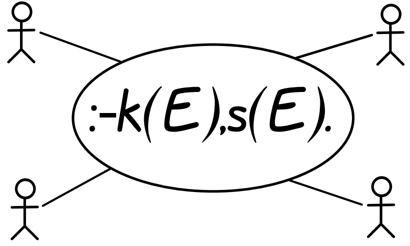
5th Workshop on

Knowledge Engineering and Software Engineering (KESE2009)

at the 32nd German Conference on Artificial Intelligence

September 15, 2009, Paderborn, Germany

Joachim Baumeister and Grzegorz J. Nalepa (Editors)



Technical Report No. 461, Würzburg University, Würzburg, Germany, 2009

The KESE Workshop Series is available online: https://ai.ia.agh.edu.pl/wiki/kese:start

Preface

Joachim Baumeister and Grzegorz J. Nalepa

Intelligent Systems (Informatik 6)
University of Würzburg
Würzburg, Germany
joba@uni-wuerzburg.de
—
AGH University of Science and Technology
Kraków, Poland
gjn@agh.edu.pl

Intelligent systems have been successfully developed in various domains based on techniques and tools from the fields of knowledge engineering and software engineering. Thus, declarative software engineering techniques have been established in many areas, such as knowledge systems, logic programming, constraint programming, and lately in the context of the Semantic Web and business rules.

The fifth workshop on Knowledge Engineering and Software Engineering (KESE 2009) was held at the KI-2009 in Paderborn, Germany, and brought together researchers and practitioners from both fields of software engineering and artificial intelligence. The intention was to give ample space for exchanging latest research results as well as knowledge about practical experience. Topics of interest includes but were not limited to:

- Knowledge and software engineering for the Semantic Web
- Ontologies in practical knowledge and software engineering
- Business rules design and management
- Knowledge representation, reasoning and management
- Practical knowledge representation and discovery techniques in software engineering
- Agent-oriented software engineering
- Database and knowledge base management in AI systems
- Evaluation and verification of intelligent systems
- Practical tools for intelligent systems
- Process models in AI applications
- Declarative, logic-based approaches
- Constraint programming approaches

This year, we mainly received contributions focussing on the "intelligent web": Pascalau and Giurca introduce a rule engine for web browsers, that is capable to handle DOM (Document Object Model) events within the browser. Cañadas et al. describe an approach for the automatic generation of rule-based web applications, that is based on ideas of the Model Driven Development (MDD). Nalepa and Furmańska propose an ontology that maps the design process of an intelligent application and thus promises efficient development. Reutelshoefer et al. show, how multimodal knowledge appears in knowledge engineering projects and show how such knowledge can be refactored within a Semantic

Wiki. The intelligibility of medical ontological terms is discussed and evaluated by Forcher et al. This year we also encouraged to submit tool presentations, i.e., system descriptions that clearly show the interaction between knowledge engineering and software engineering research and practice. At the workshop, two presentations about current tools were given: Kaczor and Nalepa introduced the toolset HaDEs, i.e., the design environment of the HeKatE methodology. Pascalau and Giurca show-cased the JavaScript rule engine JSON Rules.

The organizers would like to thank all who contributed to the success of the workshop. We thank all authors for submitting papers to the workshop, and we thank the members of the program committee as well as the external reviewers for reviewing and collaboratively discussing the submissions. For the submission and reviewing process we used the EasyChair system, for which the organizers would like to thank Andrei Voronkov, the developer of the system. Last but not least, we would like to thank Klaus-Dieter Althoff (U. Hildesheim) as the workshop chair and Bärbel Mertsching (U. Paderborn) as the KI09 conference chair for their efforts and support.

Joachim Baumeister
Grzegorz J. Nalepa

Workshop Organization

The 5th Workshop on Knowledge Engineering and Software Engineering
(KESE2009)
was held as a one-day event at the
32nd German Conference on Artificial Intelligence (KI2009)
on September 15, 2009 in Paderborn, Germany.

Workshop Chairs and Organizers

Joachim Baumeister, University Würzburg, Germany
Grzegorz J. Nalepa, AGH UST, Kraków, Poland

Programme Committee

Klaus-Dieter Althoff, University Hildesheim, Germany
Stamatia Bibi, Aristotle University of Thessaloniki, Greece
Joaquin Cañadas, University of Almería, Spain
Uli Geske, FhG FIRST, Berlin, Germany
Adrian Giurca, BTU Cottbus, Germany
Rainer Knauf, TU Ilmenau, Germany
Frank Puppe, University Würzburg, Germany
Dietmar Seipel, University Würzburg, Germany
Ioannis Stamelos, Aristotle University of Thessaloniki, Greece
Gerhard Weiss, SCCH, Austria

External Reviewers

Jochen Reutelshoefer, University Würzburg, Germany

Table of Contents

Regular Papers.

A Lightweight Architecture of an ECA Rule Engine for Web Browsers . . . 1
Emilian Pascalau, Adrian Giurca

A Model-Driven Method for automatic generation of Rule-based Web Applications . . . 13
Joaquín Cañadas, José Palma, Samuel Túnez

Design Process Ontology - Approach Proposal . . . 25
Grzegorz J. Nalepa, Weronika T. Furmańska

A Data Structure for the Refactoring of Multimodal Knowledge . . . 33
Jochen Reutelshoefer, Joachim Baumeister, Frank Puppe

Evaluating the Intelligibility of Medical Ontological Terms . . . 46
Björn Forcher, Thomas Roth-Berghofer, Kinga Schumacher

Tool Presentations.

HaDEs - Presentation of the HeKatE Design Environment . . . 57
Krzysztof Kaczor, Grzegorz J. Nalepa

JSON Rules - The JavaScript Rule Engine . . . 63
Emilian Pascalau, Adrian Giurca

A Lightweight Architecture of an ECA Rule Engine for Web Browsers

Emilian Pascalau[1] and Adrian Giurca[2]

[1]Hasso Plattner Institute, Germany,
emilian.pascalau@hpi.uni-potsdam.de
[2]Brandenburg University of Technology, Germany,
giurca@tu-cottbus.de

Abstract. There is a large literature concerning rule engines (forward chaining or backward chaining). During the last thirty years there were various proposals such as RETE, TREAT and the derived Gator algorithm. Significantly, RETE was embedded into various expert systems such as Clips and its successor Jess, and Drools including in a number of commercial rule engines and was extended various times including with support for ECA rules. However, none of them is able to directly process DOM Events. The goal of this paper is to present the architecture of a forward chaining Event-Condition-Action (ECA) rule engine capable to handle Document-Object-Model Events. This architecture is instantiated into a JavaScript-based rule engine working with JSON rules.

1 Motivation

There is a large literature concerning rule engines (forward chaining or backward chaining). During the last thirty years there were various proposals such as RETE [6], TREAT [15] and the Gator algorithm [13] which is derived from the other two. Significantly, RETE was embedded into various expert systems such as Clips and its successor Jess[7], and Drools [18]. RETE [6] was extended various times including with support for ECA rules [5].

However, none of them is able to directly process DOM Events. The goal of this paper is to present the architecture of a forward chaining ECA rule engine capable to handle Document-Object-Model Events. This architecture is instantiated into a JavaScript-based rule engine working with JSON rules [10].

The main goals this design should address are:

– to move the reasoning process to the client-side resulting in reduced network traffic and faster response;
– to handle complex business workflows;
– information can be fetched and displayed in anticipation of the user response;
– pages can be incrementally updated in response to the user input, including the usage of cached data;
– to offer support for intelligent user interfaces;
– enable users to collaborate and share information on the WWW through real-time communication channels (rule sharing and interchange);

Complex event processing (CEP), is a methodology of processing events taking into consideration processing multiple events with the goal of identifying the meaningful events within a specific time-frame or event cloud. CEP employs techniques such as *detection of complex patterns*, *event correlation*, *event abstraction*, *event hierarchies*, and *relationships between events* such as causality, membership, and timing, and event-driven processes. A number of projects were developed in the last ten years on these issues. However, there is one event ontology which offers large opportunities to be exploited in the context of actual technologies such as Asynchronous JavaScript and XML (AJAX) [8] allowing the development of intelligent Rich Internet Applications (RIAs) i.e. web applications that typically run in a web browser, and do not require software installation ([1]) - The Document Object Model Events (DOM Events). This event ontology[1] provides a large amount of events types designed with two main goals: (1) the design of an event system allowing registration of event listeners and describing event flow through a tree structure (the DOM), and (2) defining standard modules of events for user interface control and notifications of document mutation, including defined contextual information for each of these event modules.

This ontology is already implemented into browsers giving extremely powerful capabilities to RIAs which use it. Nowadays, several Web 2.0 applications use heavily AJAX in order to provide desktop-like behavior to the user. The number of RIAs is increasing because of the broad bandwidth of today's Internet connections, as well as the availability of powerful and cheap personal computers. However, traditional ways of programming Internet applications no longer meet the demands of intelligent (rule-enabled) RIAs. For example a highly responsive Web 2.0 application such as Google Mail, can be much easily personalized/customized using rules towards a declarative description of its behavior.

Implementing intelligent RIAs require reasoning possibilities inside the browser. In addition, using Event-Condition-Action (ECA) Rules to represent knowledge unveils the opportunity to design and run rule-based applications in the browser.

2 The Architecture of an ECA Rule Engine for Web Browsers

2.1 The Components View

As depicted in Figure 1, the complete system comprises the *Event Manager*, *Rule Repository*, *Inference Engine* and *Working Memory.*

The main design goal of this architecture was to comply with the principles of Software as a Service (SaaS) architectures [4]. Therefore, the main capabilities considered in this design were:

- *Distributed Architecture* - all these components can act in different network locations.

[1] `http://www.w3.org/TR/DOM-Level-3-Events/`

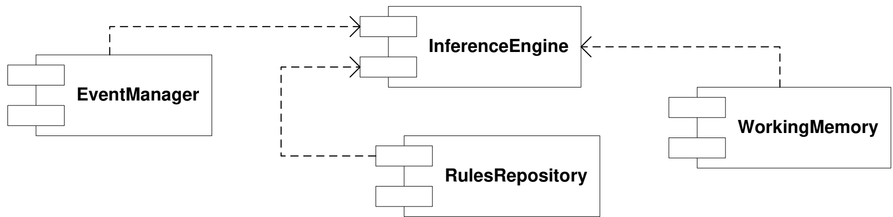


Fig. 1. Components View

– *Event-driven architecture* - We emphasize that both human agents and software agents interact with this architecture by creating events i.e. the reasoning is event driven. Moreover, the architecture instantiation gets translated into a full event driven engine.

This architecture is a live system i.e. an event-based system that is reactive and proactive. It is reactive because it reacts based on the events it receives. It is proactive because by itself generates events, that can be consumed also by other entities being part of the whole system.

2.2 The Working Memory

In the database community the main goal of designing ECA engines was to provide generic functionality independent from the actual languages and semantics of event detection, queries, and actions (see for example, [3] and [19]). However, two main issues make the difference: (a) in the case of an ECA architecture the Working Memory besides the usual standard facts it contains also event-facts and (b) the distinction between facts and event-facts is that the last ones are immediately consumed while traditionally facts are kept until specific deleting actions are performed.

2.3 The Event Manager

During the last years there is an intense work either on defining design patterns for complex event processing [17] or theoretical work on how Event-Condition-Action rules can be combined with event algebras for specification of the event part [2].

Our goal is to provide a light *Event Manager* capable to process faster simple events without duration. Particularly, this architecture must handle DOM Level 3 Events. However, the extension points in the *Event Manager* make possible future extensions for complex events processing if we will be able to provide motivating use cases.

Basically the *Event Manager* (depicted in Figure 2) has an event vocabulary and listen for events. Its main activity is to create an event queue to be processed by the inference engine.

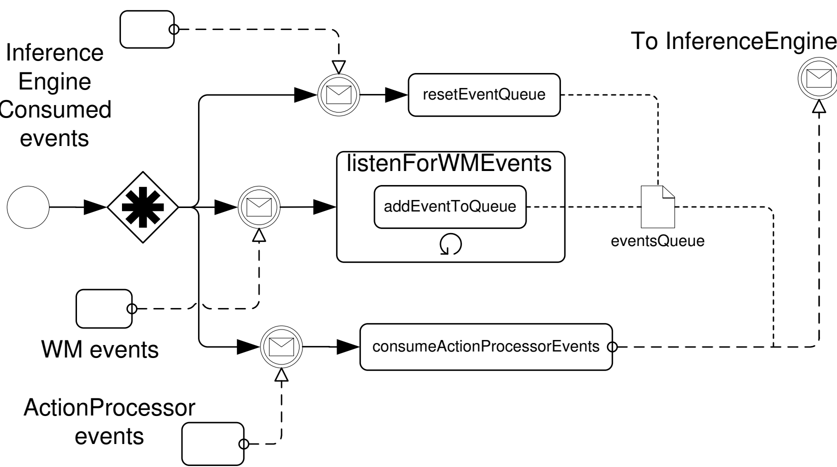


Fig. 2. EventManager

The *Event Manager* keeps on catching Working Memory (WM) event-facts and stores them in an event queue. In addition, it listens for two internal *Action Processor* messages:

- *busy* - The *Event Manager* keeps on catching WM events and storing them in the working queue of events.
- *idle* - The *Action Processor* informs that it is not working right now. The *Event Manager* pro-actively takes control and send its own message to the *Inference Engine* with the actual working queue of events. Each time an event is caught by the *Event Manager* it tries to find out about the state of the *Action Processor*. If there is no new message from the *Action Processor* it keeps going on based on its actual knowledge of the *Action Processor* state. It changes its knowledge when it receives a new message from the *Action Processor*.

Finally, the manager handles the inference engine consumed events. Our model looks for mandatory handling of engine consumed events as the default mechanism to achieve the event consumption. Therefore if there are events which are not processed/consumed by the inference engine they are kept by the manager on its lifetime or until they are consumed by rules.

2.4 The Inference Engine

Our goal were not to use RETE and its variants (although influences exist) but to build a lightweight engine. Our goals were not to embed strong efficient execution algorithms rather to offer a simple, extensible and fast rule execution engine. All these design goals were coming from our main goal: *running rules in the Web browser*.

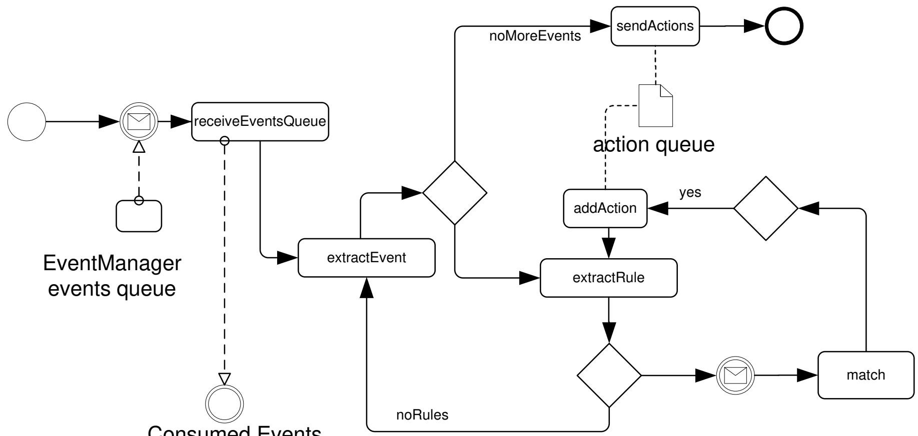


Fig. 3. InferenceEngine

The basic activities inside the *Inference Engine* (see the Figure 3) are to *consume events* (from the events queue delivered by the *Event Manager*) *match rule conditions* (match) and *deliver action queue* to the *Action Processor* (`sendActions`). Despite other architectures where the actions are consumed inside the inference engine, we decided for a separate component since the *Action Processor* is not just a blind action executor but is able to perform various consistency checks after it has received its queue of actions. The rules intended to be handled by this architecture are JSON Rules [10] (see 1 for a small rule example) which provide priorities for handling rule order execution. Our engine does not provide any conflict resolution mechanism i.e. does not handle any specificity, recency or refractoriness principles, but it can be extended to support such mechanisms. Finally the engine has no formal semantics such as other expert systems paradigms [14]. The syntax of a JSON rule is similar to JSON notation.

Example 1 (JSON Rule example).

```
{"id": "rule101",
  "appliesTo": ["http://mail.yahoo.com/"],
  "eventExpression": {"type": "click",
                      "target": "$X"
                     },
  "condition": [
                 "$X:HTMLAnchorElement($hrefVal:href)",
                 "new RegExp(/showMessage\?fid=Inbox/).test($hrefVal)"
               ],
  "actions":["append($X.textContent)"]
}
```

2.5 The Rules Repository

As we already know, the purpose of business rules repositories is to support the business rule information needs of all the rule owners in a business rules-based

approach in the initial development of systems and their lifetime enhancement. Specifically, the purpose of a business rules repository is to provide: (a) Support for the rule-related requirements of all business, (b) Query and reporting capability for impact analysis, traceability and business rule reuse including web-based publication and (c) Security for the integrity of business rule and rule-related information.

Parts of our previous work (see for example, [16]) introduced the architecture of such a registry. Basically, inside this architecture, the *Rules Repository* is responsible to handle loading and deploying of rule sets.

3 JSON Rules - Architecture Instantiation

We introduce the instantiation of our architecture in the JSON Rules context. Recall from [10] that JSON Rules where introduced and defined to tackle a particular environment which is the Web Browser. While the first part of this work addresses the architectural issues from the Platform-Independent Model (PIM) [12], [9] perspective, this part addresses it from the Platform-specific Model (PSM) perspective.

According to the reference architecture for Web Browsers introduced in [11] the system introduced here finds itself as part of the *Rendering Engine*. In the general perspective the JSON Rules engine will come as part of the accessed resource.

In the case of the Mozilla [2] browser's architecture [11] the system might be either part of the *Rendering Engine* or part of the UI Toolkit (XPFE[3] - Mozilla's cross-platform front end) if the system is packed as a browser add-on. The second approach gives greater flexibility since the UI of Mozilla browsers is XML based and as such uses an extended version of the rendering engine used to display the content of a specified resource. Based on this the JSON Rules engine introduced here seams quite feasible to be used to change also the UI and behavior of the browser itself.

The general components view depicted in Figure 1 gets instantiated in the JSON Rules context as depicted in Figure 4.

Depicted in Figure 4 are the main packages of the JSON Rules engine. While the `engine` and `repository` packages are self explanatory to some extent `lang` package contains all the JSON Rules language entities. The `utils` package contains entities dealing with different aspects such as: JSON parsing, or object introspection and so on. The `io` package provides the necessary entities managing IO operations. The `engine` package contains the following sub-packages: `eventmanager`, `actionprocessor`, and `matcher`. The general flow of the whole system is described in the Figure 5 (initially introduced in [10]).

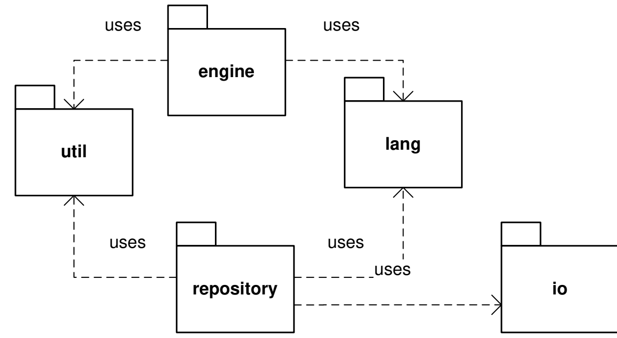


Fig. 4. Rule Engine - Packages

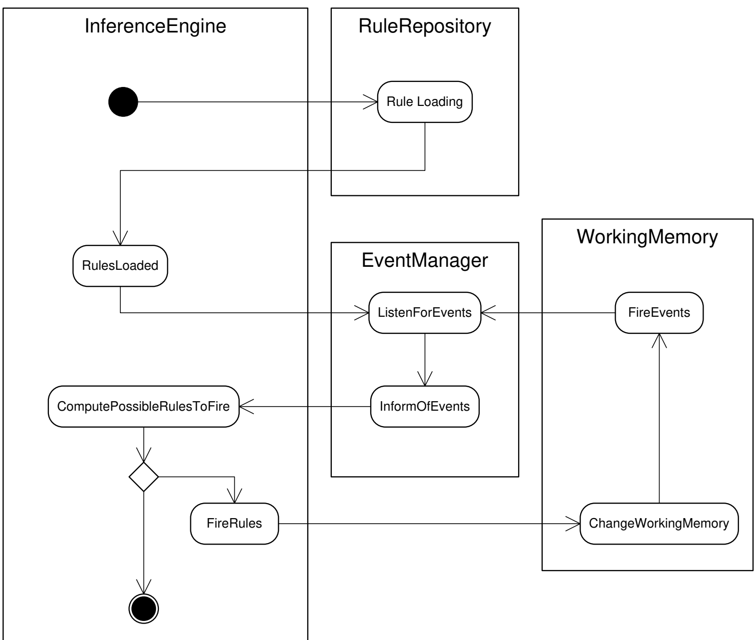


Fig. 5. The Rule Engine State Diagram

3.1 Main System

The `MainSystem` (Figure 6) is the interface through which the inference engine is accessed. As seen in the Figure 6 the only mandatory input is an Uniform Resource Identifier (URI) pointing towards a rule repository, from where rules will be loaded.

[2] `http://www.mozilla.com/`
[3] `http://www.mozilla.org/xpfe/`

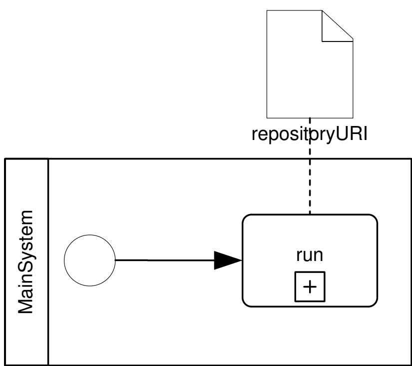


Fig. 6. MainSystem

Although not concretely depicted here through this interface a user could also specify other settings such as different event type for which the event manager should listen for.

Having specified a repository location the `MainSystem` performs the `run` activity.

The lifetime of the rule engine is in the scope of the lifetime of the current DOM inside the browser. Using the engine is simple. Firstly one must load the engine e.g. this

```
<script type="text/javascript"
  src="http://www.domain.com/jsonRulesEngine_Version.js">
</script>.
```

Secondly one must create an instance of the engine, for example:

```
var jsonRulesEngine=new org.jsonrules.JSONRulesMainSystem();
```

Having the engine instantiated, it is now possible to run it by calling `run()` with the URI of location of the repository as input parameter:

```
jsonRulesEngine.run("http://www.domain.com/rulesRepo.txt");
```

In a basic application the main steps that happen are:

- When an event is raised, the `EventManager` catches that event. Then the `EventManager` checks the `ActionProcessor`'s state.
- If the `ActionProcessor` is running, then `EventManager` stores the event in the queue of events that the `InferenceEngine` must later on process.
- However if the `ActionProcessor` is idle then the `EventManager` sends a message to the `InferenceEngine` containing the queue of events that must be processed. The `InferenceEngine` responds back to the `EventManager`, and informs it that it has received/consumed the queue such that the `EventManager` can reset its own queue.

– Events are processed one by one. For each event rules triggered by that event will be matched against the `WorkingMemory`. The action of each executable rule is added to the list of executable actions (to be processed by the `ActionProcessor`) according with possible priority of rules.
– The list of executable actions it is send to the `ActionProcessor`, to execute them.

3.2 Working Memory

As already introduced in [10] the `WorkingMemory` consists of the loaded DOM for the current active resource. Recall that by resource we mean the content which a browser loads in the form of a DOM representation from a specified URI. `WorkingMemory` facts are based on the DOM content. Moreover, in the context of our architecture, `WorkingMemory is` driven by events and contains event-facts. This type of behavior is imposed by the event-based nature of the DOM.

3.3 Event Manager

In addition to DOM Level 3 Events, the DOM specification provides the necessary interfaces through which an user-defined event can be created. However, in general, DOM events are simple events even though users could create their own events. There is also the possibility to use and define complex events by means of user defined APIs such as Yahoo YUI[4], Dojo toolkit[5] etc. To deal with such user defined APIs the `EventManager` uses the concept of adapter. An adapter can be written for each API and in this way events defined using those APIs could also be tackled by the `EventManager`.

Another significant aspect of the browser based instantiation is that the whole flow is by nature sequential. Actual browsers' JavaScript engines are sequential, and because of this, so is the whole engine introduced here. However in the eventuality of a browser with capabilities to run parallel JavaScript tasks then the general architecture could be instantiated following the ability to run parallel tasks.

3.4 Inference Engine

Figure 7 depicts the interaction between the `MainSystem` and the `InferenceEngine`. The `InferenceEngine` receives a `page` object form the `MainSystem`. Its subcomponents (`EventManager`, `ActionProcessor`, `Matcher`) are also instantiated and in this manner the system becomes alive by listening and throwing events.

3.5 Rule Repository

While a more detailed perspective on rule repositories has been already introduced in [16] here we use a simplified version of that. Rules defined in the repos-

[4] `http://developer.yahoo.com/yui/`
[5] `http://www.dojotoolkit.org/`

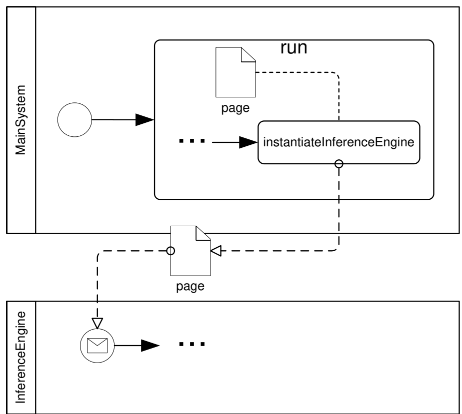


Fig. 7. System-InferenceEngine

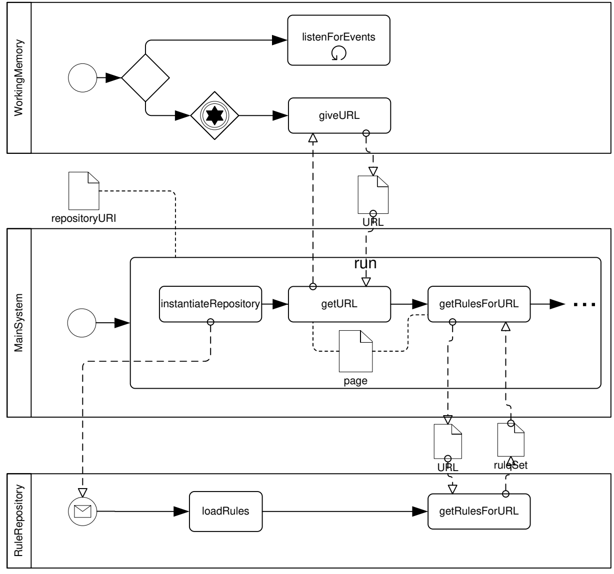


Fig. 8. RuleRepository

itory refer to a specific URI. This means that a specific rule can be used in the context of a specific resource. Rules referring to the same URI are grouped in rule sets.

Figure 8 depicts the interaction between the `MainSystem` and the `RuleRepository`. Basically, the `MainSystem` triggers a `RuleRepository` instance. The repository loads the rules from the `repositoryURI` specified location. Readers may notice that the repository might contain rules that do not refer to the current active resource. As such the `MainSystem` requests the URI of the current resource from the `WorkingMemory`. Based on that URI it requests from the repository the rule set referring to the current resource. Finally, based on this information (i.e. the URI of the current resource and the rule set associated) the `MainSystem` creates a `Page` object which will be used by the `InferenceEngine`.

4 Conclusion

This paper describes the general architecture of an ECA rule-based and forward chaining engine for web browsers. The design of such an engine derives from the goal to perform intelligent RIAs. The instantiation of the architecture results in a JavaScript-based ECA rule engine capable to load and execute ECA rule sets in the browser. This way we achieve a main goal: Implementing intelligent RIAs require reasoning possibilities inside the browser. The next steps related to this research are: (1) to investigate the capabilities of this engine to handle rule-based mashups on the Web and (2) to analyze scalability of the engine against the main browsers.

References

1. Jeremy Allaire. Macromedia Flash MXA next-generation rich client. `http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf`, March 2002.
2. Erik Behrends, Oliver Fritzen, Wolfgang May, and Franz Schenk. Embedding Event Algebras and Process for ECA Rules for the Semantic Web. *Fundamenta Informaticae*, 82(3):237–263, 2008.
3. Erik Behrends, Oliver Fritzen, Wolfgang May, and Daniel Schubert. An ECA Engine for Deploying Heterogeneous Component Languages in the Semantic Web. In *Current Trends in Database Technology - EDBT Worshops*, pages 887–898, 2006.
4. Keith Bennett, Paul Layzell, David Budgen, Pearl Brereton, Linda Macaulay, and Malcolm Munro. Service-Based Software: The Future for Flexible Software. In *Proceedings of the Seventh Asia-Pacific Software Engineering Conference (APSEC2000)*, pages 214 – 221. IEEE Computer Society, 2000. `http://www.bds.ie/Pdf/ServiceOriented1.pdf`.
5. Bruno Berstel. Extending the RETE Algorithm for Event Management. In *TIME*, pages 49–51, 2002.
6. Charles Forgy. Rete – A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.
7. E. Friedman-Hill. Jess The Rule Engine for the Java Platform. `http://www.jessrules.com/jess/docs/Jess71p2.pdf`, November 2008.

8. Jesse James Garrett. Ajax: A new approach to web applications. http://www.adaptivepath.com/ideas/essays/archives/000385.php, February 2005.
9. Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Architecture and Ontology Development*. Springer Verlag, 2006.
10. A. Giurca and E. Pascalau. JSON Rules. In *Proceedings of the Proceedings of 4th Knowledge Engineering and Software Engineering, KESE 2008*, volume 425, pages 7–18. CEUR Workshop Proceedings, 2008.
11. Alan Grosskurth and Michael W. Godfrey. A Reference architecture for web browsers. In *Proceedings of the 21st IEEE international conference on software maintenance (ICSM'05)*, page 661664. IEEE Computer Society, 2005. http://grosskurth.ca/papers/browser-archevol-20060619.pdf.
12. Object Management Group. MDA Guide Version 1.0.1. http://www.omg.org/docs/omg/03-06-01.pdf, 2003.
13. E. Hanson and M. Hasan. Gator: An optimized discrimination network for active database rule condition testing. Technical report, 1993.
14. Antoni Ligeza. *Logical Foundations for Rule-Based Systems*, volume 11 of *Studies in Computational Intelligence*. Springer Verlag, 2nd edition edition, 2006.
15. D. Miranker. Treat: A better match algorithm for AI production systems. In *Proceedings of the AAAI'87 Conference*, 1987.
16. Emilian Pascalau and Adrian Giurca. Towards enabling SaaS for Business Rules. In *Business Process, Services Computing and Intelligent Service*, pages 207–222, 2009. http://bpt.hpi.uni-potsdam.de/pub/Public/EmilianPascalau/ism2009.pdf.
17. Adrian Paschke. Design Patterns for Complex Event Processing. *CoRR*, abs/0806.1100, 2008.
18. Mark Proctor, Michael Neale, Michael Frandsen, Sam Griffith Jr., Edson Tirelli, Fernando Meyer, and Kris Verlaenen. Drools 4.0.7. http://downloads.jboss.com/drools/docs/4.0.7.19894.GA/html_single/index.html.
19. Marco Seiriö and Mikael Berndtsson. Design and Implementation of an ECA Rule Markup Language. In *RuleML*, pages 98–112, 2005.

A Model-Driven Method for automatic generation of Rule-based Web Applications

Joaquín Cañadas[1], José Palma[2] and Samuel Túnez[1]

[1] Dept. of Languages and Computation. University of Almeria. Spain
jjcanada@ual.es, stunez@ual.es
[2] Dept. of Information and Communication Engineering. University of Murcia. Spain
jtpalma@um.es

Abstract. Rule languages and inference engines incorporate reasoning capabilities in Web information systems. In this paper, a Model-Driven Development (MDD) approach for automatic code generation of rule-based Web applications is proposed. A rule-based model specifying domain expert knowledge and business logic through production rules (if-condition-then-action) becomes the source model for the development approach. Demonstrating our proposal, a tool supports the creation of rule models and the automatic execution of model-to-model and model-to-code transformations. As a result, a rich, functional, rule-based Web architecture is generated, based on the Model-View-Controller architectural pattern and the JavaServer Faces technology, and integrating a Jess rule engine to perform inference tasks.

Key words: Model-Driven Development, Web Applications, Rule-based systems

1 Introduction

The design of rule languages and inference engines to provide Web information systems with reasoning capabilities is an important Semantic Web research topic [1] in which production rules (if-condition-then-action) play a leading role, since they enable a declarative representation of domain expert knowledge and business logic. Rule engines deal with rule bases and execute inference methods for firing the right rules in order to deduce information and conclude new results [2].

This paper addresses the development of rule-based systems embedded in Web applications to provide Web systems with inference capabilities, applying a model-driven approach to automate the development process of rule-based Web applications.

The terms Model-Driven Architecture (MDA) [3] and Model-Driven Development (MDD) [4] refer an approach of software development that uses models as first class entities, enabling the definition and automatic execution of transformations between models and from models to code. The creation of metamodels for specifying modeling languages is a basic task in MDA/MDD. Also the

specification of transformations between models, called model-to-model (M2M) transformations, and from model to code, called model-to-text (M2T) transformations. The main advantage of this approach of software development is that MDD tools enable these transformations to be specified and executed automatically, using supporting languages and tools for MDA/MDD. This development approach is currently being applied to many domains in software development, such as embedded systems, Web engineering, Ontology Engineering, and more. However it has some limitations because it is relatively new, supporting tools for MDD are not mature enough, and it introduces some rigidity since writing models is not as flexible and expressive as writing source code.

In this work Conceptual Modeling Language (CML) [5] is used as rule modeling formalism, a language for knowledge representation defined by the CommonKADS methodology [6]. It enables the specification of the domain ontology and a set of production rules which are bound to ontology concepts. 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.

To put our proposal into practice, a supporting tool developed using several tools provided by the Eclipse Modeling Project[3] applies a model-driven approach to rule models and automatically generates the implementation of a functional rule-based Web application. The resulting Web architecture is based on the Model-View-Controller (MVC) architectural pattern and the JavaServer Faces (JSF) framework [7], and incorporates rich JBoss Richfaces components [8] to enhance the user interface with AJAX (Asynchronous JavaScript And XML) capabilities. The Jess rule engine [9] is embedded in the Web architecture to provide inference features. The functionality of the rule-based Web application is predefined to create, read, update and delete instances (CRUD). In contrast to current tools for automatic generation of CRUD systems that perform those functions on relational databases, the contribution of our approach is that CRUD operations are executed on the Jess rule engine working memory, enabling the inference mechanism to execute a forward-chaining inference mechanism to drive the reasoning process.

The proposed approach materializes *InSCo* [10], a methodology which intertwines knowledge engineering and software engineering approaches in hybrid intelligent information systems development.

This paper is organized as follows: Section 2 introduces rule-based systems and rule modeling languages for the Web. Next, the rule-based modeling approach for specifying the Web applications proposed is described in section 3. After that, the model-driven method for rule-based Web application development is detailed in Section 4. The MDD support tool is presented in Section 5. Section 6 describes related work, and finally main conclusions and future work are summarized.

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

2 Overview of Rule-based systems and rule modeling

Rule-based systems originated in Artificial Intelligence, as the kind of expert or knowledge-based system that use rules as knowledge representation formalism. In this kind of software system, the human expert's knowledge applied for solving a complex task such as diagnosis, monitoring, assessment, and so on, 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 as the human expert would do [11, 12].

In general, a rule-based system consists of a set of production rules, a working memory and an inference engine. The rules encode domain knowledge and business logic as condition-action pairs. The working memory initially represents the system input, 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. The execution of the action part of a rule involves inferring new data.

More recently, 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 rule-based systems and business rules in the implementation of complex decision-making processes [13].

Rule formalisms are an active area of research addressing the development rule languages and inference engines to add reasoning to complex information systems. The Object Management Group (OMG) proposed the Ontology Definition MetaModel [14] and Production Rule Representation [15] as standard metamodels for introducing both technologies in the context of MDA/MDD. Relevant initiatives to standardize and exchange rules are the Rule Markup Initiative (RuleML) [16], the Semantic Web Rule Language (SWRL) [17], the REWERSE Rule Markup Language (R2ML) [18], and the Rule Interchange Format (RIF) [19].

We use CML as the rule-modeling language because, although it is currently not one of the most common options for rule modeling, it has several features desirable for production-rule formalisms. It enables unified representation of ontologies and rules, in which rules and ontology are naturally related. It meets the requirements of rule representation formalisms, such as modeling rule antecedent, rule consequent, named rules, and rulesets, binding rules to ontology concepts, and so on. And finally, it is simpler and easier to use than other formalisms, although this may mean less expressiveness in certain situations.

3 Modeling Rule-based Web applications

The proposed model-driven approach for rule-based Web application development focuses on introducing rule modeling in the specification of Web applications. However, other modeling concerns related to Web design features must be also considered, powering the automatic code generation process. The CML

model describing the ontology and rule model is presented at a conceptual level, whereas interaction and presentation features are specified at a Web design level.

3.1 Conceptual rule-based modeling

The CML formalism for knowledge modeling entails the specification of simplified domain ontologies and production rules. A CML (domain knowledge) model is basically composed of two elements, domain schemas and knowledge bases. Domain concepts, binary relationships, rule types and value types (enumerated literals) are modeled in a domain schema. A knowledge base is composed of instances of concepts, instances of relationships called tuples, and instances of rules. Figure 1 shows the domain knowledge model components.

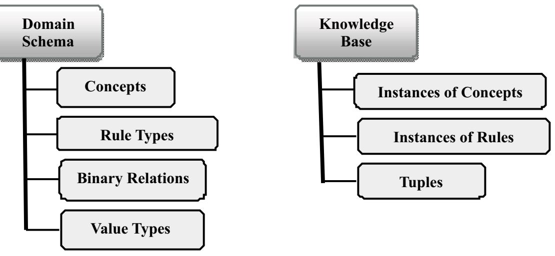


Fig. 1. Domain Knowledge structure

CML was originally defined as a textual notation by means of an abstract grammar described in EBNF (Extended Backus-Naur Form). To use CML in the context of MDD, we have specified a metamodel for CML. The main difference between this formalism and other conceptual modeling approaches in software engineering, such as UML class diagrams, is its ability to model production rules with rule type and the rule instance constructors. A rule type describes the structure of a set of rules through the specification of the ontology types bound to the rule antecedent and consequent. Rule types are particularized into rule instances which represent specific, logical dependencies between rule antecedent and consequent concept attributes.

3.2 Web design modeling

CML models are enriched with interaction and presentation characteristics to specify rule-based Web applications design features.

Interaction features enable the specification of user interactivity through a set of properties associated to CML constructors. The following properties dealing with attribute management will illustrate some interaction characteristics:

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

– *notifiesTo* and *isNotifiedBy*: These properties are used to indicate what attributes must be refreshed by the re-rendered AJAX facility in a user event, for example a mouse click or a change in the attribute value.

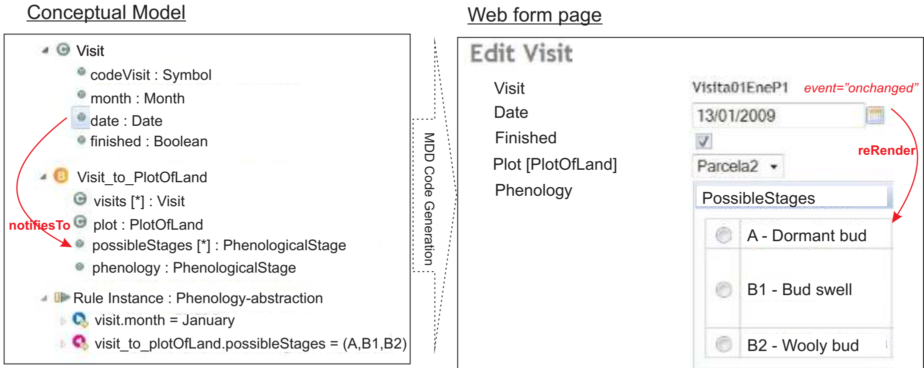


Fig. 2. Modeling interaction with *NotifiesTo* property

Figure 2 shows an example of how the interaction between two attributes defined in the conceptual model is specified using the *isDerived* and *notifiesTo* properties, showing how they affect Web forms for editing instances. The example is taken from SAVIA, a decision-support system for pest control in greenhouse crops and grapes that is being developed by applying rule-based modeling and the proposed model-driven approach for rule-based Web system development. The left side of Figure 2 shows a selection of SAVIA conceptual rule model elements. In particular, a concept called *Visit*, a relationship called *Visit_to_PlotOfLand*, and an example of rule instance belonging to the group of rules that specify the possible phenological stages of the crop depending on the date of the visit. The concrete rule instance is: *"if the month of the Visit is January then the possibleStages are (A, B1, B2)"*. Focusing on interaction specification, the *possibleStages* attribute is derived since it is inferred by the rule engine when it fires rules such as the one above. And the attribute *date* of visit notifies *possibleStages*, making that when the event *onChange* happens in the Web form date field, then an action makes the rule engine run and the list of *possibleStages* is re-rendered, updating the list with the new values determined 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 what concepts will appear in the application menu, and what attributes are included as columns of tables showing all instances of a concept type.

4 MDD for Rule-based Web applications

4.1 General perspective

Figure 3 shows the proposed MDD schema for rule-based Web applications, which is divided into two processes. The first one (the bottom flow in Fig. 3) generates the implementation of the rule base in a rule implementation technology, and the second one (the top flow in Figure 3) produces the code for the Web architecture.

The development process starts with the specification of a conceptual rule model which defines the domain ontology and the set of rules using an platform-independent formalism such as CML. Application of the model-driven approach produces two different results. One one hand, ontology and rules are transformed into Jess, which supports the development and deployment of rule-based systems tightly coupled to Java applications. As a result, a Jess rule base, 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 with the interaction and presentation features. Web application code is based on the MVC architectural pattern and the JavaServer Faces (JSF) framework, producing a set of JavaBeans classes and JSP (Java Server Pages).

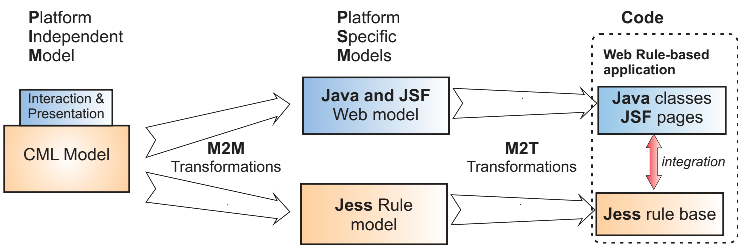


Fig. 3. MDD schema for Rule-based Web system generation

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, entailing integration of the rule engine into the Web application.

The rule-based Web application generated benefits of having the decision logic externalized from core application code, since uncoupling the rules from the source code increases scalability and maintainability of rule-based applications [20]. Our approach makes it possible for the two MDD processes to be executed separately, and therefore, any change in the rule model affecting only to rule logic (rule instances) but without affecting to the structure of information (concepts, relationships, and so on) can be translated to 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.

4.2 MDD of Jess rules

The first transformation of Jess rules in MDD involves the CML source model being translated into a platform-specific model based on a Jess rules metamodel, using an M2M transformation. The metamodel proposed for Jess rules (Figure 4) is an extended version of a simple rule metamodel for rule-based systems described in [21].

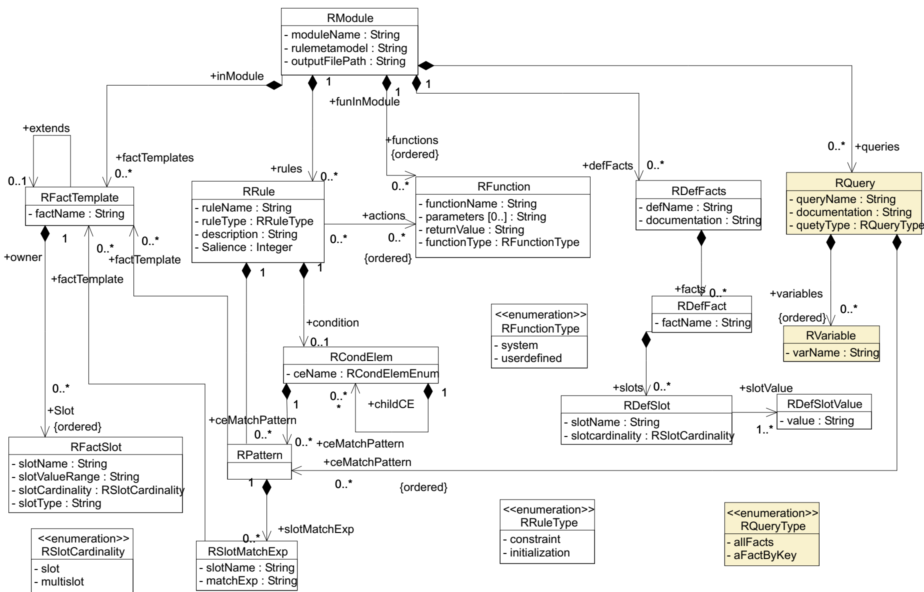


Fig. 4. Jess Rules Metamodel

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 a *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 part (antecedent), and a reference called *actions* representing the rule's action part (consequent). *RPattern* and *RSlotMatchExp* metaclasses define pattern matching expressions in rule conditions. Actions are function calls that assert new facts, or retract or modify existing facts.

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

The mapping from CML rule-based models to Jess rule models is designed by a M2M transformation which maps each CML metamodel constructor to one or several Jess Rule metamodel elements. The Jess rule model generated by the M2M transformation is the source model for a 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. The M2T transformation is designed using JET, as described later in this paper.

4.3 MDD of JSF Web architecture

A second MDD process is applied (see Figure 3) to generate a Web architecture that integrates rules into a Web application. In this process, Jess rules can be integrated into the Web application, since both the Jess rule base and the Web architecture are generated from the same CML model.

Figure 5 shows the proposed target architecture for rule-based Web applications, based on the MVC architecture pattern, the JSF framework and rich AJAX JBoss Richfaces components.

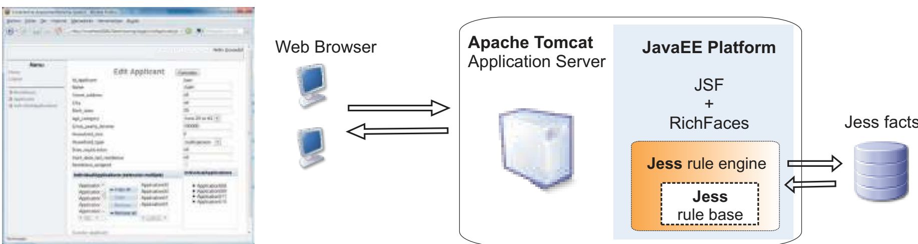


Fig. 5. Rule-based Web application architecture

The integrated rule engine manages the Jess rule base and the text file containing persistent *facts*. The Web application enables the user to perform four basic predetermined functions, create new instances, read the current list of instances, update and delete instances. That 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 with conditions evaluated as true, and executing their actions to infer new information or modify existing one.

A metamodel for the JSF Web architecture was designed. 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 instance creation and edition, a JSF page for listing all instances of that concept type, 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.

As a result, the use of both rules and AJAX technology improves 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 the instance creation or edition, for example, updating choice-field values.

5 Tool Support: *InSCo-Gen*

Our rule-based Web application model-driven development approach is demonstrated by our proof-of-the-concept, the *InSCo-Gen* tool. *InSCo-Gen* was developed using MDD tools provided by the Eclipse Modeling Project. Models and metamodels were defined using the Eclipse Modeling Framework (EMF[4]), including three metamodels, the CML metamodel for conceptual models, the Jess Rule metamodel used for representing Jess platform-specific models, and the JSF metamodel used by Web-based specific models.

Conceptual models conforming to the CML metamodel are created using the built-in reflective EMF editor. In order to improve model specification, the reflective editor is customized using Exeed (EXtended Emf EDitor) [22], a plugin which can modify editor default icons and labels, adding Exeed annotations to the metamodel. A screenshot with a model created with this editor is shown in Figure 2.

Modeling certain aspects of Web design, such as interaction and presentation, is implemented in different ways. Whereas interaction features are added to the conceptual CML metamodel through metaclass properties, presentation is defined by a set of XML configuration files, which can be edited by the developer before generating the Web application code.

Two M2M transformations are designed with Atlas Transformation Language (ATL[5]). 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 (JET[6]). As a result, *InSCo-Gen* automatically produces the Web application code, on one hand, source text files with Jess rules and facts, 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 (Application Programming Interface) to integrate 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 [8], an open source framework that adds AJAX capability to JSF applications.

[4] http://www.eclipse.org/modeling/emf/
[5] http://www.eclipse.org/m2m/atl/
[6] http://www.eclipse.org/modeling/m2t/?project=jet

6 Related Work

Our proposal uses the CML rule and ontology modeling formalism as the MDD source model. To put CML into the MDD framework, we defined a metamodel for CML. The definition of a UML Profile for the specification of CML knowledge models is addressed in [23] where the authors also discusses the possible mapping of the profile elements to a Jess platform specific model.

Some previous work has proposed the generation of Jess rules from ontology and rule models, such as OWL (Ontology Web Language) [24] and SWRL (Semantic Web Rule Language) [25]. These proposals focus on Jess code generation without applying a genuine MDD/MDA approach. But the most important difference between the proposal presented in this paper and those publications is that they do not integrate Jess into a functional Web application, so the Jess rule base generated must be run in a development tool using a shell, such as the Protege JessTab [26].

An MDD approach to Web Applications based on MVC and JavaServer Faces is described in [27]. Existing Web Engineering methods, such as UWE [28], WebML [29] and WebDSL [30], approach the design and development of Web applications addressing such concerns as structure, presentation, navigation. However, they do not consider rule modeling in Web application development. Our proposal focus on introducing rule modeling in this context, and we do not consider other concerns of Web application modeling such as navigation model, since we simplify the functionality to CRUD operations and, therefore, types and navigation links are fixed and preset.

Regarding MDD of Web applications integrating rules, [31] describes MDD principles for rule-based Web services modeling using R2ML, and proposes an MDD approach for generating Web services from rule models. Whereas this proposal focuses on a Web services architecture, our work is based on a MVC architecture using the JSF framework.

7 Conclusions and future work

In this paper, rule-based and model-driven techniques are intertwined for the development of rule-based Web applications. The main contribution of our work is to enrich the specification of Web applications with a rule modeling formalism, introducing a new concern in Model-Driven Web Engineering. A model-driven approach for generating Web implementation enhanced with inference features is described and 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 components to enhance the user interface with AJAX capabilities. The Jess rule engine is embedded in the Web application to provide inference capabilities. Our proposal does not include a navigation model, since application functionality is predetermined by CRUD functions.

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

The approach is being evaluated through its use in the development of 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 grapes.

As future work, it is planned to use other ontology and rule modeling languages such as OWL and SWRL as source models for the model-driven approach, and define interoperability modules with other rule formalisms. Different rule platforms, such as JBoss Rules [32], will be also considered as a target rule technology. The Web application generated, which is aimed at enriching the architecture with database facilities, will be improved to provide a complete persistence layer.

Acknowledgments. This work was supported by the Spanish Ministry of Education and Science under the project TIN2004-05694, and by the Junta de Andalucia (Andalusian Regional Govt.) project P06-TIC-02411.

References

1. Eiter, T., Ianni, G., Krennwallner, T., Polleres, A.: Rules and ontologies for the semantic web. In Baroglio, C., Bonatti, P.A., Maluszynski, J., Marchiori, M., Polleres, A., Schaffert, S., eds.: Reasoning Web. Volume 5224 of Lecture Notes in Computer Science., Springer (2008) 1–53
2. Brachman, R.J., Levesque, H.J.: Knowledge representation and reasoning. Morgan Kaufmann, San Francisco (2004)
3. Object Management Group: MDA Guide Version 1.0.1. OMG document: omg/2003-06-01 (2003)
4. Mellor, S., Clark, A., Futagami, T.: Model-Driven Development - Guest editors introduction. IEEE Software **20**(5) (Sep-Oct 2003) 14–18
5. Anjewierden, A.: CML2. Technical Report 11, University of Amsterdam (1997) URL: http://www.swi.psy.uva.nl/projects/kads22/#cml2.
6. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W.V., Wielinga, B.: Knowledge Engineering and Management: The CommonKADS Methodology. The MIT Press, Cambridge (2000)
7. Sun Microsystems: JavaServer Faces http://java.sun.com/javaee/javaserverfaces/.
8. JBoss: RichFaces (2007) http://www.jboss.org/jbossrichfaces/.
9. Friedman-Hill, E.: Jess in Action: Java Rule-Based Systems. Manning Publications (2003)
10. del Águila, I.M., Cañadas, J., Palma, J., Túnez, S.: Towards a methodology for hybrid systems software development. In: Proceedings of the Int. Conference on Software Engineering and Knowledge Engineering (SEKE). (2006) 188–193
11. Durkin, J.: Expert Systems: Catalog of Applications. Akron (Ohio), Intelligent Computer Systems Inc. (1993)
12. Russell, S.J., Norvig, P.: Artificial intelligence: a modern approach. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1995)
13. Object Management Group: Semantics of Business Vocabulary and Business Rules (SBVR). http://www.omg.org/spec/SBVR/1.0 (2008)
14. Object Management Group: Ontology Definition Metamodel RFP (2003) Available: http://www.omg.org/cgi-bin/doc?ad/2003-03-40.

15. Object Management Group: Production Rule Representation RFP (2003) Available: http://www.omg.org/cgi-bin/doc?br/2003-09-03.
16. RuleML: The Rule Markup Initiative (2001) URL: http://www.ruleml.org.
17. Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language combining OWL and RuleML. W3C. Available at www.w3.org/Submission/2004/SUBM-SWRL-20040521 (2004)
18. REWERSE Working Group I1: R2ML -The REWERSE I1 Rule Markup Language (2006) URL: http://oxygen.informatik.tu-cottbus.de/rewerse-i1/?q=R2ML.
19. Rule Interchange Format Working Group: RIF Use Cases and Requirements. W3C Working Draft (2006) URL: http://www.w3.org/TR/rif-ucr/.
20. Frankel, D., Hayes, P., Kendall, E., McGuinness, D.: The Model Driven Semantic Web. In: 1st International Workshop on the Model-Driven Semantic Web (MDSW2004), Monterey, California, USA. (2004)
21. Chaur G. Wu: Modeling Rule-Based Systems with EMF. Eclipse Corner Articles http://www.eclipse.org/articles/ (2004)
22. Kolovos, D.S.: Exeed: EXtended Emf EDitor - User Manual. http://www.eclipse.org/gmt/epsilon/doc/Exeed.pdf (2007)
23. Abdullah, M., Benest, I., Paige, R., Kimble, C.: Using unified modeling language for conceptual modelling of Knowledge-Based systems. In: Conceptual Modeling - ER 2007. (2007) 438–453
24. Mei, J., Bontas, E.P., Lin, Z.: OWL2Jess: A Transformational Implementation of the OWL Semantics. Lecture Notes in Computer Science **3759** (2005) 599–608
25. OConnor, M., Knublauch, H., Tu, S., Grosof, B., Dean, M., Grosso, W., Musen, M.: Supporting Rule System Interoperability on the Semantic Web with SWRL. Lecture Notes in Computer Science **3759** (2005) 974–986
26. Eriksson, H.: Using JessTab to integrate Protege and Jess. Intelligent Systems, IEEE **18**(2) (2003) 43–50
27. Distante, D., Pedone, P., Rossi, G., Canfora, G.: Model-Driven development of web applications with UWA, MVC and JavaServer faces. In: L. Baresi, P. Fraternali, and G.-J. Houben (Eds.): ICWE 2007, LNCS. Volume 4607., Springer, Heidelberg (2007) 457–472
28. Koch, N., Knapp, A., Zhang, G., Baumeister, H.: Uml_based web engineering: An approach based on standards. In: Web Engineering: Modelling and Implementing Web applications. Human-Computer Interaction Series. Springer, Berlin (dec 2007)
29. Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: Designing Data-Intensive Web Applications. 1 edn. Morgan Kaufmann (December 2002)
30. Groenewegen, D.M., Hemel, Z., Kats, L.C., Visser, E.: WebDSL: a domain-specific language for dynamic web applications. In: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, Nashville, TN, USA, ACM (2008) 779–780
31. Ribarić, M., Gašević, D., Milanović, M., Giurca, A., Lukichev, S., Wagner, G.: Model-Driven engineering of rules for web services. In: Generative and Transformational Techniques in Software Engineering II: International Summer School, GTTSE 2007, Braga, Portugal, July 2-7, 2007. Revised Papers, Springer-Verlag (2008) 377–395
32. JBoss: Drools documentation http://www.jboss.org/drools/documentation.html.

Design Process Ontology – Approach Proposal*

Grzegorz J. Nalepa[1] and Weronika T. Furmańska[1]

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
gjn@agh.edu.pl, wtf@agh.edu.pl

Abstract. The ARD+ method supports the conceptual design of the XTT^2-based rules. However, number of limitations of the method have been identified. In this paper a new approach to the conceptual design of rules is proposed. The main idea comes down to the use of the Semantic Web methods and tools to represent ARD+ and overcome the limitations of the original method. In the approach proposed in this paper an OWL ontology capturing the semantics of the ARD+ model is proposed. Such an ontology models the functional dependencies between rule attributes, as well as the history of the design process. At the same time it is more flexible that the original method, opening up possibility of integration with other modeling methods and tools, e.g. UML.

1 Introduction

Practical design support is important for intelligent systems [1]. The phase of knowledge acquisition and initial conceptual modeling with help of human experts largely influences the quality complex systems. In case of knowledge-based system a hierarchical and iterative feature of this process improves the design.

ARD+ [2,3,4] method has been invented to support the conceptual design of the XTT^2-based rules. ARD+ (*Attribute Relationship Diagrams*) is a rule prototyping method in the HeKatE project (http://hekate.ia.agh.edu.pl). It supports the logical rule design with the XTT^2 method (*eXtended Tabular Trees*) [5,6]. However, number of limitations of the method have been identified.

In this paper a new approach to the conceptual design of rules is proposed. The main idea comes down to the use of the Semantic Web methods and tools to represent ARD+ and overcome the limitations of the original method. In the approach proposed in this paper an OWL ontology capturing the semantics of the ARD+ model is proposed. Such an ontology models the functional dependencies between rule attributes, as well as the history of the design process. At the same time it is more flexible than the original method, opening up possibility of integration with other modeling methods and tools, e.g. UML.

The rest of the paper is organized as follows: The next section describes the context of ARD+ rule prototyping. Then the limitations of the original method

* The paper is carried out within the AGH UST Project No. 10.10.120.105.

are outlined, and the motivation for its extension given. The proposed *Design Process Ontology* (DPO) is a new approach to the conceptual design of rules. The DPO is introduced in the subsequent section. Possible directions for the future work are given in the final section.

2 ARD+ Rule Prototyping Method

The *Attribute Relationship Diagrams* (ARD+) method [2,3,4] supports the conceptual design of rule systems. The primary assumption is, that the state of the intelligent system is described by the attribute values, which correspond to certain system characteristics. The dynamics of the system is described with rules. In order to build the model of the dynamics, the attributes (in this approach state variables) need to be identified first. The identification process is a knowledge engineering procedure, where the designer (knowledge engineer) uses ARD to represent the identified attributes, together with their functional dependencies captured. Using them, rules can be built in the next logical design phase.

ARD is a general method, that tries to capture two features of the design: the attributes, with functional relations between them, and the hierarchical aspect of the process. The second feature is related to the fact that in practice the knowledge engineering process is a gradual refinement of concepts and relations.

In Fig. 1 a simple ARD dependency diagram can be observed. It is in fact one of the phases of the benchmark thermostat case study [7] studied in detail in the HeKatE project. The diagram models a simple dependency read as "thermostat `Temperature` depends on `Time` specification". This is a general statement – currently ARD does not model what the specific dependency is, only a simple fact that some dependency exists.

In the following design stage this model can be refined, by specifying `Time` as a compound attribute, and later on discovering that the set of newly introduced attributes (`Date, Hour, season, and operation`) can be in fact decomposed into two subsets that depend on each other. The nodes of the ARD diagram correspond to so-called *characteristics* (properties) that are described by one or more *attributes*. Attributes can be *conceptual* (general), and *physical* (specific).

Two transformations of the model are possible: finalization and split. The specification transformation (between `Time` and `Date`, `Hour`, `season`, and `operation`) is called *finalization*, whereas the other one is called *split*. These are captured in the *Transformation Process History* diagram (TPH). Together with the ARD dependency diagram they form the *ARD Model*. In the model on the right (Fig. 3) the black edges correspond to finalization and split transformations, and the blue edges show the functional dependencies.

In general, ARD could be used support the design of both forward and backward chaining rules. However, so far it's been mainly used for forward chaining. The basic idea is that having the most detailed, specific ARD dependency diagram, rule prototypes can be automatically built. A rule prototype is a pair of sets of attributes present in the rule premise, and a set of decision attributes. The prototype is aimed at an *attributive* rule language [3], such as XTT^2 [5,6].

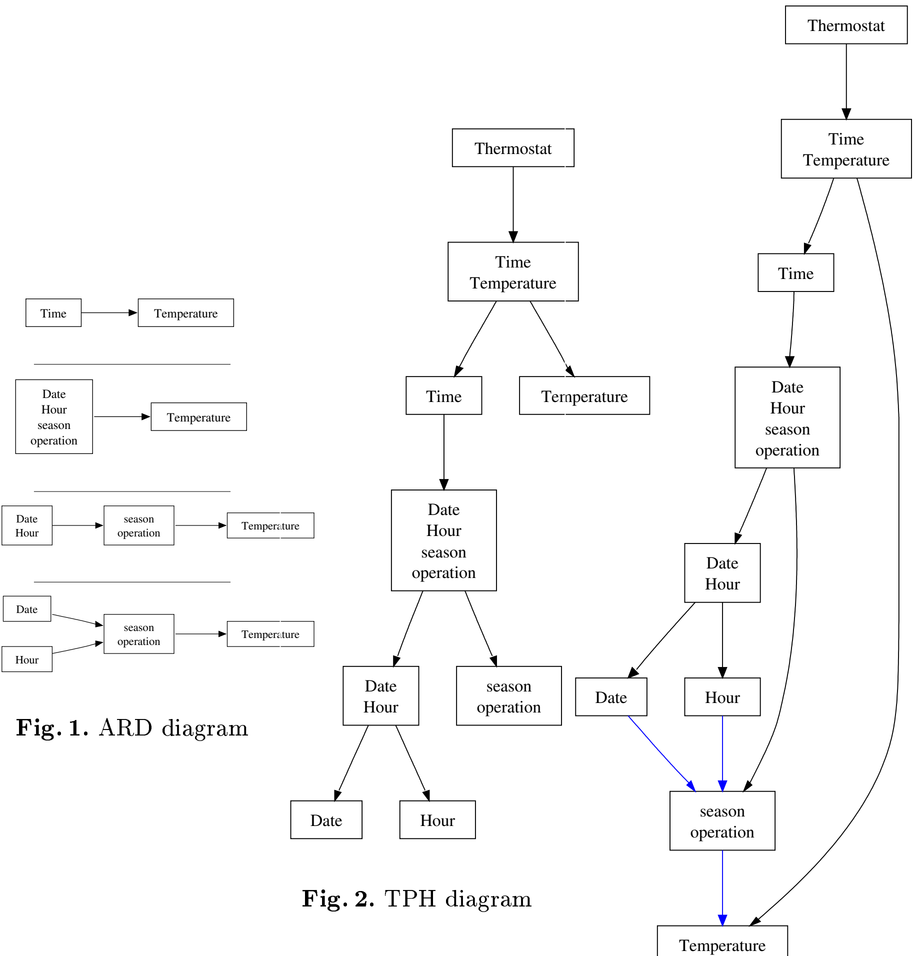


Fig. 1. ARD diagram

Fig. 2. TPH diagram

Fig. 3. ARD+ model

3 Motivation

The ARD+ method [4] is an extension of the original ARD [2,3]. Compared to its predecessor it has a formalized description and a well-defined set of model transformations. ARD+ also introduced the concept of capturing the *evolution of the model design* by the means of the TPH diagram. The method also provided a practical algorithm for building XTT prototypes. The ARD+ design process requires the knowledge engineer to identify attributes, characteristics and relations in both the ARD and TPH diagrams. However, it is apparent that the method has certain important *limitations* or drawbacks.

1. The identification of attributes and dependencies is a straightforward task only in the case of simple, small systems with 10-30 rules. However, it could

turn out a very tedious and time consuming tasks in case of complex systems having tens of attributes and hundreds of rules.
2. ARD+ allows only to capture general functional dependencies, without classifying them in any way. In fact the "ARD dependency" has a very unspecific semantics. In ARD+ it is possible to state "A depends on B and C" but it is not possible to specify *how* it depends, what is the nature of the dependency.
3. The semantics of TPH is also a very broad one. An edge in the TPH diagram simply means that the new ARD+ characteristics is somehow related to another characteristics on the previous stage of the design. Again, it is not explicitly specified *how* it is related. In fact, in ARD+ two transformations are possible: finalization and split. The goal of the TPH is to capture these transformations. However it does not explicitly differentiates them (there is only one class of TPH edges).
4. The last problem concerns a coherent description of the ARD+ model. In [4] two diagrams are described: the ARD+ diagram capturing the functional dependencies between properties grouping attributes at a given design stage, and the TPH diagram, capturing the history of the design process. Later on, in the design tool VARDA [8] a combined diagram – here referred as an *ARD+ model* – has been introduced. It combines the dependency and history diagrams as observed in Fig. 3. However, it has not been formally described and analyzed.

The first problem concerns support for knowledge acquisition in general, and has been addressed in [9]. A practical approach to a partial automation of the attribute and dependency identification process by the use of knowledge discovery methods has been introduced there.

The focus of this paper is to propose a single coherent solution for the three remaining problems. A richer knowledge representation model is proposed. In particular it should:

1. allow to specify different classes of functional dependencies,
2. provide more expressive means for the history description,
3. allow to build a single coherent model combining both functional dependencies and history information in a single model.

In the next section a proposal of using standard Semantic Web methods meeting the above mentioned requirements is put forward.

4 Design Process Ontology Proposal

The basic idea presented here comes down to proposing an *ontology* – called the *Design Process Ontology* (DPO) –capturing the functional dependencies present in the main ARD diagram and history information captured in the TPH.

In general, an ontology is a knowledge representation [10] that serves as a formal definition of objects in the universe of discourse and relationships among them. The domain is described by means of concepts (classes), roles (properties)

and instances (individuals). Relations can be specified both among classes and individuals. Ontologies allow for a formal definition of the vocabulary in the universe of discourse, together with its intended meaning and constraints present in the domain. In this paper the Web Ontology Language (OWL) [11], specifically the OWL DL dialect, based on Description Logics (DL) [12] is used.

Similarities of the ontology-based modelling approach and the ARD method has been investigated in [13,14]. Alternative approaches to a mapping between ontology concepts and system attributes have been considered. One of them consists in representing system attributes and characteristics as concepts in an ontology. Another proposal is to treat attributes as instances of a generic class *Attribute*. Both approaches allow for describing the relations between system attributes using ontology properties. In this paper an ontology based on the former approach is presented.

Design Process Ontology is a proposal of a *task ontology* [15]. Its aim is to capture the system characteristics together with dependencies among them, as well as represent the gradual refinement of the design process. Basically, DPO consists of a general class *Attribute* and four properties: `dependsOn`, `transformedInto`, `splitInto`, and `finalizedInto`.

The property `dependsOn` is very general and may be further specialized. It is used to represent *functional dependencies* among the system characteristics. At this stage we do not formally specify the semantics, which intuitively may be put as "one attribute depends on the other". *Functional* in this context have a different meaning than *functional properties* used in OWL (`owl:FunctionalProperty`), where they denote that the property has an unique value for each instance.

The other three properties (`transformedInto`, `splitInto` and `finalizedInto`) denote the TPH relations – the design process transformations. A hierarchy of the TPH properties may be introduced as follows (DL convention):
$split_into \sqsubseteq transformed_into$, $finalized_into \sqsubseteq transformed_into$.
The domain and the range of all the properties is the general class `Attribute`.

DPO may be specialized by concrete ontologies for specific design tasks. In this case system characteristics (conceptual and physical attributes) subclass the `Attribute` class. All the characteristics and attributes identified in a system are represented as independent classes. The properties may be specialized accordingly, so that they range over concrete system classes rather than the general `Attribute` class. An example of such an ontology for the Thermostat system is depicted in Fig. 4. The ontology has been built in OWL using Protegé.

ARD is a method used in a gradual refinement process. As the process progresses, the functional dependencies change and new TPH relations are added. It is worth emphasizing that the historical TPH relations remain unchanged, whereas the functional ones are different at each process stage (observe Fig. 2). Thus, the ontology is different at various design stages. At a given moment an ontology represents all of the characteristics and attributes identified in the system from the beginning of the design process. All the TPH relations, such as split and finalization are shown. As for the functional dependencies, only the most specific relations identified at certain moment are shown (observe Fig. 4).

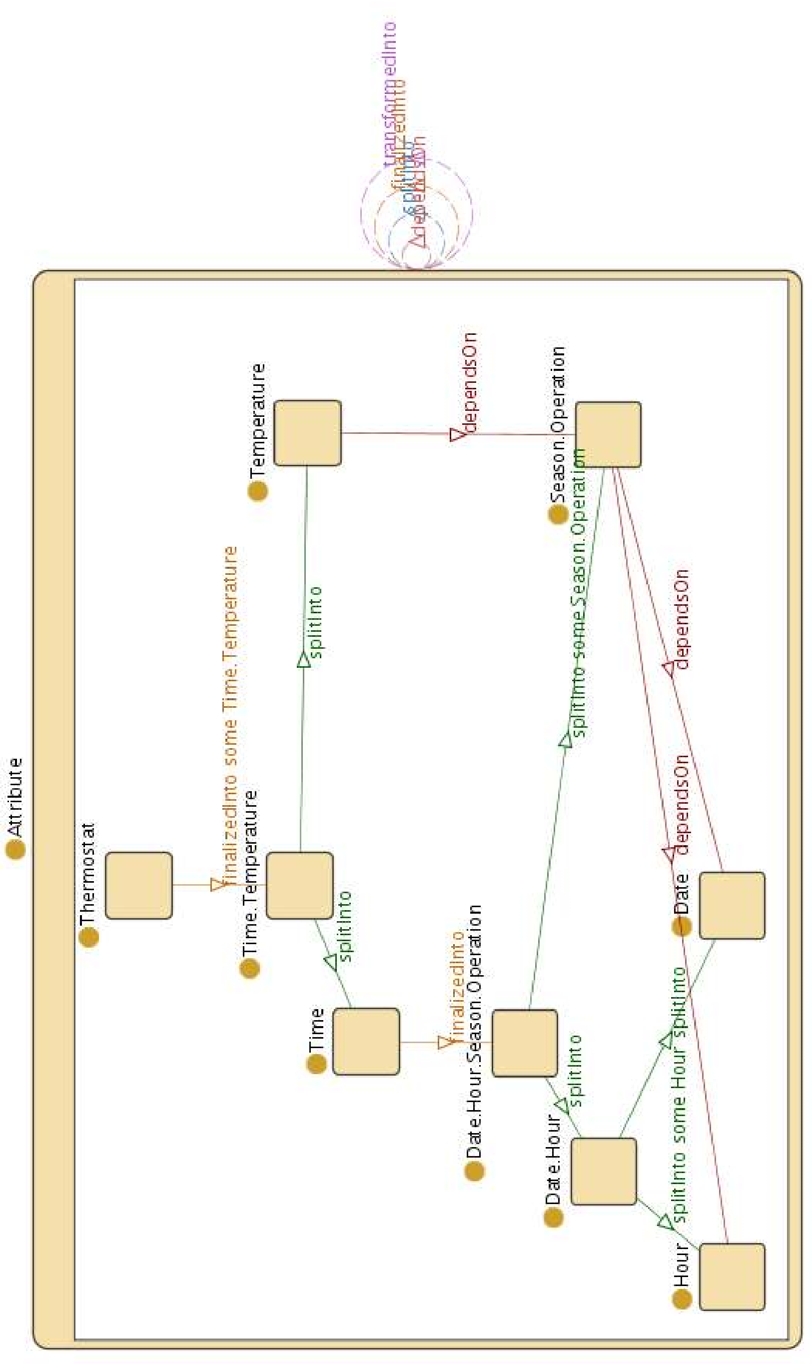


Fig. 4. Simple DPO in OWL designed in Protegé

5 Conclusions and Future Work

The paper concerns the conceptual prototyping of decision rules. The ARD+ method discussed in a paper provides simple means for capturing functional dependencies between attribute present in rules. Moreover, it allows to capture and represent the evolution of the model, the history of the design. However, it has some limitations addressed in the paper.

In order to solve these problems, it is proposed to use the Semantic Web tools in the system conceptual design phase. The proposal is to capture the system elements and various dependencies among them, using an ontology. In the *Design Process Ontology*, certain specific relations are defined. The ontology presented in this paper includes only the basic relations and serves as the illustration of the approach. A concrete ontology specializing the DPO is equivalent to ARD model of a system. Moreover, it provides a more coherent model, while allowing to introduce more relations, which would not be possible in the original ARD.

Future work concerns further investigation of the possibilities of using ontologies in the design process. This includes formalizing various ARD+ model features in Description Logics. As for now, the dependencies between the system characteristics are modelled with various OWL properties (roles). It will be considered, if some of those relations can be incorporated into class descriptions. Certain formal descriptions would help to verify relations such as `split_into`.

The set of various dependencies represented in an ontology will be enlarged. The general functional relation `depends_on` should be specialized, including the differentiation between AND/OR dependencies (as used in AND/OR graphs in diagnostic systems). The set of TPH relations may also be enriched.

As the design process progresses, the Design Process Ontology changes. The transformations between subsequent ontologies will be analyzed and their formalization will be proposed. Use of rules on top of the DPO is considered. These rules could be possibly introduced as DLP (DL Programs) or expressed in SWRL.

A future requirement – not directly addressed here – concerns support for certain annotations present in the UML class diagram. In this case, the reworked method would be closer in semantics to the UML-based design. For more details see [16,17]. The possibilities of integrating OWL with UML using Protege4 is also to be discussed. This approach could provide means to use and integrate both Semantic Web technologies as well as classic software engineering methods to design intelligent systems.

References

1. Giarratano, J., Riley, G.: Expert Systems. Principles and Programming. Fourth edition edn. Thomson Course Technology, Boston, MA, United States (2005) ISBN 0-534-38447-1.
2. Nalepa, G.J., Ligęza, A.: Conceptual modelling and automated implementation of rule-based systems. In: Software engineering : evolution and emerging technologies. Volume 130 of Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam (2005) 330–340

3. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
4. Nalepa, G.J., Wojnicki, I.: Towards formalization of ARD+ conceptual design and refinement method. In Wilson, D.C., Lane, H.C., eds.: FLAIRS-21: Proceedings of the twenty-first international Florida Artificial Intelligence Research Society conference: 15–17 may 2008, Coconut Grove, Florida, USA, Menlo Park, California, AAAI Press (2008) 353–358
5. Nalepa, G.J., Ligęza, A.: A graphical tabular model for rule-based logic programming and verification. Systems Science **31**(2) (2005) 89–95
6. Nalepa, G.J., Ligęza, A.: Hekate methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science (2009) accepted for publication.
7. Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. Addison-Wesley, Harlow, England; London; New York (2002) ISBN 0-201-71159-1.
8. Nalepa, G.J., Wojnicki, I.: Varda rule design and visualization tool-chain. In Dengel, A.R., et al., eds.: KI 2008: Advances in Artificial Intelligence: 31st Annual German Conference on AI, KI 2008: Kaiserslautern, Germany, September 23–26, 2008. Volume 5243 of LNAI., Berlin; Heidelberg, Springer Verlag (2008) 395–396
9. Atzmueller, M., Nalepa, G.J.: A textual subgroup mining approach for rapid ard+ model capture. In Lane, H.C., Guesgen, H.W., eds.: FLAIRS-22: Proceedings of the twenty-second international Florida Artificial Intelligence Research Society conference: 19–21 May 2009, Sanibel Island, Florida, USA. (2009)
10. van Harmelen, F.: Applying rule-based anomalies to kads inference structures. ECAI'96 Workshop on Validation, Verification and Refinement of Knowledge-Based Systems (1996) 41–46
11. McGuinness, D.L., van Harmelen, F.: Owl web ontology language overview, w3c recommendation 10 february 2004. Technical report, W3C (2004)
12. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F., eds.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press (2003)
13. Szostek-Janik, J.: Translations of knowledge representations for rule-based systems. AGH University of Science and Technology (2008) MSc Thesis.
14. Nalepa, G.J., Furmańska, W.T.: Proposal of a new rule-based inference scheme for the semantic web applications. In: 1st International Conference on Computational Collective Intelligence - Semantic Web, Social Networks & Multiagent Systems. (2009) To be published.
15. Guarino, N.: Formal ontology and information systems. In: Proceedings of the First International Conference on Formal Ontologies in Information Systems. (1998) 3–15
16. Nalepa, G.J., Kluza, K.: Uml representation proposal for xtt rule design method. In Nalepa, G.J., Baumeister, J., eds.: 4th Workshop on Knowledge Engineering and Software Engineering (KESE2008) at the 32nd German conference on Artificial Intelligence: September 23, 2008, Kaiserslautern, Germany, Kaiserslautern, Germany (2008) 31–42
17. Nalepa, G.J.: Xtt rules design and implementation with object-oriented methods. In Lane, H.C., Guesgen, H.W., eds.: FLAIRS-22: Proceedings of the twenty-second international Florida Artificial Intelligence Research Society conference: 19–21 May 2009, Sanibel Island, Florida, USA. (2009)

A Data Structure for the Refactoring of Multimodal Knowledge

Jochen Reutelshoefer, Joachim Baumeister, Frank Puppe

Institute for Computer Science, University of Würzburg, Germany
{reutelshoefer, baumeister, puppe}@informatik.uni-wuerzburg.de

Abstract. Knowledge often appears in different shapes and formalisms, thus available as multimodal knowledge. This heterogeneity denotes a challenge for the people involved in today's knowledge engineering tasks. In this paper, we discuss an approach for refactoring of multimodal knowledge on the basis of a generic tree-based data structure. We explain how this data structure is created from documents (i.e., the most general mode of knowledge), and how different refactorings can be performed considering different levels of formality.

1 Introduction

In today's knowledge engineering tasks knowledge at the beginning of a project is often already available in various forms and formalisms distributed over multiple sources, for instance plain text, tables, flow-charts, bullet lists, or rules. We define this intermixture of different shapes of knowledge at different degrees of formalization as *multimodal knowledge*. However, current state-of-the-art tools are often constrained to a specific knowledge representation and acquisition interface for developing the knowledge base. In consequence, the tools are commonly not sufficiently flexible to deal with multimodal knowledge. While the evolution of the knowledge system based on a single formalism (e.g, ontology evolution) has been thoroughly studied, the evolution of multimodal knowledge has not yet been sufficiently considered. In this paper, we propose a data structure and methods, that support representation and refactoring of multimodal knowledge. We have implemented this data structure within a semantic wiki, since such systems proved to be well-suited to support collaborative knowledge engineering at different levels of formalization.

The rest of the paper is organized as follows: In the next section, we give a detailed introduction into multimodal knowledge, and we motivate why refactoring of this type of knowledge is necessary. Further, we provide a data structure, that helps to perform refactorings. In Section 3, we introduce categories of refactorings for multimodal knowledge and we show how they can be performed using the described approach. In this context, we discuss how semi-automated methods can help on the refactoring tasks. In Section 4, we discuss the overlapping aspects of refactoring with the related domains ontology engineering and software engineering. We conclude pointing out the challenges we face with this approach and give an overview of the planned future work.

2 Multimodal Knowledge

We introduce the idea of multimodal knowledge (MMK) and its advantages and challenges. Further, we present a data structure called *KDOM* to represent multimodal knowledge in an appropriate manner. An implementation of this approach within the semantic wiki KnowWE is also given.

2.1 Multimodal Knowledge and the Knowledge Formalization Continuum

Often, knowledge is available in different (digital) representations as we already motivated above. To gain advantage of the knowledge by automated reasoning, the collection of differently shaped knowledge pieces needs to be refactored towards an initial (formalized) knowledge base. During this process the entire knowledge base may contain a wide range of different degrees of formalization and distinct representations. The full range from unstructured plain text over tables or flowcharts to executable rules, or logics sketched as in Figure 1 is metaphorically called the *knowledge formalization continuum* (KFC) [1].

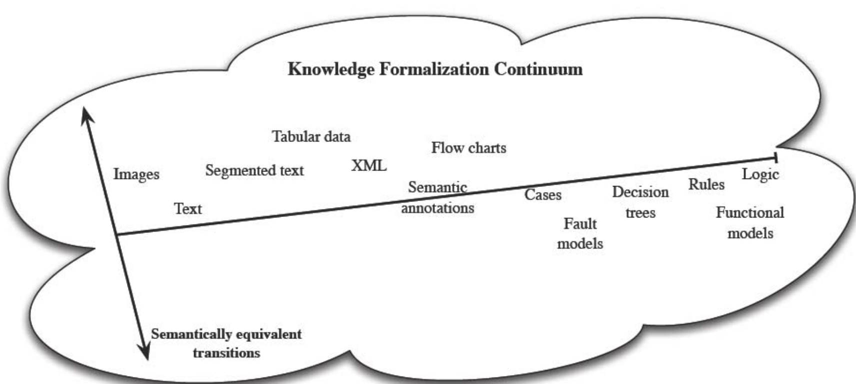


Fig. 1. Sketch of the Knowledge Formalization Continuum building the basis for multimodal knowledge

By turning knowledge into other representations, it allows for (not completely) seamless transitions in either direction - more formal or less formal. The most suitable degrees of formalization for a given project need to be carefully selected according to the cost-benefit principle. However, in any case refactorings towards the desired target knowledge base become necessary. *Refactoring* is defined as changing the structure of something without changing the semantics. For clarification, we comprehend refactoring in this context as changing structure

without changing the *intended* semantics as we are also dealing with non-explicit knowledge artefacts (e.g., plain text). This point of view also considers plain texts as first class knowledge items, which can (manually or semi-automated) be refactored to an executable representation.

Advantages of Working with Multimodal Knowledge:
User friendliness: The formats and representations the domain experts are used to (e.g., plain text in some cases) can be integrated in the knowledge engineering process. Thus, people can participate in the first step with a minimum of training efforts. Lowering the barriers of participation tackles an important problem of knowledge engineering in general.

Bootstrapping: Assuming that we can work with different representations of knowledge, the bootstrapping process shows up being extremely simple: Any documents relevant to the problem domain can just be imported into the system. The evolution of the knowledge driven by the knowledge engineering process will increase its value continuously.

Maintenance: Many (executable) knowledge bases that have been created in the past lack of maintainability. For example, in the compiled versions of large rule bases there is often no sufficient documentation attached to allow other people to further extend or modify the knowledge. Using the MMK approach — to keep the executable knowledge artefacts closely located next to original justifying less formal knowledge entities — provides knowledge engineers a context-sensitive comprehension of the formalized knowledge.

The Challenge of Working with Multimodal Knowledge:
The main challenge is to cope with the different forms of knowledge with respect to formality and syntactical shape. In the next section, we present an approach enabling the multimodal knowledge to be refactored (at the cost of some pre-engineering). However, in detail there are further challenges to be considered:

- handle redundancy of knowledge in different representations and degrees of formality
- tracing the original source of knowledge items (justification) while traversing the KFC
- keeping readability/understandability for humans (the flow of the content)

2.2 KDOM – a Data Structure for Multimodal Knowledge

The most important aspect of this approach is that free text is accepted as a fundamental representation of knowledge. The key idea is to have a self-documenting knowledge base, that can easily be understood by the domain experts. Further, explicitly formalized parts can be embedded into the free text paragraphs. Our approach, to cope with the problems of different knowledge formats described above, implies to break down the given data to (some) textual representation. Some non-textual structure like tables or flowcharts can be converted into text

(for example using cell delimiter signs or XML-formats). However, images, for instance, cannot be converted into a useful (in this context) textual representation. Thus, these items are considered as *knowledge atoms*. This approach treats such items as units, which can be refactored (merely moved) as a whole, but its internal structure cannot be changed. To apply refactoring methods, we build a detailed document tree from the given document corpus. We call this tree the *Knowledge Document Object Model* (KDOM) inspired by the DOM-tree of HTML/XML-documents. The difference is that the source data is not in XML syntax and that we have an explicit underlying semantics for (at least parts of) the content. Of course, one system cannot support every imaginable format of knowledge. Thus, some pre-engineering efforts are necessary to provide support for the formats required. These include the formats given in the startup knowledge and the target formats forming the 'goal' of the engineering task. In the pre-engineering step we define a kind of schema-tree merely forming the ontology of the syntactical structure of the content that is processed.

The KDOM Schema Tree We describe the possible compositions of syntactical entities in a multimodal knowledge document as a tree. At each level the expected occurring formats are listed as siblings. We call such a definition of a syntactical entities together with some intended semantics a *KDOM-type*. An example KDOM schema tree for documents containing text, tables, comments, and rules is shown in Figure 2[1].

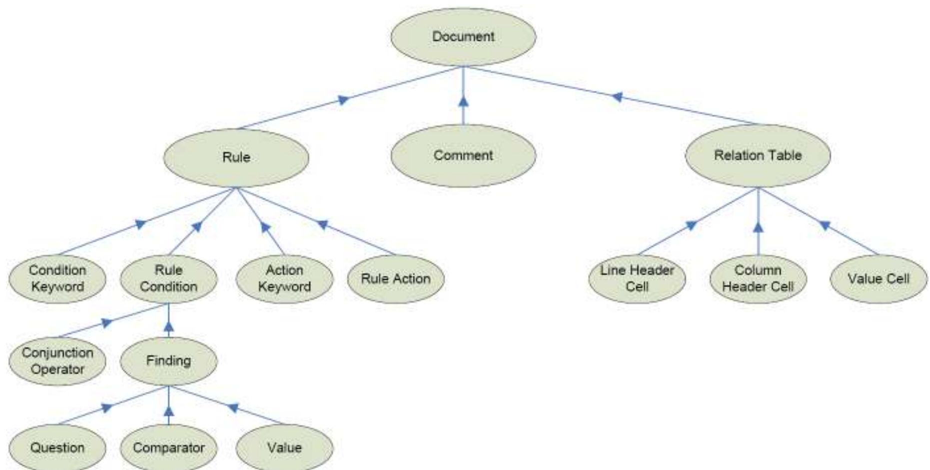


Fig. 2. Sketch of a KDOM schema tree for tables, rules, and comments.

[1] A KDOM schema is similar to an XML-Schema definition except that we do not have XML-Syntax, but an explicit definition of the syntactical shape (a parser) for each type.

This tree schema specifies the syntactical patterns, that can occur as sub-patterns of its parents. The type *Document* — which is always the root node in this KDOM schema — has three children types: *Rule*, *Relation Table*, and *Comment*. Thus, in the document rules, relation tables and comments are expected. Each of these first level types may specify children types for themselves denoting which sub-entities are expected. It does not specify any cardinalities or orders of appearances in the document.[2] In the next paragraph we outline how this KDOM schema is used to create a content tree from knowledge documents applying a *semi-parsing*-like approach. Semi-parsing denotes, that only specific parts of a document are processed in detail by parsers, while other parts remain as text nodes containing a (potentially large) fragment of plain text.

Building a Content KDOM Tree Having an appropriate schema tree of types including their parsers defined, one can start to create content trees from the documents. The following gives a short definition of the tree-based data structure:

Definition 1 (KDOM). *A* KDOM *is defined as set of nodes, where a node* $n \in KDOM$ *is defined as a tuple*

$$n = (id,\ content,\ type,\ parent,\ children).$$

Thus, each node stores a unique id, some textual content, a type (describing the role of the content), one parent (with exception of the root node having no parent), and an (ordered) list of children nodes. A valid KDOM of a document is given if:

1. *The text content of the root node equals the text content of the document.*
2. *The following constraints are true:*
   (a) *text-concatenation(n.children) = n.text for all* $n \in \{KDOM \setminus LEAFS\}$ *LEAFS being the subset of KDOM with an empty chilren set*
   (b) *n.type accepts n.text for all* $n \in \{KDOM\}$*, i.e., the text part of the node n can be mapped to the corresponding type.*

At each level in the schema tree the implicit type *PlainText* is allowed, catching arbitrary content, that is not covered by explicitly defined types (semi-parsing). This definition implies, that a concatenation of the leafs in depth-first search order results in the full document text string. We also provide types for concepts, concept values, conditions over concepts, and rules; further types can be easily added. The construction of a KDOM is sketched by pseudo code in Listing 1.1.

The root node of a document always refers to the full document — this is also the first step of the tree building algorithm. Then, in each level all children types are iterated and searched in the father's content. When one type detects

[2] The order of the siblings defines the order the entities are processed in the parsing algorithm (see Listing 1.1)

a text passage that is relevant (*findOccurences* i.e., matched by its parser), then it allocates this text fragment. Once some text fragment is allocated by a type it will only be processed by the children types of the former type (defined by the KDOM schema tree). If there is lots of unstructured text in MMK we expect that lots of text does not match any type and thus is not allocated by an (explicit) type in the tree (*createPlaintextNodes*).

Listing 1.1. A recursive algorithm to build up a KDOM tree

```
buildKDOMTree (fatherNode) :

  forall (type : childrenTypes(fatherNode))
    childrenNodes = findOccurrences(type, unallocatedTextOfFather)

    forall (childNode : childrenNodes)
      buildKDOMTree(childNode)


  forall (string : unallocatedTexts)
    createPlaintextNodes(string)
```

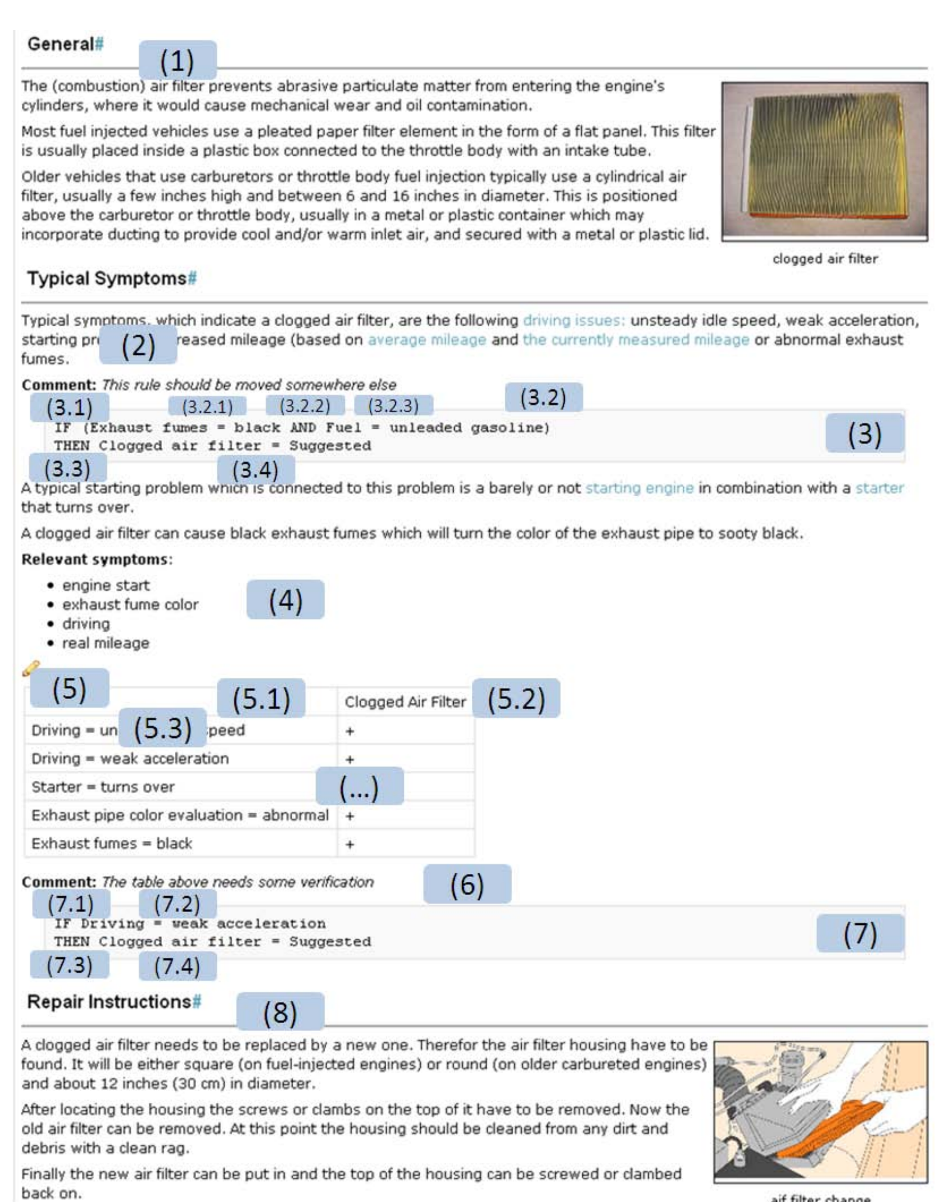
Figure 3 shows an example of a document that is parsed by the KDOM schema introduced in Figure 2. It shows a wiki system describing possible problems with cars. The particular article provides information on clogged air filters in form of plain text paragraphs, rules, and a table.

The first paragraph shows some describing text, followed by a comment line. Then, a rule (labeled number 3) is defined followed by plain text and so on. Rule and tables are labeled in detail hierarchically, e.g., (3.1) and (3.2) for the two parts of the rule. Given that the parser-components of the types of the schema tree are configured correctly to recognize the relevant text parts, we can use the proposed algorithm to create the KDOM content tree from the document. The resulting parse tree shown in Figure 4 has one node for each labeled section from the document.

As required already mentioned above any level in the tree contains the whole content of the document. The content can be considered/engineered at different levels of formality. Thus, also the refactoring methods in general can be applied at different levels.

2.3 Implementation in KnowWE

Semantic wikis have been successfully used in many different software engineering and knowledge engineering projects in the last years, e.g., KIWI@SUN [2]. Further, a semantic wiki is a suitable tool to capture multimodal knowledge as described. For this reason we implemented the introduced KDOM data structure in the semantic wiki KnowWE [3]. In fact, the KDOM tree is the main data structure carrying the data of the wiki pages. A unique ID is assigned to



General#

The (combustion) air filter prevents abrasive particulate matter from entering the engine's cylinders, where it would cause mechanical wear and oil contamination.

Most fuel injected vehicles use a pleated paper filter element in the form of a flat panel. This filter is usually placed inside a plastic box connected to the throttle body with an intake tube.

Older vehicles that use carburetors or throttle body fuel injection typically use a cylindrical air filter, usually a few inches high and between 6 and 16 inches in diameter. This is positioned above the carburetor or throttle body, usually in a metal or plastic container which may incorporate ducting to provide cool and/or warm inlet air, and secured with a metal or plastic lid.

clogged air filter

Typical Symptoms#

Typical symptoms, which indicate a clogged air filter, are the following driving issues: unsteady idle speed, weak acceleration, starting pr... reased mileage (based on average mileage and the currently measured mileage or abnormal exhaust fumes.

Comment: *This rule should be moved somewhere else*

```
IF (Exhaust fumes = black AND Fuel = unleaded gasoline)
THEN Clogged air filter = Suggested
```

A typical starting problem which is connected to this problem is a barely or not starting engine in combination with a starter that turns over.

A clogged air filter can cause black exhaust fumes which will turn the color of the exhaust pipe to sooty black.

Relevant symptoms:

- engine start
- exhaust fume color
- driving
- real mileage

	Clogged Air Filter
Driving = un... peed	+
Driving = weak acceleration	+
Starter = turns over	
Exhaust pipe color evaluation = abnormal	+
Exhaust fumes = black	+

Comment: *The table above needs some verification*

```
IF Driving = weak acceleration
THEN Clogged air filter = Suggested
```

Repair Instructions#

A clogged air filter needs to be replaced by a new one. Therefor the air filter housing have to be found. It will be either square (on fuel-injected engines) or round (on older carbureted engines) and about 12 inches (30 cm) in diameter.

After locating the housing the screws or clambs on the top of it have to be removed. Now the old air filter can be removed. At this point the housing should be cleaned from any dirt and debris with a clean rag.

Finally the new air filter can be put in and the top of the housing can be screwed or clambed back on.

aif filter change

Fig. 3. An example document containing tables, rules, and comments.

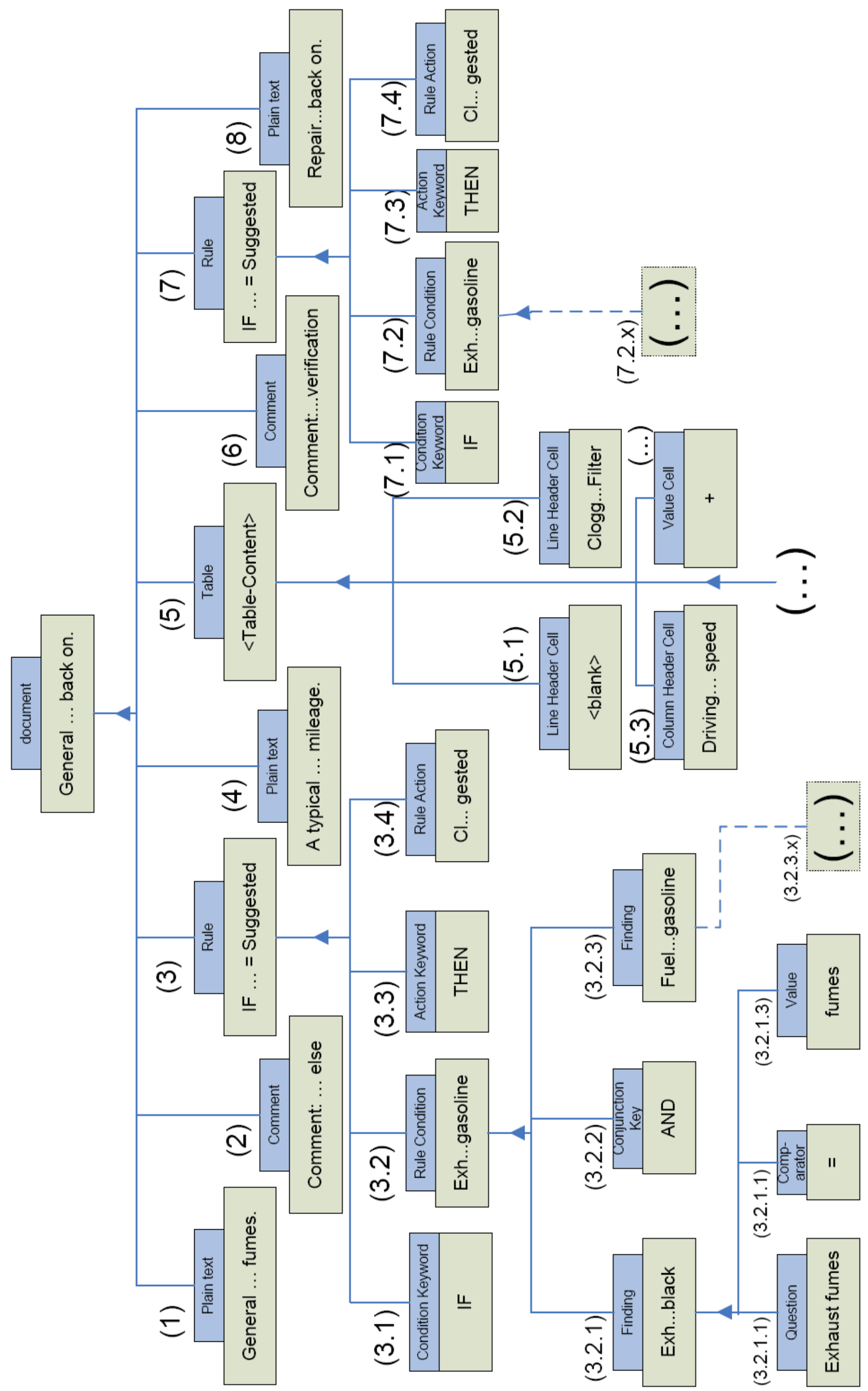


Fig. 4. Structure of the KDOM content tree for the given example document.

every content node of the tree, which allows precise referencing of specific parts of a document/wiki page. The types are integrated into the system by a plugin mechanism. For additional (groups of) types a plugin is added to the core system at system initialization time.

Figure 5 shows a class diagram with the core classes participating in the implementation of the KDOM approach in the system KnowWE.

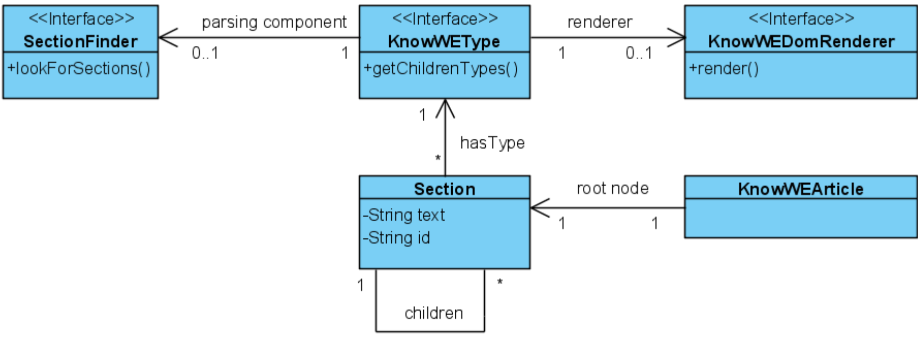


Fig. 5. A simple class diagram of the KDOM implementation in KnowWE.

For each *KnowWEType* a *SectionFinder* as parser component is specified, which is responsible to allocate the correct text areas for the given type. To generate the content tree the algorithm shown in Listing 1.1 is implemented. Of course, a big part of the pre-engineering workload is implementing parsers (SectionFinder) for defined types. For this reason, we provide a library of parser components for common formats (e.g., XML, tables, line/paragraph-based), that can be reused and extended. This allows for quick adaptation to new projects demanding specific knowledge formats. Some of the markups implemented in KnowWE can be found in [3].

3 Evolution of Multimodal Knowledge with Refactorings

The evolution of knowledge in wikis is typically performed by manual editing of the source texts of the wiki pages by the user community. Although, many systems already provide some editing assistance techniques (e.g., templates, auto-completion, tagging), the work of restructuring the knowledge already present is still accomplished by manual string manipulations in the wiki source editor.

Given the KDOM tree described in Section 2.2 the structure of the knowledge can be taken into account to develop further refactoring support. The text-based knowledge can be considered in context when examining the content *and* types of the surrounding nodes (father, siblings, cousins).

3.1 Refactorings

In the following we describe refactorings and how they can be performed with this approach:

1. Renaming of concept
2. Coarsen the value range of a concept

Rename Concept This is probably the operation used most frequently, and it is also simple to perform. The task is to identify each occurrence of the object in all documents and replace it by the new name specified. Precondition of course is, that the occurrences were captured correctly in the KDOM tree generated. Problems can arise when different objects have the same name. For example if different questions have equally named values. Overlapping value terms often occur for example on scaled feature values like *low*, *average/normal*, and *high*. Regarding the following two sketched rules the system needs to distinguish between *normal* as value for *Exhaust pipe color* and for *Mileage evaluation*, when performing a renaming task on the value ranges.

```
IF Exhaust pipe color = normal
THEN Clogged Air Filter = N3

IF Mileage evaluation = normal
THEN Clogged Air Filter = N2
```

As the text string *normal* will probably appear quite frequently in an average document base, it is necessary to identify the occurrences, when it is used as value of a specific concept. The renaming algorithm can solve these ambiguities by looking at the KDOM tree. Figure 6 shows the relevant KDOM subtrees of the two rules. Thus, the renaming algorithm can be configured to check whether a parent node of a value (*1*) is of type *Finding*(*2*). Further, it can look up the content the sibling node of type *Question* (*3*) to infer the context of the value for any occurrence.

Renaming of the occurrences in the formal parts in a consistent way is necessary for compiling executable knowledge. However, the occurrences in less formal parts cannot be identified that easily. But considering the whole knowledge corpus renaming of these occurrences is still desirable with respect to consistency of the multimodal knowledge base. We can provide all occurrences as propositions to the knowledge engineer as a simplest semi-automated approach. To improve this approach, we are planning to employ advanced NLP techniques on less detailed/formalized parts of the KDOM content trees to generate better propositions.

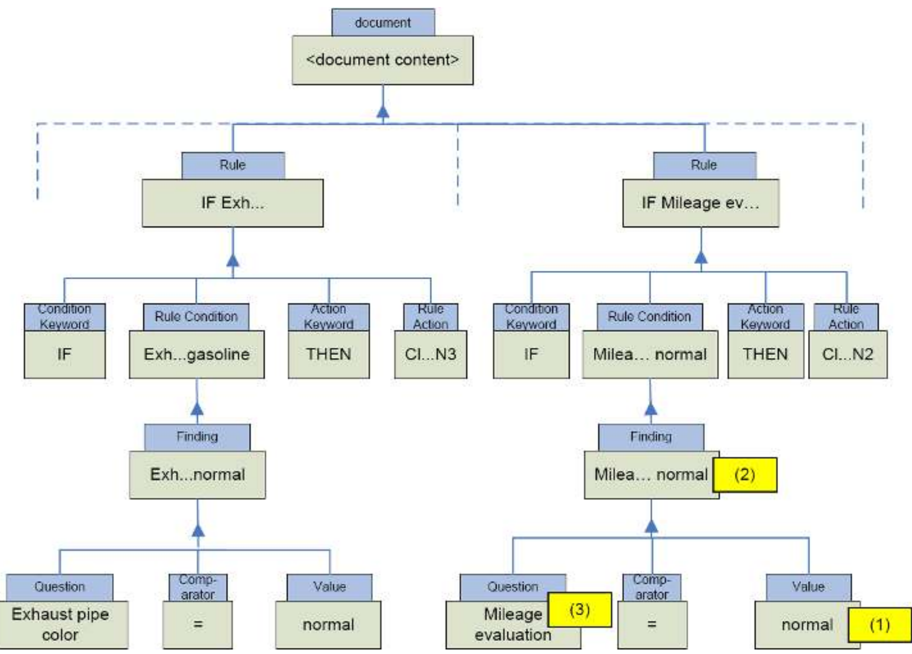


Fig. 6. KDOM subtrees for findings of the two rules listed above.

Coarsen Value Range Often, domain experts start implementing the ontological knowledge with choice questions providing detailed value ranges. During ongoing development the value range of some concepts turn out to be unnecessary precise, e.g., an over-detailed concept. In the car diagnosis scenario, the value range of *Mileage evaluation* is initially defined with the values given in the following:

```
Mileage evaluation
- decreased
- normal
- slightly increased
- strongly increased
```

During the development of the knowledge base it turns, that it is not suitable to have a distinction between *slightly increased* and *strongly increased mileage*. A reason could be, that the knowledge is not so detailed to take advantage of this distinction, resulting in an unnecessary high number of rules or disjunctive expressions. The solution is a mapping of *slightly increased* and *strongly increased* to a new value *increased*. To execute this it is necessary to find and adapt all knowledge items using the object. This operation can be performed as an iterated application of *Rename Concept* on the value range of the concept.

4 Related Work

The presented work is strongly related to refactoring in ontology engineering and techniques for refactoring in software engineering.

Refactoring in Ontology Engineering The benefit of refactoring methods has been recognized in ontology engineering, recently. Current research, however, only discuss the modifications of concepts and properties within an ontology, but does not consider possible implications of tacit knowledge that is neighbouring and supporting the ontology. For example, in [4] various deficiencies were presented that motivate the application of targeted refactoring methods. Here, the particular refactoring methods also considered the appropriate modifications of linked rule bases. In [5] an approach for refactoring ontology concepts is proposed with the aim to improve ontology matching results. The presented refactorings are mainly based on rename operations and slight modifications of the taxonomic structure. In the past, the approach *KA scripts* was presented by Gil and Tallis [6]. KA scripts and refactoring methods are both designed to support the knowledge engineer with (sometimes complex) modifications of the knowledge. More recently, a related script-based approach for enriching ontologies was proposed by Iannone et al. [7].

Refactoring in Software Engineering The presented parsing algorithm can also be compared to the parsing of formal languages (e.g., programming languages), which has been employed successfully for multiple decades. There, the parsers are often generated from a (context-free) grammar specification (e.g., ANTLR [8]) and can process the input in linear time [9]. The parse trees also are used for refactoring in development environments. However, in order to deal with multimodal knowledge one advantage of the KDOM approach is that it can also deal with non-formal languages (to some extend), for example, by employing textmining or information extraction techniques to generate nodes. Additionally, this idea will be extended to a semi-automated workflow involving the knowledge engineer. Further, we can add new syntax as plugins, and we are able to configure the schema at runtime. Even though, we are working on the integration of parse trees generated by classical parsers to allow embedding of formal languages into the semi-parsing process at better performance.

5 Conclusion

In this paper, we introduced the generic data structure KDOM to support the representation of multimodal knowledge. We explained how given documents can be parsed into this data structure with some initial pre-engineering effort. Further, we explained how it serves as the basis for refactoring of the knowledge. We proposed a selection of refactorings and sketched how they can be performed automated or semi-automated using the KDOM. We further plan to apply semi-automated processes on the construction of the KDOM tree by

proposing detected objects or relations to the knowledge engineer, who then can confirm if a given type should be attached to some text fragment.

One of the key challenges in this approach is, that the system needs to be newly configured to each knowledge engineering task, its startup document structures and its target representations. This entails that the knowledge engineering tools needs to be agile and methods and tools for the quick definition of parser components are necessary.

References

1. Baumeister, J., Reutelshoefer, J., Puppe, F.: Engineering on the knowledge formalization continuum. In: SemWiki'09: Proceedings of 4th Semantic Wiki workshop. (2009)
2. Schaffert, S., Eder, J., Grünwald, S., Kurz, T., Radulescu, M.: Kiwi – a platform for semantic social software (demonstration). In: ESWC'09: Proceedings of the 6th European Semantic Web Conference, The Semantic Web: Research and Applications, Heraklion, Greece (June 2009) 888–892
3. Reutelshoefer, J., Lemmerich, F., Haupt, F., Baumeister, J.: An extensible semantic wiki architecture. In: SemWiki'09: Fourth Workshop on Semantic Wikis – The Semantic Wiki Web (CEUR proceedings 464). (2009)
4. Baumeister, J., Seipel, D.: Verification and refactoring of ontologies with rules. In: EKAW'06: Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management, Berlin, Springer (2006) 82–95
5. Svab, O., Svatek, V., Meilicke, C., Stuckenschmidt, H.: Testing the impact of pattern-based ontology refactoring on ontology matching results. In: Third International Workshop On Ontology Matching (OM2008). (October 2008)
6. Gil, Y., Tallis, M.: A script-based approach to modifying knowledge bases. In: AAAI/IAAI'97: Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference, AAAI Press (1997) 377–383
7. Iannone, L., Rector, A., Stevens, R.: Embedding knowledge patterns into OWL. In: ESWC'09: Proceedings of the 6th European Semantic Web Conference, The Semantic Web: Research and Applications. Springer (2009) 218–232
8. Parr, T.: The Definitive ANTLR Reference: Building Domain-Specific Languages. Pragmatic Bookshelf (2007)
9. Wilhelm, R., Maurer, D.: Compiler Design. International Computer Science Series. Addison-Wesley (1995) Second Printing.

Evaluating the Intelligibility of Medical Ontological Terms

Björn Forcher[1], Kinga Schumacher[1], Michael Sintek[1], and Thomas Roth-Berghofer[1,2]

[1] Knowledge Management Department,
German Research Center for Artificial Intelligence (DFKI) GmbH
Trippstadter Straße 122, 67663 Kaiserslautern, Germany

[2] Knowledge-Based Systems Group, Department of Computer Science,
University of Kaiserslautern, P.O. Box 3049, 67653 Kaiserslautern

{firstname.lastname}@dfki.de

Abstract. The research project MEDICO aims at developing an intelligent, robust and scalable semantic search engine for medical documents. The search engine of the MEDICO demonstrator RadSem is based on formal ontologies and is designated for different kinds of users, such as medical doctors, medical IT professionals, patients, and policy makers. Since semantic search results are not always self-explanatory, explanations are necessary to support requirements of different user groups. For this reason, an explanation facility is integrated into RadSem employing the same ontologies for explanation generation. In this work, we present a user experiment that evaluates the intelligibility of labels provided by the used ontologies with respect to different user groups. We discuss the results for refining our current approach for explanation generation in order to provide understandable justifications of semantic search results. Here, we focus on medical experts and laymen, respectively, using semantic networks as form of depiction.[3]

Key words: justification, graphical explanation, semantic search, evaluation, medical terms

1 Introduction

The research project MEDICO aims (among other things) at developing an intelligent semantic search engine for medical documents and addresses different kinds of users, such as medical doctors, medical IT professionals, patients, and policy makers. The ultimate goal of the project [1] is to realize a cross-lingual and modality-independent search for medical documents, such as medical images, clinical findings or reports. Representational constructs of formal ontologies

[3] This research work has been supported in part by the research program THESEUS in the MEDICO project, funded by the German Federal Ministry of Economics and Technology (grant number 01MQ07016). The responsibility for this publication lies with the authors.

are used to annotate and retrieve medical documents. Currently, the MEDICO demonstrator RadSem [2] employs the Foundational Model of Anatomy (FMA) [3] and the International Classification of Diseases, Version 10 (ICD-10)[4]. As there is no existing ontology of the ICD-10 available we implemented a tool which parses the English and German online version providing an OWL ontology of the ICD-10.

Since semantic search results are not always self-explanatory, explanations are helpful to support users who have various intensions to use the search engine. Each user group has different requirements and comes with different a priori knowledge in the medical domain. Medical IT professionals, for instance, may want to test the search engine. In this context, explanations are interesting when the system presents unexpected results. It may turn out that the implementation or the used ontologies are incorrect. Hence, explanations can help to correct the system or to improve it. In contrast to medical IT professionals, patients and citizens are not interested in the exact implementation of the search algorithm. Instead, they may want to learn something about the medical domain. This concerns first of all medical terms but also the connection between medical concepts.

For addressing these issues, we integrated an explanation facility into RadSem. The facility is used to justify search results by revealing a connection between search and annotation concepts. Finding a connection the facility also exploits the mentioned ontologies. Thus, the connection or justification contains further concepts of the FMA or ICD-10. Especially the FMA provides several medical terms for labeling a specific concept. As medical laymen cannot associate any label with corresponding concepts a justification may not be understandable to all of them. In contrast, medical experts may prefer explanations that fit their daily language. In other words, the problem is to select appropriate labels with respect to different user groups. For this reason, we conduct an experiment and discuss its results in order to refine the explanation generation specifically to medical experts and laymen.

This paper is structured as follows. The next section gives a short overview about relevant research on explanations. Section 3 presents current techniques of semantic search algorithms and motivates the need for explanations. Section 4 contains our work of justifying semantic search results. Section 5 describes the user experiment and discusses its results in order to realize a tool that can be used to tailor explanations to different user groups. We conclude the paper with a brief summary and outlook.

2 Related Work

The notion of explanation has several aspects when used in daily life [4]. For instance, explanations are used to describe the causality of events or the semantics of concepts. Explanations help correcting mistakes or serve as justifications.

[4] `http://www.who.int/classifications/apps/icd/icd10online`

Explanations in computer science were introduced in the first generation of Expert Systems (ES). They were recognized as a key feature explaining solutions and reasoning processes, especially in the domain of medical expert systems such as MYCIN [5].

Explanation facilities were an important component supporting the user's needs and decisions [6]. In those early systems, explanations were often nothing more than (badly) paraphrased rules that lacked important aspects or too much information was given at once [7]. For that reason, Swartout and Moore formulated five desiderata for ES explanations [8] which also apply for knowledge-based systems, among them *Fidelity* and *Understandability*.

Fidelity means that the explanation must be an accurate representation of what the ES really does. Hence, explanations have to build on the same knowledge the system uses for its reasoning. Understandability comprises various factors such as *User-Sensitivity* and *Feedback*. User-Sensitivity addresses the user's goals and preferences but also his knowledge with respect to the system and the corresponding domain. Feedback is very important because users do not necessarily understand a given explanation. The system should offer certain kinds of dialog so that users can become clear on parts they do not understand.

In [9], the Reconstructive Explainer is presented producing reconstructive explanations for ES. It transforms a trace, *i. e.*, a line of reasoning, into a plausible explanation story, *i. e.*, a line of explanation. The transformation is an active, complex problem-solving process using additional domain knowledge. The degree of coupling between the trace and the explanation is controlled by a filter which can be set to one of four states regulating the transparency of the filter. The more information of the trace is let through the filter, the more closely the line of explanation follows the line of reasoning. This approach enables a disengagement of an explanation component in order to reuse it in other ES. We took up this theme in our current work.

The Semantic Web community also addresses the issue of explainability. The Inference Web effort [10] realizes an explanation infrastructure for complex Semantic Web applications. Inference Web includes the *Proof Markup Language* for capturing explanation information. It offers constructs to represent where information came from (provenance)or how it was manipulated (justifications). Inference Web includes different tools and services in order to manipulate and present the explanation information. The goal of our research is also to provide tools and algorithms using formal knowledge such as ontologies for explanation provision. The focus of our work is to generate understandable and adequate explanations for knowledge-based systems.

3 Semantic Search

There are diverse definitions of the term *semantic search*. In general, search processes comprise three steps, *i. e.*, query construction, core search process, and visualization of results [11]. In this work, we refer to the most common definition and use the term *semantic search* when formal semantics are used during

any part of the search process [12]. In this context, two main categories of semantic search can be identified: fact and semantic document retrieval. Fact retrieval engines are employed to retrieve facts (triples in the Semantic Web) from knowledge bases based on formal ontologies. Such approaches apply three kinds of core search techniques: *reasoning*, *triple based*, *i. e.*, structural interpretation of the query guided by semantic relations, and *graph traversal search* [12]. Semantic document retrieval engines search for documents which are enriched with semantic information. They use additional knowledge to find relevant documents by augmenting traditional keyword search with semantic techniques. Such engines use various *thesauri* for query expansion and/or apply *graph traversal* algorithms to available ontologies [12, 13]. Analogously, the same semantic techniques are used to retrieve other kinds of resources, *e. g.*, images, videos, where additional formal knowledge is used to describe them.

The MEDICO Demonstrator RadSem uses formal ontologies to annotate medical documents in order to describe their content. The search algorithm exploits the class structure of these ontologies to retrieve documents that are annotated with semantically similar concepts with respect to a certain search concept. For instance, searching for radiographs of the hand, users may obtain documents that are annotated with the concept *index finger* or *pisiform bone*. Currently, RadSem employs the FMA and ICD-10 ontology.

Users have various intensions to use semantic search engines. For instance, a user wants to inform himself of a medical concept he do not remember. In this case, he most probably searches for are similar or superior concept. Imaging, the user searches for information about the *shoulder height* but using the term *shoulder* for his search. If the user obtains a document and associated text snippet highlighting the term *acromion* he may not know whether the document is relevant or not. In this context, a short explanation can provide useful information to support the user's search intention. An explanation expressing that the term acromion is a synonym for shoulder height and that the shoulder height is part of the shoulder may help the user to remember.

The explanation has to reveal the connection between the query and the obtained document. In general, users are not interested in the search techniques of the engine, *i. e.*, how the document is retrieved. In daily tasks users require only a simple justification of the result. As semantic search algorithms use semantic techniques such as ontologies this formal knowledge can be leveraged to generate appropriate explanations.

4 Explanations in RadSem

The explanation facility in RadSem comprises two components: the *Justification Component* and the *Exploration Component*. As its name implies, the first component is primarily intended to justify the retrieval of medical documents. The other component can be used to explore the underlying ontologies and offers various kinds of interaction.

In general, explanations (like any kind of knowledge) have two different aspects: form and content [14]. Explanations are communicated through a certain form of depiction such as text or semantic networks [15]. With respect to the *Understandability* desideratum we chose semantic networks because they are an intelligible alternative to text [16] representing qualitative connections between concepts.

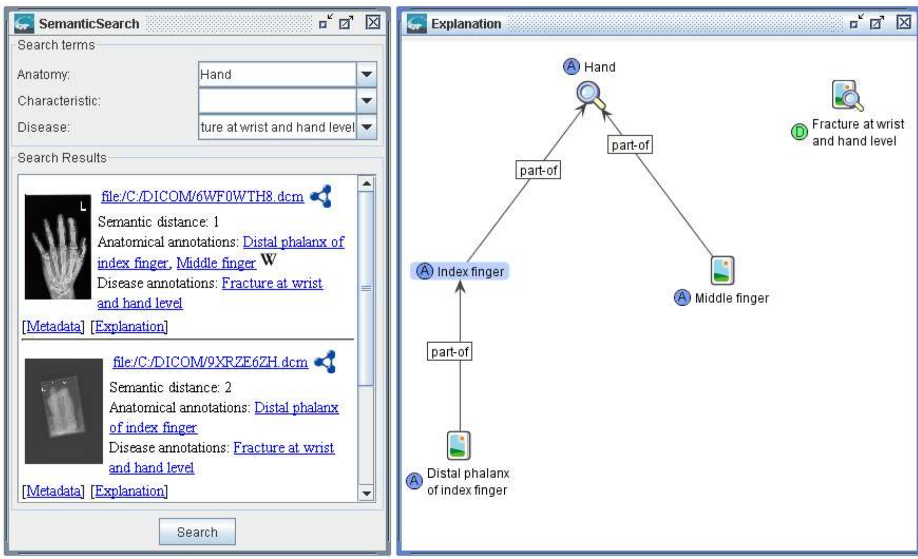


Fig. 1. Justification in RadSem

Most probably, a detailed explanation of the search algorithm used in RadSem is not important for most MEDICO users. Reusing our approach in other semantic search projects we ignore consciously the desideratum *Fidelity*. Hence, the Justification Component performs a kind of reconstructive explanation as described in Section 2 omitting all process information of the search algorithm. In this case the search concepts correspond to the input and the annotation concepts correspond to output in the line of explanation, whereas the story in between is constructed by the explanation facility using the ontologies FMA and ICD-10 as knowledge base.

Since search and annotation concepts belong to ontologies the construction is very simple. In general, ontologies can be transformed into a semantic network representing a mathematical graph. Thus, the construction of the line of explanation for semantic search in MEDICO can be reduced to a shortest path problem. We chose the Dijkstra Algorithm [17] to solve this problem. The algorithm can only be performed on non-negative edge path costs, so the question which costs to choose for properties of the ontologies arises. In our first implementation we

assume an equal distribution, *i. e.*, all properties have the same cost. Figure 1 depicts an example search in RadSem and according justification.

This simple approach already reveals two general problems. The first issue is with the generation itself. The Dijkstra Algorithm determines only one shortest path. Hence, potential alternative explanations are not found which may be better in a certain context with respect to different user groups. In addition, the number of concepts and thus, the amount of information is preset. Potentially, the explanation path contains too much or too few information. The second problem concerns the adequacy of a justification. In particular the FMA provides several synonyms to label a concept. Currently, the explanation facility uses the *preferred label* to name a certain concept in the explanation path. Most probably, not all users can associate the preferred label with a corresponding concept.

Intelligibility is an important aspect of the quality of an explanation and mainly applies for medical layman. In contrast, medical experts may prefer terms which they use in their daily work. For instance, the term *shoulder girdle* may be better for laymen in contrast to *pectoral girdle* which is more appropriate for experts. To conclude, the difficulty is to determine the best label for different user groups such as medical experts and medical laymen. In this work, we focus on the second problem. Our goal is provide a simple approach to evaluate labels with respect to the different user groups. This approach may be extended not only to evaluate single labels but also to evaluate alternative explanation paths or justifications.

Beyond question, the degree of knowledge about medical terms has a significant effect on adequacy and intelligibility. Hence, a method is required to determine the degree of knowledge of different user groups with respect to the terms or labels used in an explanation path. Therefore, an inherent constant must be considered.

An intuitive assumption is that the degree of knowledge can be correlated with the frequency of terms in natural language. A useful statistical measure are frequency classes. According to [18], the frequency class of a term t is defined as follows: Let C be a text corpus and let *f(t)* denote the frequency of a term $t \in C$. The frequency class $c(t)$ of a term $t \in C$ is defined as $\lfloor log_2(f(t^*)/f(t)) \rfloor$, where t^* denotes the most frequently used term in C. In many English corpora, t^* denotes the term *the* that corresponds to frequency class 0. Thus, a more uncommonly used term has a higher frequency class. In the following, we refer to any frequency class $c(t) = i$ as c_i.

5 Experiment

We assume that the degree of knowledge of medical terms can be correlated with frequency classes. The more often a term is used in natural language the more users know about that term. In order to verify the applicability of this assumption we conducted a user experiment. In this experiment the test persons should estimate their knowledge concerning several medical terms.

5.1 Experiment Setup

For evaluating the personal estimation of medical knowledge 200 medical terms of the FMA and ICD-10 were selected consisting of one or two tokens. As German is the mother tongue of the test persons, only German terms were considered in order to avoid a distortion of the evaluation with respect to language problems. We randomly selected ten terms for each frequency class $c_{10}, ..., c_{13}$ and 15 terms for each frequency class $c_{14}, ..., c_{21}$. The frequency classes were determined with the help of a service of the University of Leipzig.[5] The first group of terms contains well known terms such as *Schulter* (shoulder), *Grippe* (influenza), or *Zeigefinger* (index finger), which all test persons typically know. In contrast, the second group contains terms that are typically unknown to medical laymen. In addition, we randomly selected 40 terms of the FMA and ICD-10 where a frequency class could not be determined in order to have a greater probability that at least some terms were unknown to medical experts. We refer to the corresponding frequency class as c_{22}.

Table 1. Personal Knowledge Estimation

(1)	the term is completely unknown;
(2)	the term has been heard of, but cannot be properly integrated into a medical context;
(3)	the meaning of the term is known or it can be derived. In addition, the term can be vaguely integrated into a medical context;
(4)	the meaning of the term is known and it can be associated with further medical terms;
(5)	the term is completely clear and comprehensive knowledge can be associated.

The 200 medical terms were randomly subdivided into four tests each containing a varying number of frequency classes. Every test person was allowed to do only one test. Thus, we had to take care that each of the four tests was done as often as any other one. All test persons had to estimate their knowledge about each term of a test on a scale from 1 to 5 (see Table 1) indicating their *Personal Knowledge Estimation* (PKE).

5.2 Evaluation

In total, thirty-six persons participated in the experiment: twenty-eight laymen and eight medical experts. The two groups were differentiated as follows. Test persons with a profound medical qualification were classified as experts. For instance, this concerns medical staff, students and doctors. All other test persons were classified as laymen. Figures 2 and 3 depict the result of the evaluation.

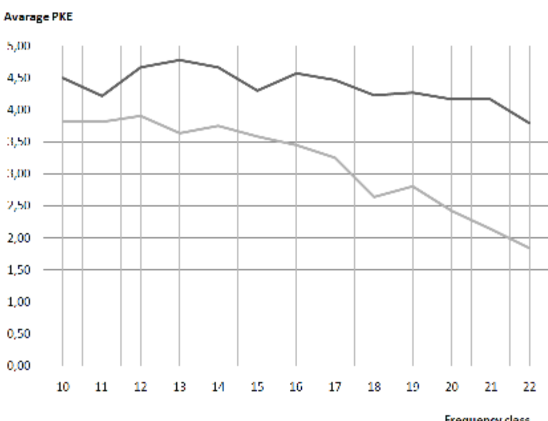


Fig. 2. Experiment results: average values for experts (black) and laymen (gray)

Figure 2 depicts an average value of the PKE as function of the frequency classes for experts (upper curve) and laymen (lower curve) as well. Figure 3 depicts the corresponding standard deviation.

Figure 2 contains two outliers for medical laymen: c_{13} and c_{19}. The first one can be traced to the term *Atlas*. It is an ambiguous term whose meaning in a geological context is quite common. In contrast, its meaning as first cervical vertebra is relatively unknown. The second outlier can be traced to some compounds which are quite common for the German language. The meaning of those terms can easily be derived but their occurrence in daily language is rare. In contrast to laymen, the curve of medical experts is without any irregularity. Merely the estimation of general terms is very interesting because experts seem to consider what they do not know with respect to the general term. The standard deviation for both groups is quite interesting. From frequency class c_{18} on the values jump up. A possible reason for that may be that people have more knowledge in subfields of the medical domain than in others, *i. e.*, when they have a certain disease.

The main objective of the experiment was not to verify a correlation between users' degree of knowledge and frequency classes. In fact, the intention is primarily to denote intervals of frequency classes as a means of prognosis whether user groups probably know a term or require supporting information. For this purpose, we introduce three Boolean functions: $k(t)$ for known terms, $s(t)$ for support requiring terms, and $u(t)$ for unknown terms. With respect to average PKE of medical laymen, we identified three suitable intervals, and defined the functions as follows (index l indicates laymen):

1. $k_l(t)$ is true iff $c(t) \in [c_{11}, ..., c_{15}]$
2. $s_l(t)$ is true iff $c(t) \in [c_{16}, ..., c_{19}]$

[5] `http://wortschatz.uni-leipzig.de/`

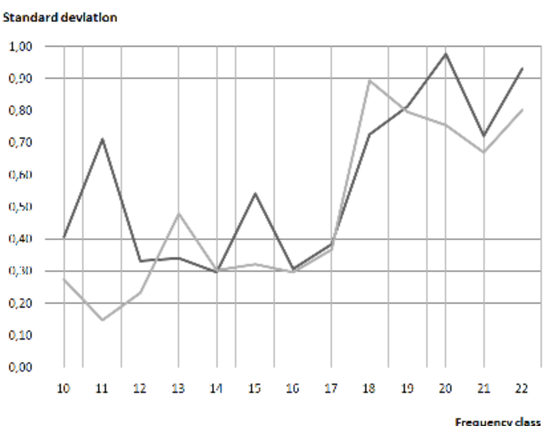


Fig. 3. Experiment results: standard deviation for experts (black) and laymen (gray)

3. $u_l(t)$ is true iff $c(t) \in [c_{20}, ..., c_n]$ and $n > 20$

The proposed functions do not apply to medical experts. The average PKE of all concepts indicates that medical experts generally know terms used in the FMA and ICD-10. Thus, only the function $k_e(t)$ can be defined which is always true (index e indicates experts).

As mentioned before, the functions allow evaluating a justification as presented in Section 4. For instance, there are two justifications A and B of the same search result whereas both comprise three terms. If the middle term of A is a *known term* and if the middle term of B is a *support requiring term*, probably justification A is the better one. The concepts can also be used to tailor justifications. Let a justification represent a path in class hierarchy and comprise four terms. If one of the mid terms is an *unknown term* and the one is *known*, the unknown term can be removed.

In many cases, labels of the FMA or ICD-10 contain other concept labels. For instance, *distal phalanx of left index finger* includes the concept labels *distal phalanx*, *left* and *index finger*. All labels have different frequency classes and thus, a prediction whether a user knows such a concept cannot be made (this applies for all non lexical labels). But this may not be necessary in order to select the most suitable label for a concept with respect to medical laymen or experts. Using the $k_l(t)$ and $k_e(t)$ it is possible to define two sets of labels. These sets can be generalized with respect to various attributes of the labels such as average frequency class of sub labels, label length or token count. The most prominent member of one class can be used to solve a label selection problem. A label with minimal distance to that member may be the most appropriate label for a concept concerning different user groups.

The presented experiment and proposed method may only be seen as a first approach to improve the current explanation generation. We ignored some important aspects of the experiment, such as compounds or ambiguous terms. In

addition, users probably may not estimate their knowledge hundred per cent correctly. For this reason, the presented approach can only be regarded as initial point to evaluate terms or complete explanation paths which can be improved by using further methods such as user interactions.

6 Summary and Outlook

In this paper we presented the explanation facility of the MEDICO Demonstrator RadSem. The semantic search engine of RadSem uses formal ontologies to annotate and retrieve medical documents. The explanation facility employs the same ontologies and uses reconstructive explanations as a means of justifying semantic search results. Improving the justifications we conduct an experiment with medical experts and laymen. The objective of the experiment was to determine a correlation between users' degree of knowledge and medical terms. We discussed the results and proposed a method which can be used to determine which terms of the used ontologies should be used in an explanation with respect to medical experts and laymen as initial start. The overall approach can be used to justify various semantic search algorithms using formal ontologies.

The next step of our research is to refine the method for tailoring and evaluating terms and explanations. In addition, we will consider various kinds of user interactions to improve this method.

References

1. Möller, M., Sintek, M.: A scalable architecture for cross-modal semantic annotation and retrieval. In Dengel, A.R., Berns, K., Breuel, T.M., eds.: KI 2008: Advances in Artificial Intelligence, Springer (2008)
2. Möller, M., Regel, S., Sintek, M.: RadSem: Semantic annotation and retrieval for medical images. In: Proc. of The 6th Annual European Semantic Web Conference (ESWC2009). (2009)
3. Rosse, C., Mejino, J.L.V.: The Foundational Model of Anatomy Ontology. In: Anatomy Ontologies for Bioinformatics: Principles and Practice. Volume 6. Springer (2007) 59–117
4. Passmore, J.: Explanation in Everyday Life, in Science, and in History. In: History and Theory, Vol. 2, No. 2. Blackwell Publishing for Wesleyan University (1962) 105–123
5. Buchanan, B.G., Shortliffe, E.H., eds.: Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley Publishing Company, Reading, Massachusetts (1984)
6. Swartout, W.R., Paris, C., Moore, J.D.: Explanations in knowledge systems: Design for explainable expert systems. IEEE Expert **6**(3) (1991) 58–64
7. Richards, D.: Knowledge-based system explanation: The ripple-down rules alternative. In: Knowledge and Information Systems. Volume 5. (2003) 2–25
8. Swartout, W.R., Moore, J.D.: Explanation in second generation expert systems. Second generation expert systems (1993) 543–585
9. Wick, M.R., Thompson, W.B.: Reconstructive expert system explanation. Artif. Intell. **54**(1-2) (1992) 33–70

10. McGuinness, D.L., Ding, L., Glass, A., Chang, C., Zeng, H., Furtado, V.: Explanation interfaces for the semantic web: Issues and models. In: Proceedings of the 3rd International Semantic Web User Interaction Workshop (SWUI'06). (2006)
11. Organizers, T.: Guidelines for the trecvid 2007 evaluation (2007)
12. Hildebrand, M., Ossenbruggen, J., van Hardman, L.: An analysis of search-based user interaction on the semantic web. Report, CWI, Amsterdam, Holland (2007)
13. Mäkelä, E.: Survey of semantic search research. In: Proceedings of the Seminar on Knowledge Management on the Semantic Web. (2005)
14. Kemp, E.A.: Communicating with a knowledge-based system. In Brezillon, P., ed.: Improving the Use of Knowledge-Based Systems with Explanation. (1992)
15. Ballstaedt, S.P.: Wissensvermittlung. Beltz Psychologische Verlags Union (1997)
16. Wright, P., Reid, F.: Written information: Some alternatives to prose for expressing the outcomes of complex contingencies. Journal of Applied Psychology **57 (2)** (1973) 160–166
17. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1** (1959) 269–271
18. zu Eissen, S.M., Stein, B.: Intrinsic plagiarism detection. In: ECIR. (2006) 565–569

HaDEs – Presentation of the HeKatE Design Environment*

Krzysztof Kaczor and Grzegorz J. Nalepa

Institute of Automatics,
AGH University of Science and Technology,
Al. Mickiewicza 30, 30-059 Kraków, Poland
kk@agh.edu.pl, gjn@agh.edu.pl

Abstract. TOOL PRESENTATION: The paper introduces the HeKatE design environment called HaDEs. The HeKatE project aims at delivering new knowledge representation methods for rule-based systems. Principal ideas include an integrated hierarchical design process covering stages from conceptual, through logical to physical design. These stages are supported by specific knowledge representation methods: ARD+, XTT^2, and HMR. The whole design process is supported by a number of tools, namely: VARDA and HJEd in the ARD+conceptual design stage and rule prototyping, HQEd for the XTT^2 logical design and finally HeaRT, the rule runtime environment. The goal of this tool presentation is to introduce the design process using a practical example.

1 Introduction

Practical design methodologies for intelligent systems remain a field of active development. Developing such a methodology requires an integration of accurate knowledge representation and processing methods [1], as well as practical tools supporting them. Some of the important features of such approaches are: scalable visual design, automatic code generation, support for existing programming frameworks. At the same time quality issues, as well as a formalized description of the designed systems should be considered.

The *HeKatE* project (see hekate.ia.agh.edu.pl) aims at providing an integrated methodology for the design, implementation, and analysis of rule-based systems [2,3]. An important goal of the project is to allow for an easy integration of knowledge and software engineering methods and approaches, thus providing a *Hybrid Knowledge Engineering* methodology. The project delivers several new knowledge representation methods, as well as a set of practical tools supporting the whole design process.

This paper provides a short overview of the project including its main objectives and tools in Sect. 2. Then in Sect. 3 the HeKatE design toolchain called HaDEs is introduced. The paper accompanies a tool presentation given at the KESE 2009 workshop.

* The paper is supported by the HeKatE Project funded from 2007–2009 resources for science as a research project.

2 HeKatE Project Overview

2.1 Research Objectives

The main principles of the HeKatE project are based on a critical analysis of the state-of-the art of the rule-based systems design, see [4].

Formal Language for Knowledge Representation. It should have a precise definition of syntax, properties and inference rules. This is crucial for determining its expressive power, and solving formal analysis issues.

Internal Knowledge Base Structure. Rules working within a specific context, are grouped together and form the extended decision tables. These tables are linked together forming a partially ordered graph structure which encodes the flow of inference.

Systematic Hierarchical Design Procedure. A complete, well-founded design process that covers all of the main phases of the system lifecycle, from the initial conceptual design, through the logical formulation, all the way to the physical implementation is proposed. A constant verification of the model w.r.t. critical formal properties, such as determinism and completeness is provided.

In the HeKatE approach the control logic is expressed using forward-chaining decision rules. They form an intelligent rule-based controller or simply a business logic core. The controller logic is decomposed into multiple modules represented by attributive decision tables. The emphasis of the methodology is its possible application to a wide range of intelligent controllers. In this context two main areas have been identified in the project: control systems, in the field of intelligent control, and business rules [5] and business intelligence systems, in the field of software engineering. In the case of the first area the meaning of the term "controller" is straightforward. In the second area the term denotes a well isolated software component implementing the application logic, or logical model.

2.2 Main Methods

HeKatE introduces a formalized language for rule representation [4]. Instead of simple propositional formulas, the language uses expressions in the so-called *attributive logic* [3]. This calculus has a stronger expressiveness than the propositional logic, while providing tractable inference procedures for extended decision tables. The current version of the rule language is called XTT^2 [6]. The current version of the logic, adopted for the XTT^2 language, is called ALSV(FD) (*Attributive Logic with Set Values over Finite Domains*).

Based on the logic, a rule language called XTT is provided [7,6]. XTT stands for *eXtended Tabular Trees*. The language is focused not only on providing an extended syntax for single rules, but also allows for an explicit structurization of the rule base. XTT introduces explicit inference control solutions, allowing for a fine grained and more optimized rule inference than in the classic Rete-like solutions. The representation has a compact and transparent visual representation suitable for visual editors.

HeKatE also provides a complete hierarchical design process for the creation of the XTT-based rules. The main phase of the XTT rule design is called the *logical design*. The logical rule design process may be supported by a preceding *conceptual design* phase. In this phase the rule prototypes are built with the use of the so-called *Attribute Relationship Diagrams*. The ARD method has been introduced in [8], and later refined in [3]. The principal idea is to build a graph, modelling functional dependencies between attributes on which the XTT rules are built. The version used in HeKatE is called ARD+ as discussed in [9]. The practical implementation on the XTT rule base is performed in the physical design phase. In this stage the visual XTT model is transformed into an algebraic presentation syntax called HMR. A custom inference engine, HeaRT runs the XTT model described in HMR.

The complete framework including the discussed methods and tools is depicted in Fig. 1.

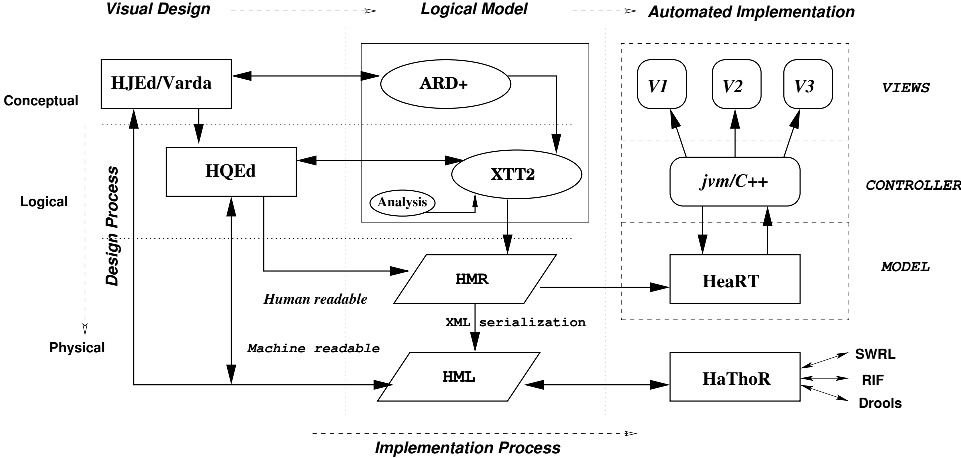


Fig. 1. The complete design and runtime framework

3 HaDEs Design Toolchain

The HeKatE design process is supported by a number of tools supporting the visual design and the automated implementation of rule-based systems (see `https://ai.ia.agh.edu.pl/wiki/hekate:hades`).

HJEd visual editor supports the ARD+ design process. It is a cross-platform tool implemented in Java. Its main features include the ARD+ diagram creation with on-line design history available through the TPH diagram. An example of a design capturing functional dependencies between system attributes is shown in Fig. 2. It is a medical diagnosis system. The diagram on the left shows the dependencies between rule attributes, whereas the right one captures the design

history. Once created, the ARD+ model can be saved in a XML-based HML (HeKatE Markup Language) file. The file can be then imported by the HQEd design tools supporting the logical design.

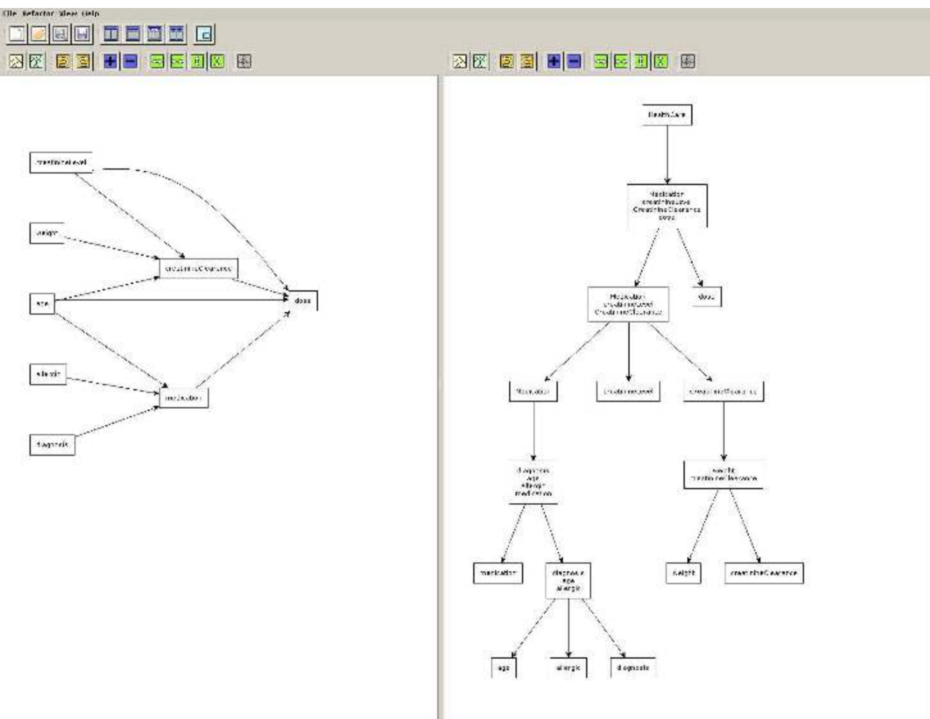


Fig. 2. ARD+ design in HJEd

VARDA is a prototype semi-visual editor for the ARD+ diagrams implemented in Prolog, with an on-line model visualization with Graphviz. The tool also supports prototyping of the XTT model, where table headers including a default inference structure are created, see Fig. 3. In this case three tables are generated. The ARD+ design is described in Prolog, and the resulting model can be stored in HML.

HQEd provides support for the logical design with XTT, see Fig. 4. In the figure some additional decision tables to input attribute values are present. It is able to import a HML file with the ARD+ model and generate the XTT prototype. It is also possible to import the prototype generated by VARDA. HQEd allows to edit the XTT structure with on-line support for syntax checking on the table level. Attribute values entered are checked against their domains and a number of possible anomalies is eliminated.

The editor is integrated with a custom inference engine for XTT^2 called HeaRT. The role of the engine is twofold: run the rule logic designed with the

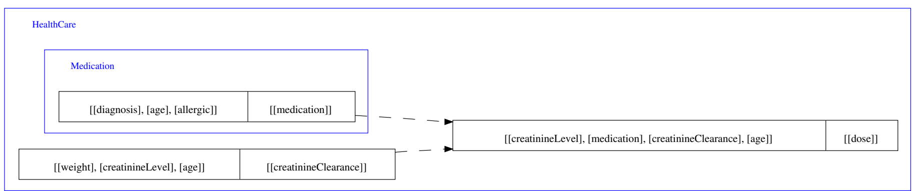


Fig. 3. XTT model generation in VARDA

use of the editor, as well as provide constant formal analysis of the rulebase. The communication uses a custom TCP-based protocol.

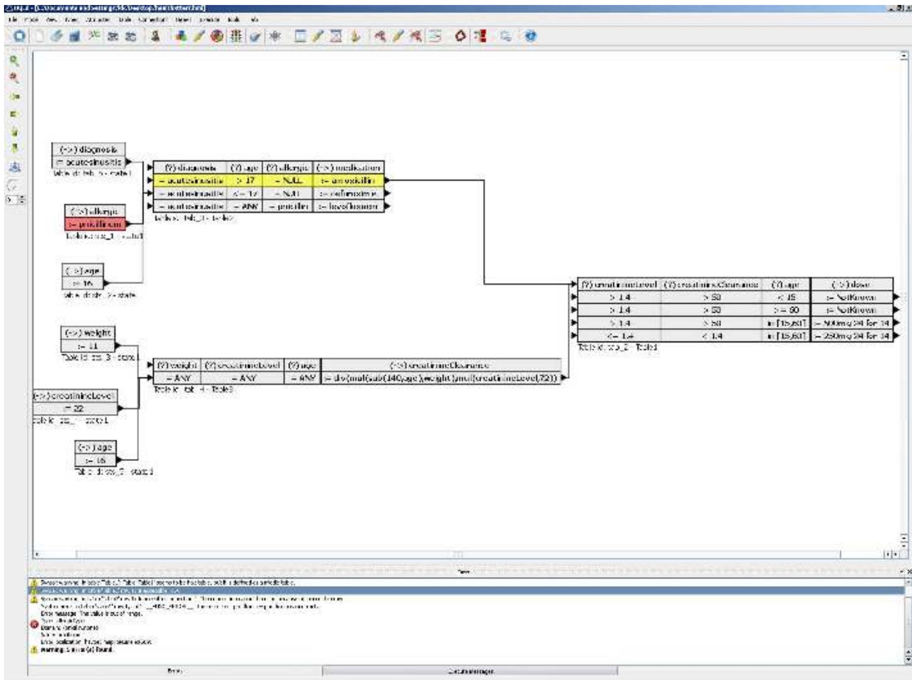


Fig. 4. XTT model edited in HQEd with anomalies detected

HeaRT (HeKatE Run Time) is a dedicated inference engine for the XTT^2 rule bases. It is implemented in Prolog in order to directly interpret the HMR representation which is generated by HQEd. HMR (HeKatE Meta Representation) is a textual representation of the XTT^2 logic designed by HQEd. It is a human readable form, as opposed to the machine readable HML format. The HeaRT engine implements the inference based on the ALSV(FD) logic [6,4].

HalVA (HeKatE Verification and Analysis) is a modularized verification framework provided by HeaRT. So far several plugins are available, including completeness, determinism and redundancy checks. The plugins can be run form the interpreter or from HQEd using the communication protocol.

4 Conclusions

The paper shortly introduces the main concepts of the HeKatE project, its methods and tools. The main motivation behind the project is to speed up and simplify the rule-based systems design process, while assuring the formal quality of the model. The HeKatE design process is supported by the HeKatE design environment called HaDEs. During the presentation given at the KESE workshop the tools were presented using practical examples.

References

1. van Harmelen, F., Lifschitz, V., Porter, B., eds.: Handbook of Knowledge Representation. Elsevier Science (2007)
2. Giarratano, J., Riley, G.: Expert Systems. Principles and Programming. Fourth edition edn. Thomson Course Technology, Boston, MA, United States (2005) ISBN 0-534-38447-1.
3. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
4. Nalepa, G.J., Ligęza, A.: Hekate methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science (2009) accepted for publication.
5. Ross, R.G.: Principles of the Business Rule Approach. 1 edn. Addison-Wesley Professional (2003)
6. Nalepa, G.J., Ligęza, A.: Xtt+ rule design using the alsv(fd). In Giurca, A., Analyti, A., Wagner, G., eds.: ECAI 2008: 18th European Conference on Artificial Intelligence: 2nd East European Workshop on Rule-based applications, RuleApps2008: Patras, 22 July 2008, Patras, University of Patras (2008) 11–15
7. Nalepa, G.J., Ligęza, A.: A graphical tabular model for rule-based logic programming and verification. Systems Science **31**(2) (2005) 89–95
8. Nalepa, G.J., Ligęza, A.: Conceptual modelling and automated implementation of rule-based systems. In: Software engineering : evolution and emerging technologies. Volume 130 of Frontiers in Artificial Intelligence and Applications. IOS Press, Amsterdam (2005) 330–340
9. Nalepa, G.J., Wojnicki, I.: Towards formalization of ARD+ conceptual design and refinement method. In Wilson, D.C., Lane, H.C., eds.: FLAIRS-21: Proceedings of the twenty-first international Florida Artificial Intelligence Research Society conference: 15–17 may 2008, Coconut Grove, Florida, USA, Menlo Park, California, AAAI Press (2008) 353–358

JSON Rules - The JavaScript Rule Engine

Emilian Pascalau[1] and Adrian Giurca[2]

[1]Hasso Plattner Institute, Germany,
emilian.pascalau@hpi.uni-potsdam.de
[2]Brandenburg University of Technology, Germany,
giurca@tu-cottbus.de

Abstract. TOOL PRESENTATION: There is a considerable browser potential in being able to easily wire together different services into new functionality. Usually, developers use JavaScript or related technologies to do browser programming. This short paper presents, JSON Rules, a JavaScript rule engine running Event-Condition-Action rules triggered by Document-Object-Model Events.

1 Introduction

The Rule Engine implementing the JSON Rules [1] language was designed to fulfill at least the following requirements:

- create and execute rules in a Web browser
- support for ECA and PR rules
- forward chaining rule engine, influenced by the RETE algorithm;
- process atomic event-facts;
- the Working Memory contains beside regular facts, event facts.

The main goal of the rule engine is to empower users with the client side abilities to model/execute web scenarios/applications/mashups by means of business rules (See [1] and [2]). Particularly intelligent UI scenarios are in the main stream of interest.

For a better understanding of the context we consider the following situation: *We are looking for a job using the Monster Job Search Service. Once the job is obtained the location is shown on Google Maps.*

2 The JSON Rules language

The language was initially introduced in [1]. JSON notation combined with JavaScript function calls offers large capabilities to express various kinds of rules. Recall that we deal both with production rules and with Event-Condition-Action (ECA) rules i.e. rules of the form

```
RuleID: ON EventExpression IF C1 && ... && Cn DO [A1, ..., Am]
```

where the event part is optional and denotes an event expression matching the triggering events of the rule; `C1, ... Cn` are boolean conditions using a Drools like syntax and `[A1, ... Am]` is a sequence of actions.

2.1 Ontology of events - DOM events

The JSON event expression is related to the Event interface specification in DOM Level 3 Events[1], therefore the properties of this expression have the same meaning as in the Event specification. At runtime these properties of this expression are matched against the incoming DOM events and their values can be processed in the rule conditions and actions.

Example 1 (ECA Rule).

```
{"id": "rule101",
  "appliesTo": ["http://mail.yahoo.com/"],
  "eventExpression": {"type": "click",
                      "target": "$X"
                     },
  "condition":[
              "$X:HTMLAnchorElement($hrefVal:href)",
              "new RegExp(/showMessage\?fid=Inbox/).test($hrefVal)"
              ],
  "actions":["append($X.textContent)"]
}
```

3 The Engine

There is an important difference between the actual rule engines and the JavaScript Rule Engine implementing the JSON Rules language for at least two reasons: events facts are ***not static facts*** that require usual operation such as: `delete`, `update` on the Working Memory but they are ***dynamic facts***. They are dynamically consumed based on the appearance time. Second the whole engine is a **live** system: it is **reactive** because reacts based on events and it is **proactive** for by itself produces events.

The project is hosted on Google Code platform[2].

3.1 How you can use the engine

The engine is programmed in JavaScript and can be used as any JavaScript framework. Basically, the lifetime of the rule engine is in the scope of the lifetime of the current DOM inside the browser. Simple steps to make it run are:

1. Load the engine in your page:

```
<script type="text/javascript"
  src="http://www.domain.com/jsonRulesEngine_Version.js">
</script>
```

[1] `http://www.w3.org/TR/DOM-Level-3-Events/`

[2] `http://jsonrules.googlecode.com`

2. Create an instance of the engine:

```
var jsonRulesEngine=new org.jsonrules.JSONRulesMainSystem();
```

3. Run the engine by calling `run()` with the URI of location of the repository as input parameter:

```
jsonRulesEngine.run("http://www.domain.com/rulesRepo.txt");
```

When the engine and the rulesets are available, the main things that happen are:

- When an event is raised, the `EventManager` catches that event. Then the `EventManager` checks the `ActionProcessor` state.
- If the *ActionProcessor* is running, then the `EventManager` stores the event in the queue of events that the `InferenceEngine` must later on process.
- However if the `ActionProcessor` is idle then the `EventManager` sends a message to the `InferenceEngine` containing the queue of events that must be processed. The `InferenceEngine` responds back to the `EventManager`, and informs it that it has received/consumed the queue such that the `EventManager` can reset its own queue.
- Events are processed one by one. For each event rules triggered by that event will be matched against the `WorkingMemory`. The action of each executable rule is added to the list of executable actions (to be processed by the `ActionProcessor`) according with possible priority of rules.
- The list of executable actions it is send to the `ActionProcessor`, to execute them. Any JavaScript functions can be called in the rule actions' part.

4 Conclusions

This paper describes shortly the general ideas behind an ECA rule-based and forward chaining engine for browsers.

References

1. Adrian Giurca and Emilian Pascalau. JSON Rules. In *Proceedings of the Proceedings of 4th Knowledge Engineering and Software Engineering, KESE 2008*, volume 425, pages 7–18. CEUR Workshop Proceedings, 2008.
2. Emilian Pascalau and Adrian Giurca. A Rule-Based Approach of Creating and Executing Mashups. In *Proceedings of the 9th IFIP Conference on e-Business, e-Services, and e-Society (I3E 2009)*, LNCS. Springer, 2009. forthcoming.

Author Index

Baumeister, Joachim 33

Cañadas, Joaquín 13

Forcher, Björn 46
Furmańska Weronika T. 25

Giurca, Adrian 1, 63

Kaczor, Krzysztof 57

Nalepa, Grzegorz J. 25, 57

Palma, José 13
Pascalau, Emilian 1, 63
Puppe, Frank 33

Reutelshoefer, Jochen 33
Roth-Berghofer, Thomas 46

Schumacher, Kinga 46

Ténez, Samuel 13