modeling

Rule-Based Systems (RBSs) have been the leading technology for the development of knowledge-based systems since artificial intelligence emerged. An RBS is a kind of

[1] http://www.eclipse.org/modeling/.

software system in which the human expert's knowledge is applied to solve a complex task, such as diagnosis, monitoring, assessment, and so on, and it is represented as a set of declarative production rules. Rule engines are able to interpret the rules and reason using some inference method to come to a conclusion, just as the human expert would (Durkin, 1993; Russell and Norvig, 1995).

In general, an RBS consists of a set of rules (rule base), a working memory and an inference engine. Rules encode domain knowledge and business logic as condition-action pairs. The working memory represents system input first, but the actions that occur when rules are fired can cause the state of the working memory to change. The inference engine runs a reasoning method to fire rules, typically forward and backward chaining mechanisms, which involve inferring new data.

With Web evolution, interest in making the Web a more effective platform for intelligence systems quickly emerged (Grove, 2000). Rules and ontologies provide meaning and reasoning facilities to Semantic Web applications (Eiter *et al*., 2008). Although ontology formalisms are quite well developed and the Web Ontology Language (OWL) (Dean *et al*., 2004) has become a standard, rule formalisms are still an active area of research, considerable effort being made to develop rule languages and inference engines that add reasoning to complex information systems.

The OMG proposed the ontology definition metamodel (Object Management Group, 2009a) and the production rule representation (Object Management Group, 2009b) as standard metamodels for introducing both technologies in the context of MDA. Relevant initiatives to standardize and exchange rules are the Rule Markup Initiative (RuleML) (Boley *et al*., 2001), the Semantic Web Rule Language (SWRL) (Horrocks *et al*., 2004), and the REWERSE Rule Markup Language (R2ML) (Wagner *et al*., 2006). Along the same line, the World Wide Web Consortium (W3C) urges rule-based interchange for the Web with the Rule Interchange Format (RIF) (Kifer, 2008).

The software engineering community has also focused on rules as a proper formalism for representing business logic in software systems. Today these two points of view have merged, favoring the widespread adoption of RBSs and business rules in the implementation of complex decision-making processes (Object Management Group, 2008).

4. Modeling rule-based Web applications

The model-driven approach for rule-based Web application development we propose focuses on the specification of Web semantics using a formalism that combines rules and lightweight ontologies.

Lightweight ontologies are comparable in expressiveness to conceptual models widely used in software engineering, such as Unified Modeling Language (UML) class diagrams (Kogut *et al*., 2002). They enable domain structure specification using concepts, concept taxonomies, relationships between concepts and properties that describe them. In contrast, *heavyweight* ontologies add axioms and constraints to lightweight ontologies, clarifying the meaning of the terms collected in the ontology. Hence, heavyweight ontology formalisms are semantically richer than conceptual models (Gómez-Pérez *et al*., 2004).

We use a knowledge modeling formalism proposed in the CommonKADS methodology called the Conceptual Modeling Language (CML) (Schreiber *et al*., 2000), which has several desirable features: (i) it enables unified representation of lightweight ontologies and rules, in which rules and ontologies are naturally related by binding rules to ontology concepts; (ii) it meets the requirements of rule representation formalisms, such as modeling rule antecedent, rule consequent, named rules, and rulesets, with an expressiveness comparable to the SWRL, which is neither highly expressive nor decidable, but it remains simple (Paptaxiarhis *et al*., 2009); (iii) models written in this formalism are independent of any implementation technology and, therefore, can be used as the source model in a model-driven approach; and (iv) the use of a non-standard modeling language allows us to extend the notation for representing some semantic features of rule-based Web applications, as explained below.

4.1. Motivating example. We use a simple example to illustrate our work, i.e., the family relationships use case (Horrocks *et al*., 2004). The concept *Person* is defined by a set of attributes such as *name*, *age*, *gender*, *father* and *mother*. *Name* and *age* are primitive types, *String* and *Integer*, respectively. *Gender* is the literal *male* or *female* defined as an enumerated type. The type for *father* and *mother* is the concept *Person*. Each of these attributes can have only at most one value, so their cardinality is single. Relative relationships for a person, such as *siblings*, *brothers*, *sisters*, *children*, *sons*, *daughters*, *uncles*, *aunts*, and so on, can be defined using either *Person*-to-*Person* binary relationships or *Person* type attributes. The second option was chosen for simplicity, using a cardinality of "many" since there may be a list of values (*siblings*, *brothers*, etc. of a person can be zero, one or many). Figure 2 shows a UML class diagram for the family example.

Rules for deriving the values of relationships with relatives are defined in the *Person* class. For example, if two *Person*s have the same father and the same mother, then they are siblings. Using a pseudocode syntax, with variables x, y of the *Person* type, this rule can be expressed as follows:

```
IF (x.father==y.father)
AND (x.mother==y.mother)
```

```
THEN (x.siblings = y).
```

Similarly, the 'uncle' rule is defined by the expression

```
IF (x.father==y) AND (y.brother==z)
        THEN (x.uncle = z).
```

4.2. Conceptual rule-based modeling. The CML formalism for knowledge modeling enables the specification of lightweight domain ontologies and production rules. A CML model is basically composed of two elements, domain schemata and knowledge bases. Domain concepts, binary relationships, rule types and value types (enumerated literals) are modeled in a domain schema. A knowledge base consists of instances of concepts (individuals), instances of relationships (tuples), and instances of rules. Figure 3 shows the knowledge model components.

As presented next, the CML metamodel specifies the primitives that can be used in the modeling language, using a simplified UML class diagram notation called MOF (Object Management Group, 2003b). The portion of the CML metamodel defining *Concept*, *ValueType* (enumeration) and *BinaryRelation* primitives is shown in Fig. 4.

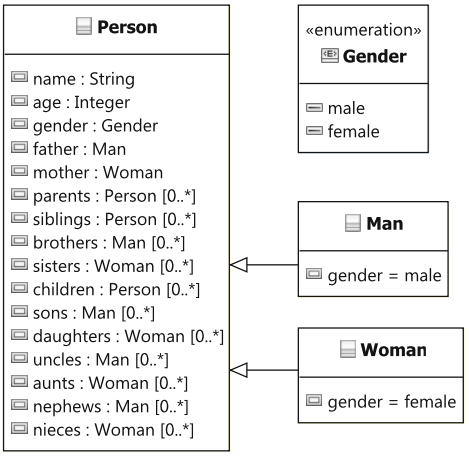

Fig. 2. UML family model.

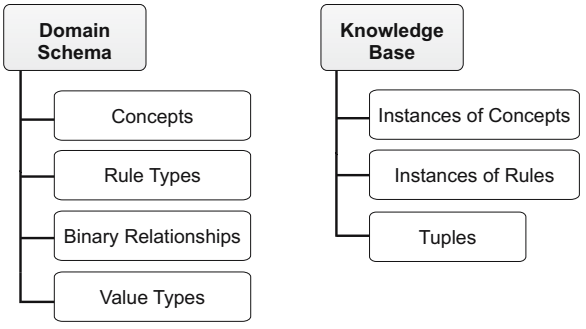

Fig. 3. Domain knowledge structure.

The main difference between this formalism and other conceptual modeling approaches in software engineering, such as UML class diagrams, is the ability of modeling production rules due to the *rule type* and *rule instance* primitives. A *rule type*, defined as part of a domain scheme, describes the structure of a set of rules through the specification of the types bound to the rule's antecedent and consequent. The metamodel for *rule type* is shown in Fig. 5. A set of rules with similar structures are modeled as *ImplicationRule* that extends *RuleType*. Each implication rule has *ConnnectionSymbol*, a word or text representing the meaning of the rule in natural language, and an antecedent and a consequent that bind to one or several *Concept*s or *BinaryRelation*s.

*Rule type*s are particularized into *rule instances* which represent specific, logical dependencies between the rule's antecedent and consequent concept attributes. A *knowledge base* contains instances of concepts, relations and rule types, which are respectively defined in the metamodel by *Instance*, *Tuple* and *RuleTypeInstance* classes. Production rules are specified as *RuleTypeInstance*s by means of conditions or expressions related to the concepts defined in the rule type antecedent and consequent (Fig. 6). Conjunctions, disjunctions and negations can be defined as antecedents of rule instances. Several binary operators, summarized in Table 1, can be used in expressions, some of them defined for single values and others for lists of values. Both variables and literal values can be used as operands in conditions and actions.

The family model can be specified conforming to the metamodel described. The *Person* class is defined as a concept with a set of attributes. Those with only one value are defined with *singleValued* cardinality (*father*, *mother*, ...), whereas *multiValued* cardinality is applied to attributes with a list of values (*brothers*, *sisters*, ...). The *Gender* type is defined as an enumerated *ValueType* with two literal values, *male* and *female*. *PersonAbstraction* is a *rule type* defining the skeleton for rules that relate *Person* properties in the antecedent with *Person* properties in the consequent. Concrete rules for deriving each family relationship are defined as rule instances of that rule type. An excerpt from the family model, created with an authoring tool based on the proposed metamodel, is shown in Fig. 7. This tool is described in Section 5.4.

4.3. Adding specific features for rule-based Web modeling. Models specified using the formalism described above represent the application domain, based on ontology classes, using logical expressions from reasoning and business logic based on production rules. Other formalisms combining rules and ontologies, such as those mentioned in Section 3, can be equally expressive. However, since our models are intended for use in a model-driven method for rule-based Web application development, the formalism has been enriched to include some specific de-

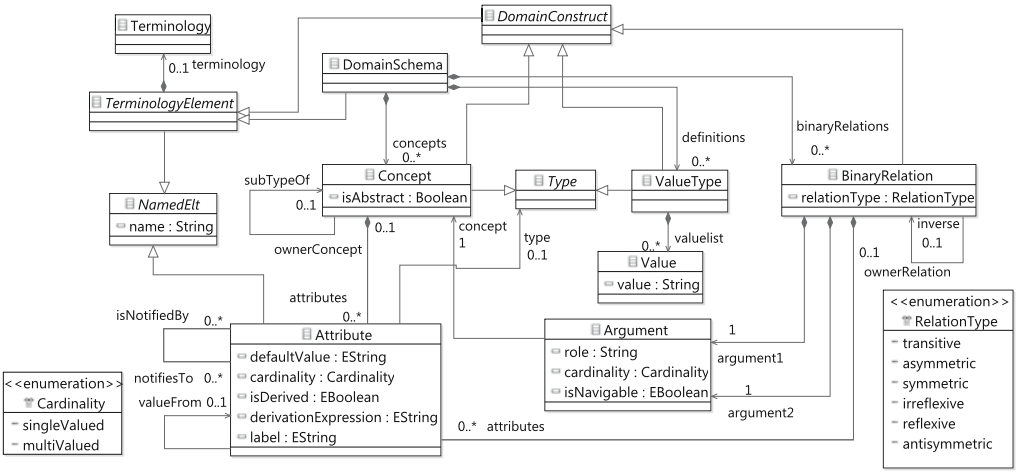

Fig. 4. Excerpt from the CML metamodel: concept and relationship.

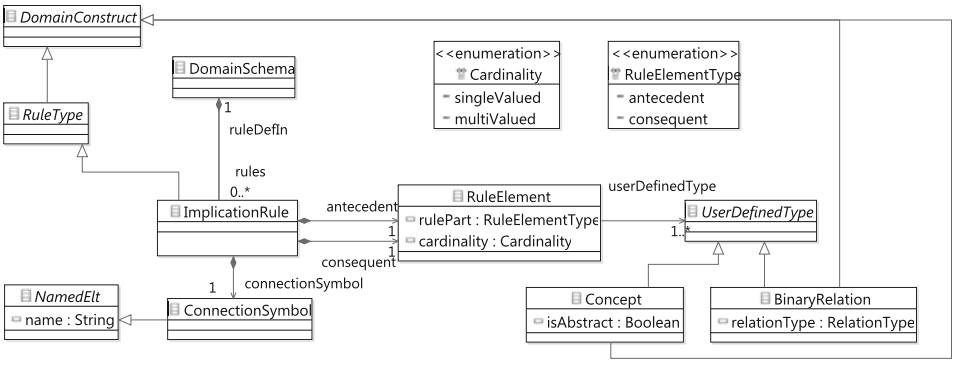

Fig. 5. Excerpt from the CML metamodel: rule type.

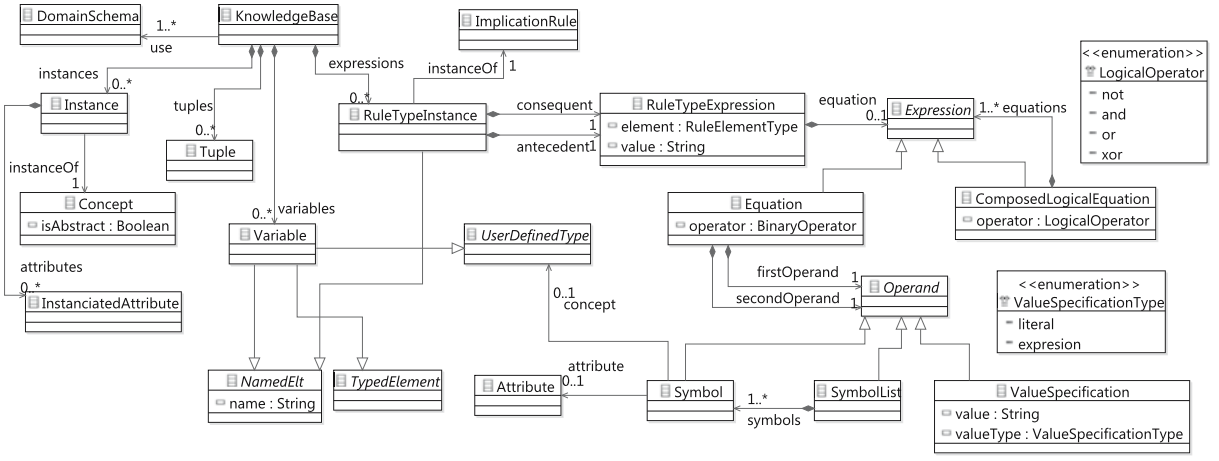

Fig. 6. Excerpt from the CML metamodel: knowledge base and rule type instance.

Table 1. Binary operators in the CML.

Operator	*Operands*	*Result*	*Description*
<	numeric	logical	less than
>	numeric	logical	greater than
<=	numeric	logical	less or equal than
>=	numeric	logical	greater of equal than
==	numeric	logical	equal
!=	numeric	logical	distinct
+	numeric	numeric	plus
-	numeric	numeric	minus
*	numeric	numeric	product
/	numeric	numeric	division
=	*any*	*any*	assignment of type compatible values
add	list, element	list	adding an element to the list
remove	list, element	list	removing an element from the list
includes	list, element	logical	*true*, the list includes the element; *false* otherwise
excludes	list, element	logical	*true*, the list does not include the element; *false* otherwise
includesAll	list, list	logical	*true*, the first list includes all elements of the second list; *false*, otherwise
excludesAll	list, list	logical	*true*, the first list does not include all elements of the second list; *false*, otherwise
union	list, list	list	union of lists
intersection	list, list	list	intersection of lists
difference	list, list	list	difference of lists

sign features for this kind of software systems, such as interaction and presentation characteristics.

Interaction features enable the specification of user interactivity in the Graphical User Interface (GUI) through a set of properties associated with modeling primitives. The following properties dealing with the attribute metaclass (see Fig. 4) illustrate some interaction characteristics:

- *isDerived*: set to *true* when the attribute value is inferred by the rule engine, so it cannot be edited by the user.

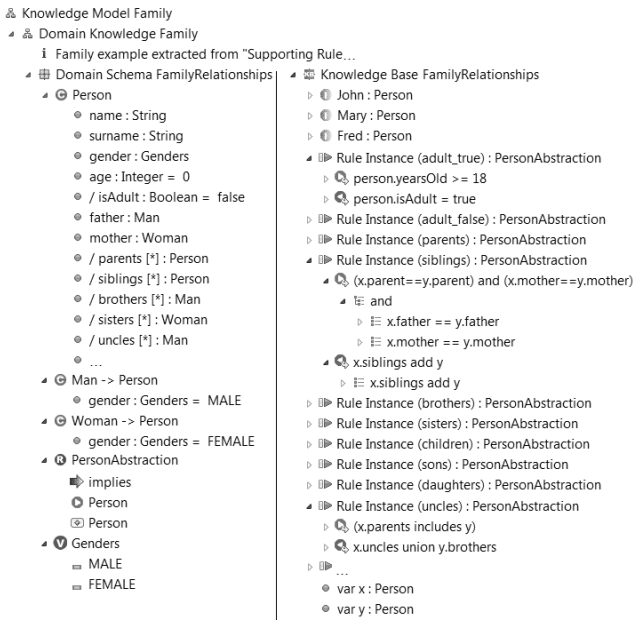

Fig. 7. Excerpt from the family model.

- *notifiesTo*: sets the attributes to be refreshed or updated in the user view when the attribute value changes.
- *isNotifiedBy*: the opposite of *notifiesTo*.
- *valueFrom*: sets the attribute that contains the list of values to be chosen as the value for the current attribute.

In the family example, the property *isDerived* is *true* for all the attributes of *Person* whose values are inferred from rules, such as *siblings*, *brothers*, *sisters*, *children*, *sons*, *daughters*, *uncles*, *aunts*, and so on. The *father* attribute *notifiesTo* informs all the other attributes that are affected, such as *siblings*, *brothers*, *sisters*, *uncles* and *aunts*, when the father of a person changes. The *uncles* attribute *isNotifiedBy* is informed by *father* and *mother*. Finally, if a new attribute called *favoriteUncle* is added to *Person*, enabling one of the uncles to be selected as the favorite, it obtains its possible values from *uncles* by the proper setting of *valueFrom*.

Figure 8 shows an example of how the interaction of the *father* with *parents* and *siblings* attributes is specified using the *isDerived* and *notifiesTo* properties, and how it affects the generated Web forms generated for editing *Person* class instances. *Parents* and *siblings* are derived attributes so their values cannot edited by the end user but deducted by the rule engine. Moreover, *father* notifies to both *parents* and *siblings*, so when the father's value changes then an *onChanged* event makes the rule engine run and the lists of *parents* and *siblings* are re-rendered, updating them with the new values deduced by the rule engine.

Presentation features specify the conceptual model element's visibility properties, enabling user interface customization. For example, this makes it possible to select

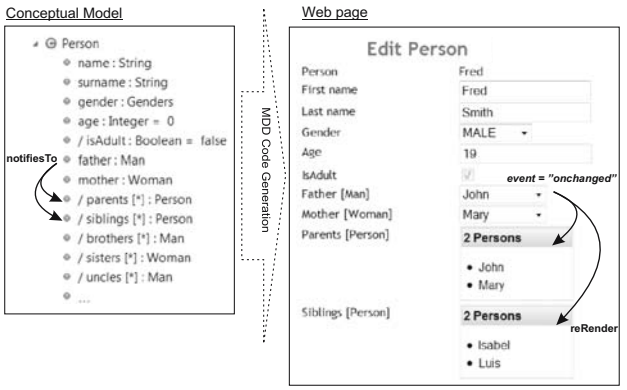

Fig. 8. Modeling interaction with the *NotifiesTo* property.

what concepts will appear in the application menu, and what attributes are included as columns of tables showing all instances of a concept type.

5. MDD for rule-based Web applications

5.1. General perspective. Figure 9 shows the proposed MDD schema for rule-based Web applications, which is divided into two processes. The first one (the bottom flow in Fig. 9) generates the implementation of the rule base in Jess, and the second one (the top flow in Fig. 9) produces the code for the Web architecture.

The development starts with the specification of a conceptual rule model defining the domain ontology and the set of rules representing the decision logic, using a platform-independent formalism such as the CML. The application of the model-driven approach produces two different results. On the one hand, ontology and rules are transformed into Jess, which supports the development and deployment of rule-based systems tightly bound to Java applications. As a result, a Jess rule base, i.e., a text file containing the set of rules converted to Jess syntax, is generated.

Furthermore, a Web-based architecture is generated from the CML model extended by interaction and presentation features. The Web application code is based on the MVC architectural pattern, the JavaServer Faces (JSF) framework (Geary and Horstmann, 2007) and JBoss Richfaces components (JBoss, 2009b), producing a set of JavaBean classes and JSF pages.

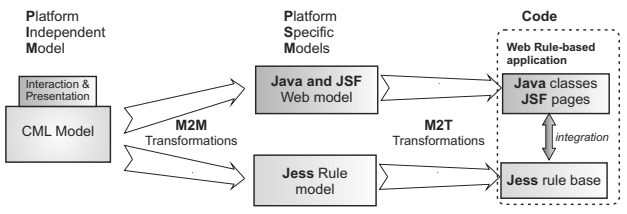

Fig. 9. MDD schema for rule-based Web system generation.

Although the two MDD processes are executed independently of each other, the final result must integrate the rule base into the Web application. This is done by the appropriate method calls to the Jess API (Application Programming Interface) in the Java code generated, which entails embedding the Jess rule engine into the Web application.

Figure 10 shows the target architecture for the rule-based Web applications generated. The embedded rule engine manages the Jess rule base and the text file of persistent instances of concepts, called *facts*. This rule engine consists of three basic parts: the working memory contains all the information or *facts*, the rule set contains all the rules, and the rule engine checks whether the rules match the working memory and then executes them. A set of Java classes and JSF pages for creating, editing and listing instances of each concept type comprises the Web architecture.

The Web application enables the user to perform four basic predetermined functions: create new instances, read the current lists of instances, update and delete instances. These CRUD operations are executed on the Jess rule engine working memory, enabling the inference mechanism to fire appropriate rules when necessary. The rule engine executes a forward-chaining inference mechanism to drive the reasoning process, firing the rules when conditions are evaluated as true, and executing actions to modify the existing information or infer new one.

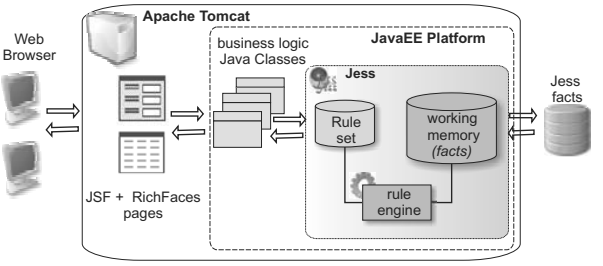

Fig. 10. Rule-based Web application architecture.

The Web application generated benefits from decision logic externalized from the core application code, since uncoupling the rules from the source code increases the scalability and maintainability of rule-based applications (Frankel *et al.*, 2004). In addition, our development approach makes it possible for the two MDD processes to be executed separately and, therefore, any change in the rule model affecting only rule logic (rule instances) but not the information structure (concepts, relationships, ...) can be translated into a new rule base without having to modify or regenerate anything else in the Web architecture. This approach makes Web applications easier to maintain and evolve.

5.2. MDD of Jess rules. As previously mentioned, the first model-driven process in our approach involves the transformation of source models into the Jess code. The first model-to-model (M2M) transformation involves translating the CML source model into a platform-specific model for Jess rules, for which a platform specific Jess rules metamodel (Fig. 11) is designed, extending a simple rule metamodel for rule-based systems as described by Chaur (2004).

In the Jess rules metamodel, a metaclass is defined for each Jess language element. The root element of a Jess rule model is *RModule*. A module contains fact templates, rules, functions, facts and queries. The *RFactTemplate* metaclass models fact templates, the Jess constructor for storing information.

RRule enables the representation of rules. A rule has *ruleName*, a property called *salience* that determines the order in which applicable rules are fired by the rule engine, a containment reference *condition* representing the rule's condition (antecedent), and a reference called *actions* representing the rule's action (consequent). The *RPattern* and *RSlotMatchExp* metaclasses define pattern matching expressions in rule conditions. Actions are defined as *RFunction* calls that assert new facts, or retract or modify the existing ones.

Facts are defined by the *RDefFacts* metaclass. And acquired from instances of concepts in the CML source model. Finally, *RQuery* models queries to consult the working memory at runtime.

Mapping from CML models to Jess rule models is designed by an M2M transformation which maps CML metamodel primitives to one or several Jess rule metamodel constructors. The following rules briefly summarize this mapping:

- *Domain Schema to Knowledge Base*. A domain schema maps to a Jess knowledge base, that is, a `.clp` file. The name of the domain knowledge is the same as the name of the corresponding `.clp` file.
- *Concept to Template*. A *Concept* is transformed to a fact template in Jess, that is, *RFactTemplate*. The fact template name is the same as the concept name. Each concept attribute maps to *Slot* in the Fact Template. The slot name corresponds to that of the attribute, single attribute cardinality maps to a slot whereas multivalued attributes map to a multislot.
- *Relation to Slot*. Each binary relationship is translated into a slot in the concept related by the first argument in the relationship.
- *Concept Instance to DefFact*. An instance is mapped to *RDefFact*, and instantiated attributes correspond to slot values in the fact definition. When an instance has a slot for a relationship, a tuple gives the value to that slot.
- *RuleType and RuleInstance to Rule*. Each rule instance is mapped to *Rule* with *Antecedent* and *Consequent*. *Antecedent* is composed of patterns and matching expressions referring the templates included in the rule type definition. *Pattern* indicates the templates that must have a pattern matching expression in the rule. *Matching Expression* represents a condition in the antecedent of a rule, for example, a logical comparator expression.
- *Concept to Queries*. Finally, two *Queries* are created for each Concept, *all instances query* and *Fact by-Id query*. The formed query is necessary for the JSF Web interface to retrieve all the instances of a specific class, and the latter for it to search for a specific instance by its identifier slot value.

The Jess rule model generated by the M2M transformation is the source model for a model-to-text (M2T) transformation which automatically generates the Jess rule base source code, producing a Jess file `(.clp)` with a code for every element included in the Jess rule model.

5.3. MDD of the JSF Web architecture. A second MDD process is applied (see Fig. 9) to generate a Web architecture embedding rules into a Web application. In this process, Jess rules are integrated into the Web application, since both the Jess rule base and the Web architecture are generated from the same model. Integration issues are managed and resolved.

The Web application generated is divided into two layers, a business and a user interface. The business layer is composed of a JavaBean class package taken from input model concepts, a set of helper Java classes to implement some functionalities, and a *JessEngineBean* class that integrates Jess rules into the Web application using the Jess API to achieve interaction with Jess. The user interface layer is composed of a list of JSF Web pages that allow the user to interact with the system and perform predetermined functionalities.

A JSF Web architecture metamodel was designed (Fig. 12). A JSF project is composed of a set of JSP pages, a configuration file `(faces-config.xml)` and a set of JavaBeans. In the M2M and M2T transformations from a CML model to a JSF model and finally to code, each concept is mapped to several elements: a JavaBean class, a JSF page for creating and editing instances, a JSF page for listing all such instances, and a managed bean to be included in the configuration file. Interaction and presentation features are taken into account at this level in model-driven processes.

5.4. Tool support: InSCo-Gen. To put our proposal into practice, the InSCo-Gen supporting tool (Cañadas *et al.*, 2009) applies the model-driven approach to rule

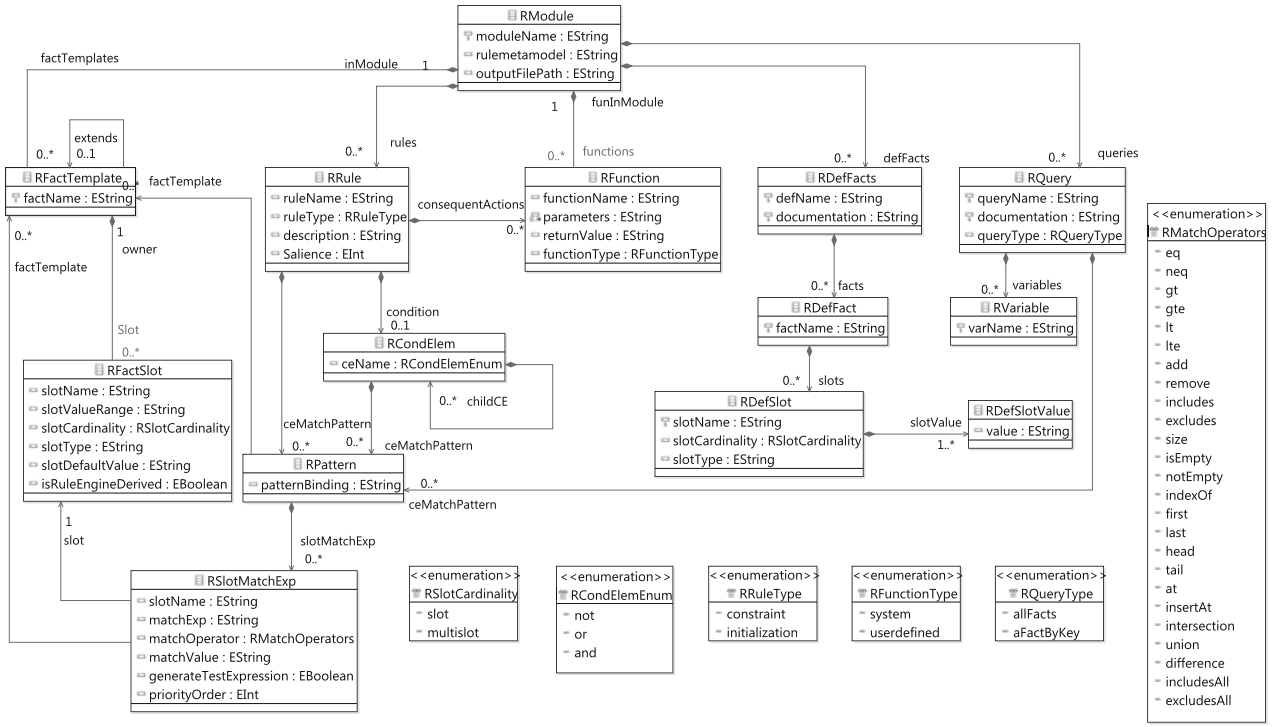

Fig. 11. Jess rules metamodel.

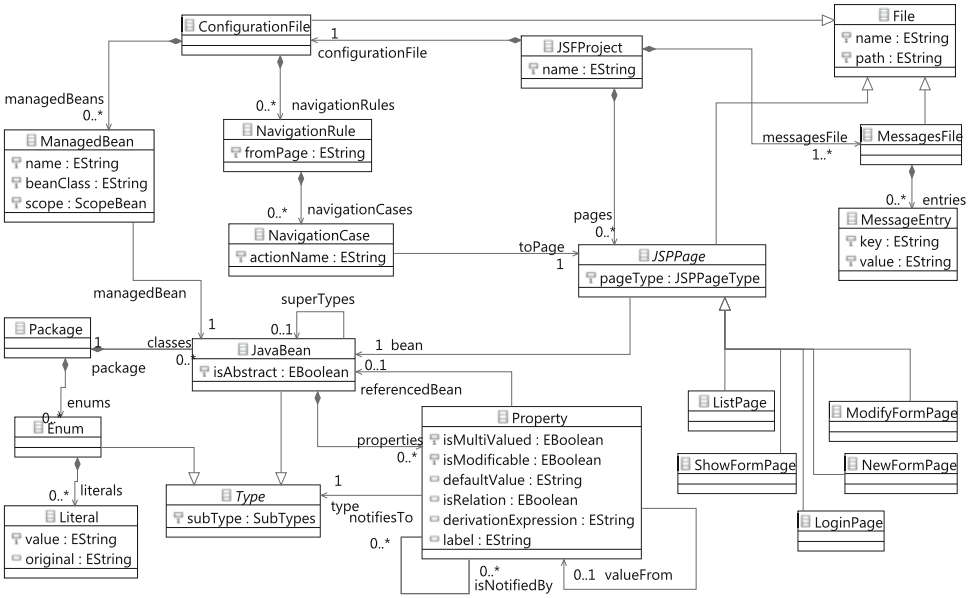

Fig. 12. JSF architecture metamodel.

models and automatically generates the implementation of a functional rule-based Web application. It was developed using Eclipse Modeling Project tools for MDD. Metamodels are defined using the Eclipse Modeling Framework (EMF[2]). Models conforming to metamodels are created using the built-in reflective EMF editor, which provides an authoring tool for CML models. Model creation is improved by customizing the reflective editor with Exeed (EXtended Emf EDitor) (Kolovos, 2007), a plugin which can modify editor default icons and labels by adding Exeed annotations to the metamodel. A screenshot of a model created with this editor is shown in Fig. 7.

Two M2M transformations were designed with the Atlas Transformation Language (ATL[3]). The first one maps a CML model to a Jess platform-specific model. The second one transforms a CML model into a JSF-specific model.

The outputs of both ATL transformations are the inputs of two M2T transformations implemented with Java Emitter Templates (JETs[4]). As a result, InSCo-Gen automatically produces the Web application code, source text files with Jess rules and facts on the one hand, and, on the other, the Web application components, the `faces-config.xml` and `web.xml` configuration files, the Java Beans for model classes, and a Jess-Engine Bean which uses the Jess Java API to embed the rule engine into the architecture. Moreover, a set of JSP/JSF Web pages are generated for the user interface. These pages are based on the RichFaces library (JBoss, 2009b), an open source framework that adds AJAX capability to JSF applications.

6. Evaluation

6.1. Development process. The proposed approach has some strengths and weaknesses. The main advantage is a drastic reduction in the development time and effort, since the tool does all the hard work. The incremental development is also favored with this approach because changes in the model can be directly reflected in system implementation. In addition, models are now the main artifacts in software development, not only out-of-date documentation, a common characteristic of MDA/MDD approaches. Furthermore, the final code quality increases, as good design and best practices are implemented in the transformations, producing a free-of-error code.

There are also several limitations. Significant time and effort necessary for developing the MDD supporting tool must be considered in evaluating the tool's benefits versus the effort required for its construction. When the goal is the development of only one rule-based system in a specific domain, software construction from scratch is the best option. However, when a kind of software solution can be applied to many similar systems and in many domains, then the construction of an MDD supporting tool like the one presented here is an appropriate solution.

Another important limitation concerns development process evolution and related tools. Models are limited to the expressiveness of the metamodels, and code generation is also limited to what can be expressed in the models. Therefore, when a change in model expressiveness or code generation is necessary, then changes in metamodels, models and transformations must be derived, making MDD tool evolution difficult.

6.2. Testing the *Family* Web application. To evaluate the resulting Web application, we tested its performance with the *Family* case study as described below. Later, the usability of the Web application is discussed.

The rule-based Web application was tested using different numbers of individuals (10, 20, 50, 100, 200, 500, 1000) and two different rule sets, the first with only 5 rules the second with 20 rules. The results are shown in Fig. 13. Execution times in the figure plot the time it takes to charge the knowledge base, that is, the time the rule engine spends on executing the rules with all the individuals just loaded for the first time. Since, at first, individuals only have a few necessary properties, such as *age*, *father* and *mother*, all the other family relationships are calculated during charging. This becomes an expensive processing task when the number of individuals increases, as the figure shows. The number of rules also has an extreme effect on the initial charge time. The time scale for the 5-rule test (left-hand scale) is much shorter than that for the 20-rule test (right-hand scale). However, both rule sets behave the same as the number of individuals increases, and both curves show the same trend of exponential growth.

It is important to note that, once the initial charge has

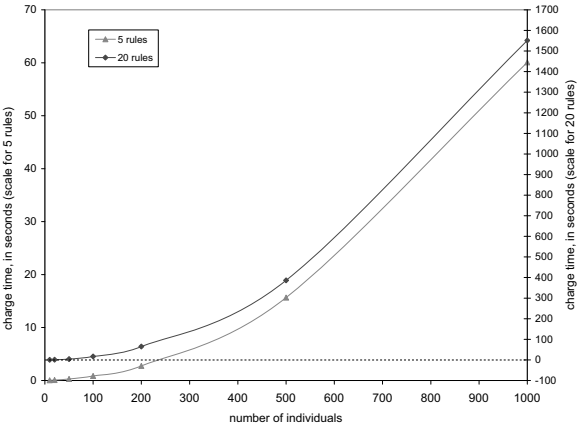

Fig. 13. Charging times for the *Family* Web application.

[2]http://www.eclipse.org/modeling/emf/.

[3]http://www.eclipse.org/m2m/atl/.

[4]http://www.eclipse.org/modeling/m2t/?project=jet.

finished, the execution of the Web application remains interactive, since creating, modifying or deleting instances only affect one or a few individuals (family-related persons). So when the user interacts with the application, the rule engine fires the appropriate rules in milliseconds providing an immediate response to the user. This performance was tested in the use case evaluated. However, production rule engines could offer a different response in other cases, since, when a fact is matched at the top of many rules in many combinations, it can take a long time to modify it (Friedman-Hill, 2003).

The resulting rule-based Web application can be accessed by end users. Its usability is improved by the use of AJAX technology and the application of good design practices. In fact, both rules and AJAX technology improve the creation and edition of instances in the Web application. Since Web forms are implemented with AJAX RichFaces components, each single form value can be validated and submitted individually as it is entered. This facility entails the rule engine firing suitable rules and inferring new information that drives instance creation or edition, for example, updating choice-fields. A snapshot of a *Family* Web application listing page is shown in Fig. 14 and an editing page in Fig. 15.

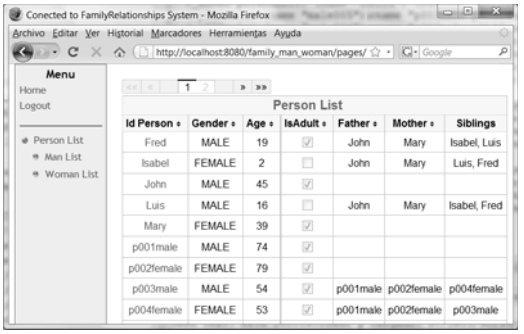

Fig. 14. *Family* Web application: list of all persons.

6.3. Case study in the agricultural domain. The proposed approach was applied in SAVIA, a rule-based Web information system for pest control in agriculture.

The productivity of present-day agricultural industry has multiplied with the incorporation of new technologies and technological innovations in traditional agricultural production systems. This is the case in south-eastern Spain, where agriculture is one of the most important economic sectors, especially in the Province of Almeria called "the Garden of Europe" because of its intensive greenhouse farming. Almost its entire vegetable production is exported to the rest of Europe. However, improper use of pesticides during recent years, when pesticide residues in some consignments were above the legal limits, led to a negative image in Europe, and even blocking some frontiers to products from Almeria.

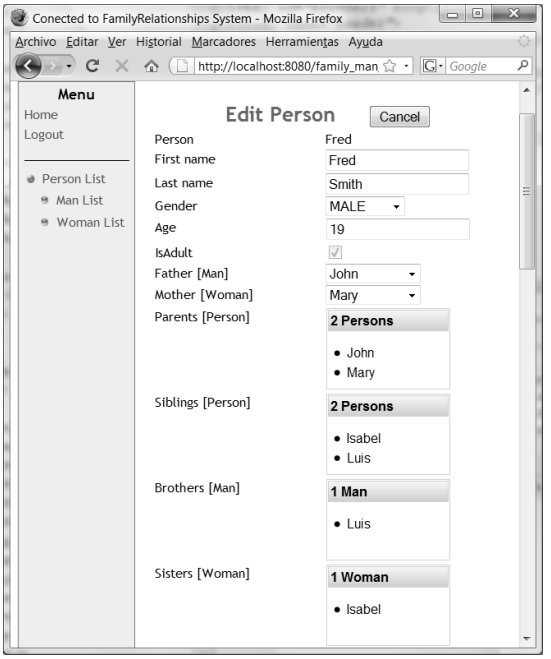

Fig. 15. *Family* Web application: person edition.

Plant health is therefore a significant factor in the economy of today's agricultural industry and has an important effect on the environment. It is one of the major concerns of both agricultural companies and authorities. The Spanish government has developed an Integrated Production (IP) quality regulation. IP is defined as agricultural production systems that use natural production resources and mechanisms to ensure long-term sustainable agriculture. In IP, biological and chemical treatments are carefully selected, keeping in mind consumer demands, economics and environmental protection.

Information technologies and specifically knowledge-based technologies applied to IP pest control can improve process management and effectiveness. With this goal, the SAVIA decision-support system applies rule-based reasoning to help growers and agricultural technicians grow their crops according to IP quality regulations by informing them whether a given treatment is necessary for pest control. It also provides online decision support to avoid problems in their crops, such as incorrect use of pesticides.

The pest control problem is formulated as a therapy administration task (del Águila *et al.*, 2003), a composition of knowledge-intensive tasks such as monitoring, diagnosis and assessment. A crop is considered a complex system made up of land, plants, pests and any useful fauna which can reduce damage. This system is affected by external variables such as climate conditions, humidity, product price, and so on. Controls that represent the least possible hazard for the crop, useful fauna and the environment can be applied to keep their balance.

The SAVIA knowledge model consists of several concepts, such as *Harmful Agent* (*Pest* or *Disease*), *Crop*, *Phenological Stage*, *Grower*, *Agricultural Technician*, and so on. Instances of domain knowledge are described, for example, *Grapes* and *Tomatoes* are instances of *Crop*; *Grape moth*, *Thrips* and *Botrytis* are instances of *Harmful Agent*; *Dormant bud*, *Begin flowering* and *Cap falling* are instances of *Phenological Stage*. Rules specify logical constraints on concepts attributes. An example of a rule for the *grape crop* is as follows: *If the date of a Sample is in January, then the phenological stage is Dormant bud*.

Crops must be monitored periodically. Data about the crop growth stage, the presence of pests or their effects and other useful information are collected by sampling. The system manages all that information. A crop sampling list is shown in Fig. 16. The rule logic implemented in SAVIA provides decision support related to two clearly differentiated tasks: (i) first, a decision about whether or not to take action, and (ii) in the affirmative, the most appropriate treatment (chemical, biological or mechanical). Although the system advises on action to be taken, it is the agricultural technician who ultimately decides what action to take.

7. Related work

Our proposal uses the CML rule and ontology modeling formalism as the MDD source model. To insert the CML in the MDD framework, we defined a CML metamodel. A UML profile for the specification of CML knowledge models is addressed by Abdullah *et al*. (2007), who also discuss the possibility of mapping the profile elements to a Jess platform-specific model.

Some previous articles have proposed the generation of Jess rules from ontology and rule models, such as the OWL (Mei *et al*., 2005) and the SWRL (O'Connor *et al*., 2005). However, they focus on the definition of reasoners for OWL and Jess, where the main issue is to derive certain facts based on OWL and SWRL semantics, respectively. The most important difference between those proposals and ours is that they run the rule base generated in a development tool such as Protege JessTab (Eriksson, 2003), using a reasoner embedded in the environment to execute rules, whereas we generate the Jess rule base and also the code necessary to embed the Jess rule engine into a functional Web application, providing a Web application for end-users.

Web engineering defines processes, techniques and models suitable for Web application development. Methodologies such as UWE (Koch *et al*., 2008), WebML (Fraternali *et al*., 2002) and WebDSL (Groenewegen *et al*., 2008) approach the design and development of Web applications by providing tools based mainly on conceptual models (Ceri *et al*., 2002) and focusing on content, navigation and presentation models as the most relevant concerns (Moreno *et al*., 2008; Linaje *et al*., 2007). A model-driven approach for Web application development based on MVC and JavaServer Faces is described by Distante *et al*. (2007). However, those methods do not consider rule modeling to be a specific concern in Web application development. Our proposal is innovative in that it introduces rule modeling in this context. To simplify our description, we have not considered other Web application modeling concerns such as navigation, and we predefine functionality as CRUD operations and, therefore, types and navigation links are fixed and preset.

There is currently no defined methodology or development process for creating rule-based systems embedded in (semantic) Web applications, and it is still a subject of research. Diouf *et al*. (2007) describe how to merge UML models and OWL ontologies for automatic business rule generation. They propose the use of ontologies for adding semantics to UML models and applying the MDA approach to the extraction of implicit rules from models. Their business rules are generated in the Semantics of Business Vocabulary and Business Rules (SBVR) syntax (Object Management Group, 2008). Although they propose a general architecture for applying MDA and OWL ontologies for generating rulesets in a target rule engine during the generation step, up to know, they have only generated the first abstraction version of business rules.

Another related method, the HeKatE methodology (Nalepa and Ligęza, 2010), describes an approach similar to MDD for the design and development of complex rule-based systems. It uses extended tabular trees for rule representation, a paradigm based on decision tables connected into an inference network. Rules are translated into a Prolog-based code, and integrated as a logic core module in the final software system.

Regarding MDD for Web applications integrating rules, Ribarić *et al*. (2007) describe MDD principles for rule-based Web service modeling using R2ML and propose an MDD approach for generating Web services from rule models. But whereas that proposal focuses on a Web service architecture, our work is based on an MVC architecture using a JSF framework.

8. Conclusions

This paper describes how rule-based and model-driven development techniques are intertwined for the development of rule-based Web applications. The semantics of such software applications are defined through a modeling formalism combining lightweight ontologies and rules, and a model-driven approach establishes how the implementation is generated from models. The ideas proposed are demonstrated by an MDD tool.

The resulting rule-based Web architecture implements the MVC architectural pattern using the JavaServer Faces framework and incorporates rich JBoss Richfaces

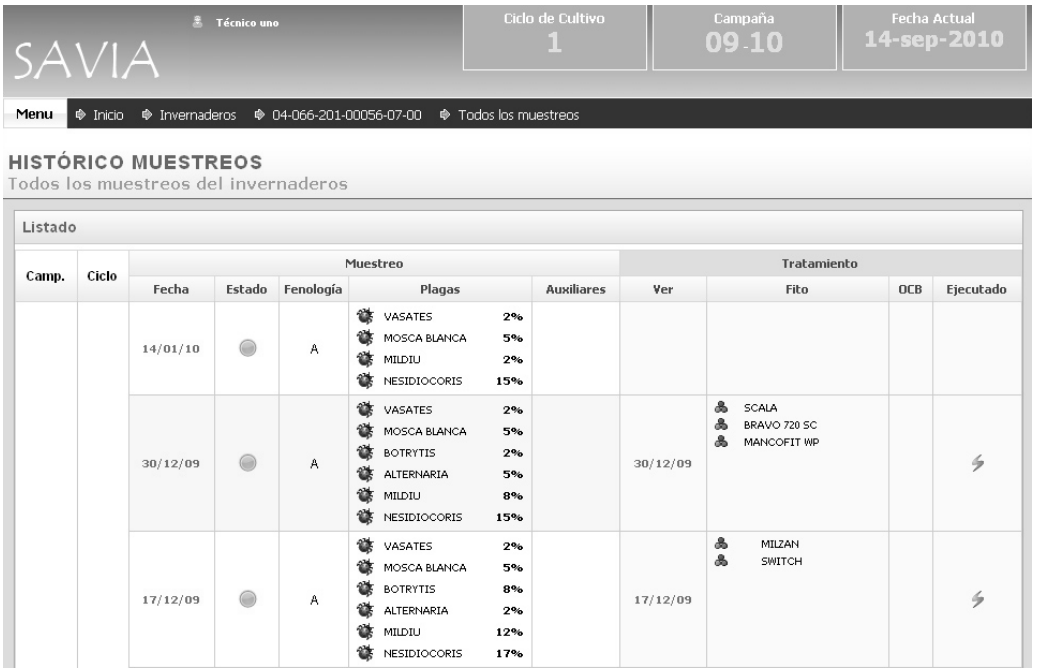

Fig. 16. Crop sampling list.

components to enhance the user interface with AJAX capabilities. The Jess rule engine is embedded in the Web application to provide inference capabilities.

Due to the declarative nature of rules, the decision logic is externalized from the core application code producing Web applications that are easier to maintain and evolve.

This approach has been evaluated with a simple example and a real-world case study, the SAVIA system—a Web decision-support system for pest control in agriculture, which makes recommendations to growers and technicians about the necessity of treating a specific pest or disease in their crops. The approach is planned to be applied in other domains such as medical diagnosis.

Future work will use other ontology and rule modeling languages such as the OWL and the SWRL as source models for the model-driven approach and define interoperability modules with other rule formalisms. Different rule platforms, such as JBoss Rules (JBoss, 2009a), will also be target rule technologies. The Web application generated, which is aimed at enriching the architecture with database facilities, will be improved to provide a complete persistence layer.

Acknowledgment

The authors wish to thank the Spanish Ministry of Education and Science for funding received under Projects TIN2009-14372-C03-01 and PET2007-0033, and Junta de Andalucia (Andalusian Regional Governitent) for support under Project P06-TIC-02411.

References

Abdullah, M.S., Benest, I.D., Paige, R.F. and Kimble, C. (2007). Using unified modeling language for conceptual modelling of knowledge-based systems, *in* C. Parent, K.-D. Schewe, V.C. Storey and B. Thalheim (Eds.), *26th International Conference on Conceptual Modeling, ER 2007, Auckland, New Zealand*, Lecture Notes in Computer Science, Vol. 4801, Springer-Verlag, Berlin/Heidelberg, pp. 438–453.

Boley, H., Tabet, S. and Wagner, G. (2001). Design rationale for RuleML: A markup language for Semantic Web rules, *in* I.F. Cruz, S. Decker, J. Euzenat and D.L. McGuinness (Eds.), *Proceedings of SWWS'01, The First Semantic Web Working Symposium, Stanford University, California, USA*, pp. 381–401.

Brachman, R.J. and Levesque, H.J. (2004). *Knowledge Representation and Reasoning*, Morgan Kaufmann, San Francisco, CA.

Cañadas, J., Palma, J. and Túnez, S. (2009). InSCo-Gen: A MDD tool for Web rule-based applications, *in* M. Gaedke, M. Grossniklaus and O. Díaz (Eds.), *Web Engineering: 9th International Conference, ICWE 2009, San Sebastián, Spain*, Lecture Notes in Computer Science, Vol. 5648, Springer-Verlag, Berlin/Heidelberg, pp. 523–526.

Ceri, S., Fraternali, P. and Matera, M. (2002). Conceptual modeling of data-intensive Web applications, *IEEE Internet Computing* **6**(4): 20–30.

Chaur G. Wu (2004). Modeling rule-based systems with EMF, *Eclipse Corner Articles*, `http://www.eclipse.org/articles/`.

Dean, M., Schreiber, G., Bechhofer, S., van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-

Schneider, P.F. and Stein, L.A. (2004). *OWL Web Ontology Language Reference*, W3C Recommendation, http://www.w3.org/TR/owl-ref.

del Águila, I.M., Cañadas, J., Bosch, A., Túnez, S. and Marín, R. (2003). Knowledge model of a therapy administration task applied to an agricultural domain, *in* V. Palade, R.J. Howlett and L.C. Jain (Eds.), *Knowledge-Based Intelligent Information and Engineering Systems, 7th International Conference, KES 2003, Oxford, UK*, Lecture Notes in Computer Science, Vol. 2774, Springer-Verlag, Berlin/Heidelberg, pp. 1277–1283.

del Águila, I.M., Cañadas, J., Palma, J. and Túnez, S. (2006). Towards a methodology for hybrid systems software development, *Proceedings of the International Conference on Software Engineering and Knowledge Engineering (SEKE), San Francisco, CA, USA*, pp. 188–193.

Diouf, M., Maabout, S. and Musumbu, K. (2007). Merging model driven architecture and Semantic Web for business rules generation, *in* M. Marchiori, J.Z. Pan and C. de Sainte Marie (Eds.), *Web Reasoning and Rule Systems, First International Conference, RR 2007, Innsbruck, Austria*, Lecture Notes in Computer Science, Vol. 4524, Springer-Verlag, Berlin/Heidelberg, pp. 118–132.

Distante, D., Pedone, P., Rossi, G. and Canfora, G. (2007). Model-driven development of Web applications with UWA, MVC and JavaServer Faces, *in* L. Baresi, P. Fraternali and G.-J. Houben (Eds.), *Web Engineering, 7th International Conference, ICWE 2007, Como, Italy*, Lecture Notes in Computer Science, Vol. 4607, Springer-Verlag, Berlin/Heidelberg, pp. 457–472.

Durkin, J. (1993). *Expert Systems: Catalog of Applications*, Intelligent Computer Systems Inc., Akron, OH.

Eiter, T., Ianni, G., Krennwallner, T. and Polleres, A. (2008). Rules and ontologies for the Semantic Web, *in* C. Baroglio, P.A. Bonatti, J. Maluszynski, M. Marchiori, A. Polleres and S. Schaffert (Eds.), *Reasoning Web*, Lecture Notes in Computer Science, Vol. 5224, Springer-Verlag, Berlin/Heidelberg, pp. 1–53.

Eriksson, H. (2003). Using JessTab to integrate Protege and Jess, *IEEE Intelligent Systems* **18**(2): 43–50.

Frankel, D., Hayes, P., Kendall, E. and McGuinness, D. (2004). The model driven Semantic Web, *1st International Workshop on the Model-Driven Semantic Web (MDSW2004), Monterey, CA, USA*, http://www.sandsoft.com/edoc2004/FHKM-MDSWOverview.pdf.

Fraternali, P., Bongio, A., Brambilla, M., Comai, S. and Matera, M. (2002). *Designing Data-Intensive Web Applications*, 1st Edn., Morgan Kaufmann, San Francisco, CA.

Friedman-Hill, E. (2003). *Jess in Action: Rule-based Systems in Java*, Manning Publications, Greenwich, CT.

Gasevic, D., Djuric, D. and Devedzic, V. (2006). *Model Driven Architecture and Ontology Development*, Springer-Verlag New York, Inc., Secaucus, NJ.

Geary, D. and Horstmann, C.S. (2007). *Core JavaServer Faces*, 2nd Edn., Prentice Hall, Upper Saddle River, NJ.

Gómez-Pérez, A., Fernández-López, M. and Corcho, O. (2004). *Ontological Engineering*, Springer-Verlag New York, Inc., Secaucus, NJ.

Groenewegen, D.M., Hemel, Z., Kats, L.C.L. and Visser, E. (2008). WebDSL: A domain-specific language for dynamic Web applications, *in* G.E. Harris (Ed.), *23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, Nashville, TN, USA*, ACM, New York, NY, pp. 779–780.

Grove, R. (2000). Internet-based expert systems, *Expert Systems* **17**(3): 129–135.

Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B. and Dean, M. (2004). *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*, http://www.w3.org/Submission/SWRL/.

JBoss (2009a). Drools documentation, http://www.jboss.org/drools/documentation.html.

JBoss (2009b). RichFaces, http://www.jboss.org/jbossrichfaces/.

Karsai, G., Neema, S. and Sharp, D. (2008). Model-driven architecture for embedded software: A synopsis and an example, *Science of Computer Programming* **73**(1): 26–38.

Kifer, M. (2008). Rule interchange format: The framework, *in* D. Calvanese and G. Lausen (Eds.), *Web Reasoning and Rule Systems, Second International Conference, Karlsruhe, Germany*, Lecture Notes in Computer Science, Vol. 5341, Springer-Verlag, Berlin/Heidelberg, pp. 1–11.

Koch, N., Knapp, A., Zhang, G. and Baumeister, H. (2008). UML-based Web engineering. An approach based on standards, *in* G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.), *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, Springer, London, pp. 157–191.

Kogut, P., Cranefield, S., Hart, L., Dutra, M., Baclawski, K., Kokar, M. and Smith, J. (2002). UML for ontology development, *Knowledge Engineering Review* **17**(01): 61–64.

Kolovos, D.S. (2007). *Exeed: EXtended Emf EDitor—User Manual*, http://www.eclipse.org/gmt/epsilon/doc/Exeed.pdf.

Linaje, M., Preciado, J.C. and Sanchez-Figueroa, F. (2007). Engineering rich internet application user interfaces over legacy Web models, *IEEE Internet Computing* **11**(6): 53–59.

Mei, J., Bontas, E.P. and Lin, Z. (2005). OWL2Jess: A transformational implementation of the OWL semantics, *in* G. Chen, Y. Pan, M. Guo and J. Lu (Eds.), *Parallel and Distributed Processing and Applications—ISPA 2005 Workshops, Nanjing, China*, Lecture Notes in Computer Science, Vol. 3759, Springer-Verlag, Berlin/Heidelberg, pp. 599–608.

Mellor, S., Clark, A. and Futagami, T. (2003). Model-driven development—Guest editors' introduction, *IEEE Software* **20**(5): 14–18.

Moreno, N., Meliá, S., Koch, N. and Vallecillo, A. (2008). Addressing new concerns in model-driven Web engineering approaches, *in* J. Bailey, D. Maier, K.-D. Schewe, B. Thalheim and X.S. Wang (Eds.), *Web Information Systems Engineering—WISE 2008, 9th International Conference, Auckland, New Zealand*, Lecture Notes in Computer Science, Vol. 5175, Springer-Verlag, Berlin/Heidelberg pp. 426–442.

Moreno, N., Romero, J.R. and Vallecillo, A. (2008). An overview of model-driven Web engineering and the MDA, *in* G. Rossi, O. Pastor, D. Schwabe and L. Olsina (Eds.), *Web Engineering: Modelling and Implementing Web Applications*, Human-Computer Interaction Series, Springer, London, pp. 353–382.

Nalepa, G.J. and Ligęza, A. (2010). The HeKatE methodology. Hybrid engineering of intelligent systems, *International Journal of Applied Mathematics and Computer Science* **20**(1): 35–53, DOI: 10.2478/v10006-010-0003-9.

Object Management Group (2003a). *MDA Guide Version 1.0.1*, OMG document: omg/2003-06-01.

Object Management Group (2003b). *Meta Object Facility (MOF) 2.0 Core Specification*, OMG document: ptc/03-10-04.

Object Management Group (2008). *Semantics of Business Vocabulary and Business Rules (SBVR)*, `http://www.omg.org/spec/SBVR/1.0`.

Object Management Group (2009a). *Ontology Definition Metamodel (ODM)*, `http://www.omg.org/spec/ODM/1.0/`.

Object Management Group (2009b). *Production Rule Representation (PRR)*, `http://www.omg.org/spec/PRR/1.0/`.

O'Connor, M.J., Knublauch, H., Tu, S.W., Grosof, B.N., Dean, M., Grosso, W.E. and Musen, M.A. (2005). Supporting rule system interoperability on the Semantic Web with SWRL, *in* Y. Gil, E. Motta, V.R. Benjamins and M.A. Musen (Eds.), *International Semantic Web Conference, ISWC 2005, Galway, Ireland*, Lecture Notes in Computer Science, Vol. 3729, Springer-Verlag, Berlin/Heidelberg, pp. 974–986.

Paptaxiarhis, V., Tsetsos, V., Karali, I. and Stamotopoulos, P. (2009). Developing rule-based Web applications: Methodologies and tools, *Handbook of Research on Emerging Rule-based Languages and Technologies: Open Solutions and Approaches*, IGI Global, Hershey, PA, pp. 371–392.

Ribarić, M., Gašević, D., Milanović, M., Giurca, A., Lukichev, S. and Wagner, G. (2007). Model-driven engineering of rules for Web services, *in* R. Lämmel, J. Visser and J. Saraiva (Eds.), *Generative and Transformational Techniques in Software Engineering II, International Summer School, GTTSE 2007, Braga, Portugal*, Lecture Notes in Computer Science, Vol. 5235, Springer-Verlag, Berlin/Heidelberg, pp. 377–395.

Russell, S.J. and Norvig, P. (1995). *Artificial Intelligence: A Modern Approach*, Prentice-Hall, Inc., Upper Saddle River, NJ.

Schmidt, D.C. (2006). Guest editor's introduction: Model-driven engineering, *Computer* **39**(2): 25–31.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W.V. and Wielinga, B. (2000). *Knowledge Engineering and Management: The CommonKADS Methodology*, MIT Press, Cambridge, MA.

Wagner, G. (2002). How to design a general rule markup language?, *in* R. Tolksdorf and R. Eckstein (Eds.) *XML Technologien für das Semantic Web—XSW 2002, Proceedings zum Workshop, 24-25 Juni 2002, Berlin*, Lecture Notes in Informatics, Vol. 14, GI, Bonn, pp. 19–37.

Wagner, G., Giurca, A. and Lukichev, S. (2006). A usable interchange format for rich syntax rules integrating OCL, RuleML and SWRL, *Proceedings of the Workshop on Reasoning on the Web (RoW2006), Edinburgh, UK*.

Joaquín Cañadas received a B.S. degree in computer science from the University of Granada, Spain, in 2000. He is an assistant professor of computer science in the Department of Languages and Computation at the University of Almeria, and a member of the Data, Knowledge and Software Engineering (DKSE) research group. He currently pursues his Ph.D. under the supervision of Profs. José Palma and Samuel Túnez. His research interests are knowledge engineering, model-driven development and Web-based decision support systems.

José Palma received a B.Sc. degree from the University of Las Palmas de Gran Canaria, Spain, in 1990, and a Ph.D. degree from the University of Murcia, Spain, in 1999, both in computer science. He has been an associate professor of computer science in the Department of Information Engineering and Communications at the University of Murcia since 2000, teaching at this department as an aggregate professor since 1996. He is currently a member of the AIKE group. Prior to joining the University of Murcia, he worked for 11 years in the Department of Computer Science and Systems of the University of Las Palmas de Gran Canaria. He has authored/co-authored various journal articles, book chapters and conference papers. His research activity is focused on medical informatics, specifically intelligent data analysis, medical knowledge-based systems and clinical knowledge management. The main techniques used in this research are fuzzy-logic, knowledge engineering methodologies, ontologies and temporal reasoning.

Samuel Túnez received a Ph.D. degree in physics from the University of Santiago de Compostela, Spain. He is an associate professor of computer science in the Department of Languages and Computation at the University of Almeria, and the head of the Data, Knowledge and Software Engineering (DKSE) research group. He has participated as the head researcher in several national and international projects. His research interests focus on knowledge-based systems, knowledge engineering, information systems and, in particular, the application of these techniques to the design of decision support systems in agricultural production.

Received: 15 April 2010
Revised: 22 November 2010