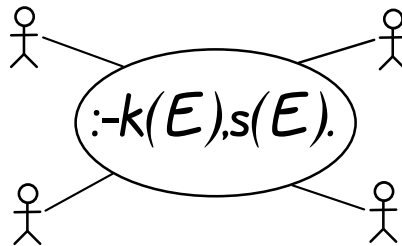


7th Workshop on
Knowledge Engineering
and Software Engineering (KESE7)

at the 14th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2011), La Laguna, Tenerife, Spain, November 10, 2011

Joaquín Cañadas, Grzegorz J. Nalepa, Joachim Baumeister
(Editors)



UNIVERSIDAD DE ALMERÍA



Technical Report TR-2011/1,
Department of Languages and Computation. University of Almería,
Almería, Spain, 2011

Preface

Joaquín Cañadas, Grzegorz J. Nalepa and Joachim Baumeister

Dept. of Languages and Computation. University of Almeria.
Agrifood Campus of International Excellence, ceiA3. Almeria, Spain
jjcanada@ual.es

—
AGH University of Science and Technology
Kraków, Poland
gjn@agh.edu.pl

—
Intelligent Systems (Informatik 6)
University of Würzburg
Würzburg, Germany
joba@uni-wuerzburg.de

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 seventh workshop on Knowledge Engineering and Software Engineering (KESE7) was held at the Conference of the Spanish Association for Artificial Intelligence (CAEPIA-2011) in La Laguna (Tenerife), Spain, 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
- 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 engineering
- Process models in AI applications
- Software requirements and design for AI applications
- AI approaches in software engineering process
- Declarative, logic-based approaches
- Constraint programming approaches

This year, we received contributions focussing on different aspects of knowledge engineering: Prieto et al. present *OntoMetaWorkflow*, a generic ontology to

represent canonical workflow terms in the domain of administrative processes. The flowchart-based language DiaFlux and a collection of anomalies that can occur when using it for knowledge base development are discussed by Hatko et al. The contribution of Kluza et al. elaborates on a hybrid and hierarchical approach to formal verification of BPMN models, using the Alvis modeling language and the XTT2 knowledge representation. Sagrado et al. define a three-layer architecture to provide a seamless integration between Knowledge Engineering and Requirement Engineering, enhancing requirement validation and requirement selection tasks in software development projects with knowledge-based techniques. Pascalau discusses a new perspective for the mashup concept introducing a new perspective on mashups as behavior in context(s). Cañadas et al. introduce a model-driven method for generating rich Web user interfaces for data-intensive Web applications from OWL domain ontologies.

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, one presentation about current tools was given: Adrian and Nalepa present a semantic wiki called *Loki* which enables a strong rule-based reasoning with semantic annotations mapped to Prolog knowledge base.

Two of the workshop contributions, Prieto et al. and Sagrado et al., are selected for being included in the Selected Papers Volume of CAEPIA 2011 proceedings: Lozano, J.A., Gámez, J.A., Moreno, J.A. (eds) LNAI series, *Current Topics in Artificial Intelligence. 14th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2011, La Laguna, Spain, November 8-11, 2011, Selected Papers*. Here extended abstracts of both publications are provided. The rest of contributions are published in the CAEPIA 2011 conference proceedings published by the Spanish Association for Artificial Intelligence.

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 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 the organizers of the CAEPIA 2011 conference for hosting the KESE7 workshop.

Joaquín Cañadas
Grzegorz J. Nalepa
Joachim Baumeister

Workshop Organization

The 7th Workshop on Knowledge Engineering and Software Engineering
(KESE7)
was held as a one-day event at the
14th Conference of the Spanish Association for Artificial Intelligence
(CAEPIA 2011)
on November 10, 2011, La Laguna, Spain.

Workshop Chairs and Organizers

Joaquín Cañadas, University of Almeria, Spain
Grzegorz J. Nalepa, AGH UST, Kraków, Poland
Joachim Baumeister, University Würzburg, Germany

Programme Committee

Isabel María del Águila, University of Almeria, Spain
Klaus-Dieter Althoff, University Hildesheim, Germany
Antonio B. Bailón, University of Granada, Spain
Joachim Baumeister, University Würzburg, Germany
Manuel Campos, University of Murcia, Spain
Joaquín Cañadas, University of Almeria, Spain
Jesualdo Tomás Fernández-Breis, University of Murcia, Spain
Adrian Giurca, BTU Cottbus, Germany
Francisco Guil, University of Almeria, Spain
José M. Juarez, University of Murcia, Spain
Jason Jung, Yeungnam University, Korea
Rainer Knauf, TU Ilmenau, Germany
Carmen Martínez-Cruz, University of Jaen, Spain
Grzegorz J. Nalepa, AGH UST, Kraków, Poland
José Palma, University of Murcia, Spain
José del Sagrado, University of Almeria, Spain
Dietmar Seipel, University Würzburg, Germany
Fernando Silva Parreiras, University of Koblenz-Landau, Germany
Ioannis Stamelos, Aristotle University of Thessaloniki, Greece
Rafael Valencia-García, University of Murcia, Spain

Table of Contents

OntoMetaWorkflow: An Ontology for Representing Data and Users in Workflows (extended abstract)	1
<i>Alvaro Prieto, Adolfo Lozano-Tello, José Luis Redondo-García</i>	
Anomaly Detection in DiaFlux Models	5
<i>Reinhard Hatko, Gritje Meinke, Joachim Baumeister, Stefan Mersmann, Frank Puppe</i>	
Proposal of a Hierarchical Approach to Formal Verification of BPMN Models Using Alvis and XTT2 Methods	15
<i>Krzysztof Kluza, Grzegorz J. Nalepa, Marcin Szpyrka, Antoni Ligęza</i>	
Architecture for the Use of Synergies between Knowledge Engineering and Requirements Engineering (extended abstract)	25
<i>José del Sagrado, Isabel M. del Águila, Francisco J. Orellana</i>	
Mashups: Behavior in Context(s)	29
<i>Emilian Pascalau</i>	
Model-Driven Rich User Interface Generation from Ontologies for Data-Intensive Web Applications	39
<i>Joaquín Cañadas, José Palma, Samuel Túnez</i>	
Loki – Presentation of Logic-based Semantic Wiki (tool presentation)	49
<i>Weronika T. Adrian, Grzegorz J. Nalepa</i>	

OntoMetaWorkflow: An Ontology for Representing Data and Users in Workflows – Extended Abstract¹

Alvaro E. Prieto, Adolfo Lozano-Tello, José Luis Redondo-García

Quercus Software Engineering Group, University of Extremadura, av. Universidad s/n
10071, Spain
{aeprieto, alozano, jluisred}@unex.es

Abstract. Administrative processes are a type of business process commonly used in public institutions and large companies. These processes are frequently reused because there are often similar processes within the organizations. The use of ontologies for modeling the workflows of administrative processes can provide significant improvements in this reuse process. In this paper, we describe OntoMetaWorkflow, a generic ontology to represent canonical workflow terms in the domain of administrative processes.

Keywords: business process, administrative process, workflows, ontologies, WEAPON.

1 Introduction

Administrative processes are generally used in administrative or legal ambits. They are characterized by being initiated by a user and which must be attended to or evaluated by other different users following a perfectly defined protocol for data, times and agents involved. These processes are often defined generically in the level of management of the organizations but must be reused in the lower levels in order to be applied in them. Examples could be the management of public contest bids, loan application procedures or a simple holiday application.

They can be managed by simple Workflow Management Systems (hereinafter referred to as WfMS) with features that facilitate to share and reuse this type of process. The use of ontologies as a basis for this type of WfMS could be very useful due to their characteristics of complete and precise representation of terms.

An appropriate case of application to reuse processes is the WfMS model based on ontologies was proposed in [1]. This model provided a generic ontology described in [1] as the basis of workflow representation. We have restructured the ontology and the WfMS model to improve the reuse process.

¹ This is a long abstract of the paper published in Lozano, J.A., Gámez, J.A., Moreno, J.A. (eds) LNAI series, Current Topics in Artificial Intelligence. 14th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2011, La Laguna, Spain, November 8-11, 2011, Selected Papers.

This paper is structured as follows: section 2 enumerates works that use ontologies in WfMS, section 3 presents a brief description of the redefinition of the WfMS [1], now called WEAPON (Workflow Engine for Administrative Processes based on ontologies), and, section 4 describes the new ontology, called OntoMetaWorkflow.

2 Use of Ontologies in WfMS

The application of ontologies to WfMS have been used previously in several approaches as [2,3,4,5] and a recent survey is available in [6].

Unlike the previous approaches, this paper presents an ontology for representing administrative processes together with their activities, domain data and users involved. Although several models and languages of workflow representation exist [7,8], the application of ontologies can provide the following advantages:

- The users, following methodologies for building ontologies, can obtain complete, precise and shared definitions of administrative process workflows.
- The data and the users of a process can be changed without modifying the definition of the data managed by activities or the definition of the workflow.
- Workflows represented in ontologies, are more easily reusable although the reuse process may involve some effort in the search, selection and, in some cases, adaptation to the new system.

3 WEAPON: Workflow Engine for Administrative Processes based on Ontologies

WEAPON is a WfMS that proposes how a workflow designer must define, on one hand, the taxonomy of relevant data of the domain and the taxonomy of users which can participate in the workflow and, on the other hand, the activities that the process contains together with the identification of which type of user defined can perform them and the data managed by every activity. WEAPON uses ideas of ontology field together with ideas of traditional WfMS and the Case Handling approach [9].

The architecture of WEAPON presents a series of interrelated components (a graphical representation is available in ²). These components are:

1. OntoMetaWorkflow³, contains the terms that form the workflows of administrative processes and their relationships. This ontology, represented in OWL Language, is built adapting the definitions of workflow elements provided by the WfMC [10] to the specific characteristics of administrative processes. It has been developed following the METHONTOLOGY methodology [11].
2. OntoDD, an ontology of the domain data and workflow participants built following the specifications of OntoMetaWorkflow. It imports the concepts defined in

² <http://quercusseg.unex.es/weapon/>

³ <http://quercusseg.unex.es/weapon/?download=OntoMetaWorkflow.owl>

OntoMetaWorkflow and must contain, firstly, the taxonomy of data which will be used in the corresponding domain and, secondly, the taxonomy of the possible workflow participants. As example, the OntoDD ontology for a loan application domain is available in ⁴.

3. OntoWF represents the workflow of the administrative process that will be managed by the WfMS. It contains the concrete workflow of the administrative process, including its properties, the activities that it contains the order of execution of said activities, the relevant data of OntoDD that will be shown or modified in an activity and the participants which can perform every activity. As example, the OntoWF ontology for a loan application process is available in ⁵.
4. WEAPON Designer, is the tool that allows users to combine WF-Net [12] representation with OntoMetaWorkflow and the OntoDD of a domain in order to design the OntoWF Ontology for a specific administrative process.
5. WEAPON Manager, is the web application that reads OntoDD and OntoWF ontologies and generates the web forms and the database that manage the workflow of the administrative process.

4 OntoMetaWorkflow

The different definition elements of OntoMetaWorkflow are classified into two types (a graphical representation of OntoMetaWorkflow is available in ⁶):

1. Definition elements of OntoDD: are used to define the classes and properties that represent the common data and the potential users of all similar processes within a domain. These elements are the *Domain Data*, *Workflow Participant* and *Root* classes. *Domain Data* stores common data of all instances of an administrative process and has the *External Document* and *Location* attributes. The *Workflow Participant* class stores the users involved in the process and has Id, Password, Name, Surname and Email attributes. The *Root* subclass is a special class that can administer the WEAPON Manager WfMS.
2. Definition elements of OntoWF: are used to define the classes and properties that represent a particular process, that is, the sequential flow of activities and their relationships with the elements of the domain defined in OntoDD. These elements are the *Administrative Process* and *Activity* classes. *Administrative Process* class is used for representing the process managed by the WfMS and has defined the *Generated By* relationship. The *Activity* class represents a logical unit of work and has defined the *Is Performed By* and *Before* relationships and the *Before Control Flow Pattern*, *Select Class Of Domain Data*, *Show Class of Domain Data*, *Select Instance Of Domain Data*, *Show Instance of Domain Data*, *Fill In Instance Attributes of Process*, *Show Instances Attribute*, *Days Time Frame*, *Day Notice* and *Activity Description* attributes.

⁴ http://quercusseg.unex.es/weapon/?download=OntoDD_LoanApplication.owl

⁵ http://quercusseg.unex.es/weapon/?download=OntoWF_LoanApplication.owl

⁶ <http://quercusseg.unex.es/weapon/?OntoMetaWorfklow>

5 Conclusions

We have presented OntoMetaWorkflow ontology and WEAPON. OntoMetaWorkflow is an ontology which specifies the elements and rules that define workflows according to the standards and recommendations of the WfMC. OntoMetaWorkflow and the methods of WEAPON have been tested in several domains, mainly in administrative processes of University of Extremadura. They work properly with administrative processes that are fully oriented to humans and, specially, in those processes that involve submitting some type of application to be considered at different stages, where different participants need to handle current information of a dossier in order to provide new data in the corresponding activity.

Acknowledgments. This work has been developed under support of Ministerio de Ciencia e Innovacion Project (TIN2008-02985), FEDER, Junta de Extremadura and Plan de Iniciacion a la Investigacion, Desarrollo Tecnologico e Innovacion 2010 de la Universidad de Extremadura (ACCVII-04).

References

1. Prieto, A.E., Lozano-Tello, A.: Use of Ontologies as Representation Support of Workflows Oriented to Administrative Management. *J. Netw. Syst. Manag.* 17, 3, 309--325 (2009)
2. Vieira, T.A.S.C., Casanova, M.A., Ferrão, L.G.: On the design of ontology-driven workflow flexibilization mechanisms. *J. Braz. Comp Soc.* 11, 2, 33--43 (2006)
3. Gasevic, D., Devedzic, V.: Petri net ontology. *Knowl-Based Syst.* 19, 4, 220--234 (2006)
4. Haller, A., Oren, E., Kotinurmi, P.: m3po: An Ontology to Relate Choreographies to Workflow Models. In: 3th IEEE International Conference on Services Computing, pp. 19--27. IEEE Computer Society, Los Alamitos, CA (2006)
5. Abramowicz, W., Filipowska, A., Kaczmarek, M., Kaczmarek, T.: Semantically enhanced Business Process Modelling Notation. In: 2nd Workshop on Semantic Business Process and Product Lifecycle Management, pp. 88--91. CEUR-WS, Innsbruck, Austria (2007)
6. Hoang, H.H., Tran, P.C., Le, T.M.: State of the Art of Semantic Business Process Management: An Investigation on Approaches for Business-to-Business Integration. In: Nguyen, N.T., Le, T.M., Świątek, J. (eds.) *Intelligent Information and Database Systems*. LNCS, vol. 5991, pp. 154--165. Springer Berlin Heidelberg (2010)
7. Aalst, W.M.P.V.D., Hofstede, A.H.M.T.: YAWL: yet another workflow language. *Inform. Syst.* 30, 4, 245--275 (2005)
8. OMG: Business Process Model and Notation (BPMN) 1.2 (2009)
9. Aalst, W.M.P.V.D., Weske, M.: Case handling: a new paradigm for business process support. *Data Knowl. Eng.* 53, 2, 129--162 (2005)
10. Hollingsworth, D.: The Workflow Reference Model. Document Number TC00-1003 Document Status - Issue 1.1. (1995)
11. Gómez-Pérez, A., Fernández-López, M., Corcho, O.: *Ontological Engineering. With Examples from the Areas of Knowledge Management, e-Commerce and the Semantic Web*. Springer-Verlag, London (2004)
12. Aalst, W.M.P.V.D., Hee, K.V.: *Workflow Management - Models, Methods and Systems*. MIT Press, Cambridge (2002)

Anomaly Detection in DiaFlux Models

Reinhard Hatko¹ and Gritje Meinke² and Joachim Baumeister³ and Stefan Mersmann² and Frank Puppe¹

¹ University of Würzburg, Institute of Computer Science, Dept. of Artificial Intelligence and Applied Informatics
97074 Würzburg, Germany

{hatko, puppe}@informatik.uni-wuerzburg.de

² Dräger Medical GmbH, 23558 Lübeck, Germany

{gritje.meinke, stefan.mersmann}@draeger.com

³ denkbares GmbH, Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
joachim.baumeister@denkbares.com

Abstract. In recent years, the use of graphical knowledge representations more and more proved to be suitable for building diagnostic and therapeutic knowledge systems. When building such systems, the quality assurance of the knowledge base is an integral part of the development process. In this paper, we present the flowchart-based language DiaFlux and we describe a collection of anomalies, that can occur when using the language for knowledge base development. The naming of many shown anomalies was motivated by the experiences made in real-world projects.

1 Introduction

In recent years, intelligent systems have been established in a variety of domains. When building such systems the developers no longer depend on pure rule-based representations, but more and more use graphical approaches that often allow for a more intuitive knowledge elicitation process. In the medical domain, for instance, workflow-oriented representations emerged in the last years to build systems based on existing guidelines and standard operating procedures (SOPs), see for instance [1].

In an industrial setting, the development of such knowledge bases is integrated in a predefined knowledge engineering process, that shows similar phases to general software engineering processes, see for instance [2,3]. All these process models also propose a quality assurance phase, where the developed artifact is tested by validation and verification methods. Here, usually the expected system behavior is tested with regression-based methods, such as empirical tests [4], but also checks at the component level are performed. The most commonly used verification method for component-based tests is the detection of (already known) anomalies. In Software Engineering such anomalies are related to object-oriented metrics [5] and bad smells [6]. Some typical examples for general anomalies are cyclic dependencies between classes and packages, infinite recursion, and

long/unmaintainable methods. The automated detection by a static code analysis and the (manual) elimination of such anomalies can prevent serious malfunctions of the built application.

It is easy to see, that the ideas of anomalies in general software code can be transferred to the artifacts produced in a knowledge engineering process. Here, the knowledge base is investigated in order to find deficient parts of the knowledge. In the past, verification methods for detecting anomalies in different knowledge representations were introduced, for instance see [7,8].

Approaches for the verification of workflow models are described, e.g., in [9]. In addition, some of the anomalies we identified represent a mixture of data- and control-flow anomalies and also involve a Truth Maintenance System. In this paper, we introduce the workflow-based knowledge representation DiaFlux for building diagnostic and therapeutic knowledge systems. The language is presented in Section 2 and Section 3 describes possible anomalies. We report a small case study in Section 4 and conclude the paper with a discussion in Section 5.

2 Graphical Knowledge Models with DiaFlux

This section first describes the application scenario. Then, we introduce the representation language *DiaFlux*.

2.1 Application Scenario

DiaFlux is a graphical guideline language intended to be used in mixed-initiative devices, that continuously monitor, diagnose, and treat a patient in the setting of an Intensive Care Unit (ICU). The clinical user interacts with such a semi-closed loop system during the care process. Actions on the patient can be initiated by both parties, the clinician and the device. Continuous reasoning is performed, as some data is continuously available as a result of the monitoring task. An execution environment for automated clinical care in ICUs and the implementation of a guideline for weaning from mechanical ventilation are presented in [10].

2.2 Language Description

Two kinds of knowledge have to be effectively combined for the specification of a clinical protocol, namely declarative and procedural knowledge [11]. The declarative part encompasses the facts and their relationships. The procedural knowledge reflects how to perform a task, i.e., the correct sequence of actions. The declarative knowledge particularly consists of the terminology, i.e., findings, solutions, and sometimes also therapies and their interrelation. The procedural knowledge is responsible for the decision which action to perform in a given situation, e.g., asking a question or carrying out a test. The appropriate sequence of actions is mandatory for efficient diagnosis and treatment, as each action has a cost (monetary or associated risk) and a benefit (for establishing or excluding currently considered solutions) associated with it. For the representation of the

procedural aspects, guideline languages employ different kinds of Task Network Models [1]. They constrain the ordering of decisions and actions in a guideline plan. Flowcharts are a common formalism to explicitly express this control flow. In DiaFlux models, a domain-specific ontology represents the declarative knowledge. It contains the definition of findings and solutions. This application ontology extends the task ontology of diagnostic problem-solving, as described in [12]. Due to its strong formalization, it provides the semantics necessary for the execution of the guidelines. The procedural knowledge is represented by flowcharts, that consist of nodes and edges. Different types of actions are represented by nodes. Connecting edges create possible sequences of actions. To constrain these sequences, an edge can be guarded by a condition that evaluates the state of the current session and thus guides the course of the care process.

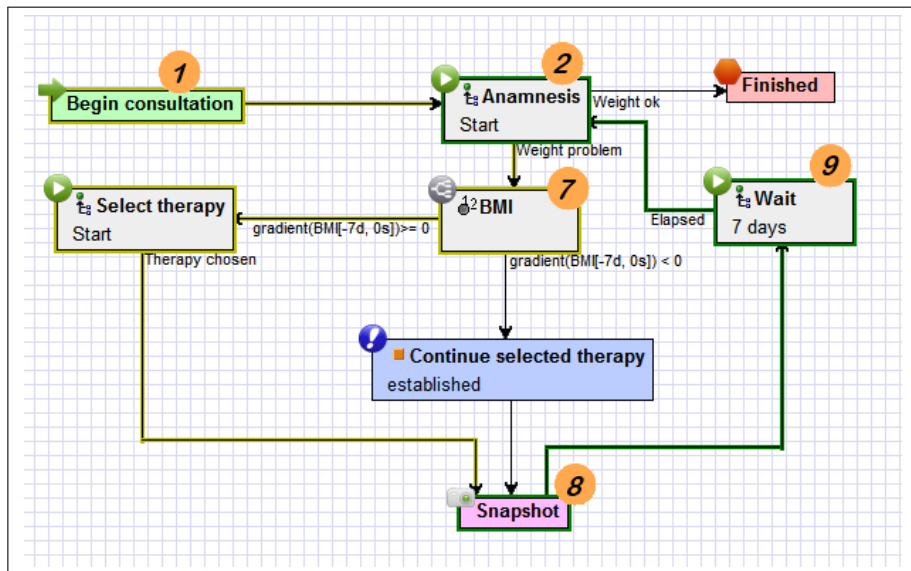


Fig. 1. The main model and starting point of a protocol for monitoring and treating overweight. The state of the current testing session is highlighted in green and yellow colors (black and grey in this figure, respectively).

In the following, we give a simple example of a protocol for the diagnosis and treatment of overweight, modeled in DiaFlux.

Figures 1 and 2 show parts of a protocol for the diagnosis and treatment of overweight modeled in DiaFlux. When a consultation session starts, the main module, as depicted in Figure 1, is activated. The execution begins at the *start node* (1), labeled “Begin consultation”. It points to the *composed node* “Anamnesis” (2). When this node is reached, the according submodule (cf. Figure 2)

is called and its start node labeled “Start” is activated. The execution of the main module awaits the completion of the called submodule. Reaching the *test node* “Height” (3) data is acquired from the user. After entering the value for body height, the execution can continue to the next test node “Weight”. As the weight is supposed to change from one session to the next, this test node acquires new data each time it is activated. Therefore, the specific testing action used is “always ask” instead of “ask”. The first one triggers data acquisition even for inputs that are already known in order to update their value. After the value for “Weight” has been entered, the *abstraction node* (4) calculates the body mass index (BMI) from the acquired data and assigns the value to the input “BMI”. An appropriate next action is chosen depending on the value of the BMI. For a value contained in the range of [25; 30[the execution progresses to the *solution node* (5) which establishes the solution “Overweight”. The following *exit node* (6) labeled “Weight problem” terminates the execution of the module. The control flow then returns to the superordinate module. For other values of “BMI” the appropriate solution is established and the according exit node is returned as result of the “Anamnesis” protocol.

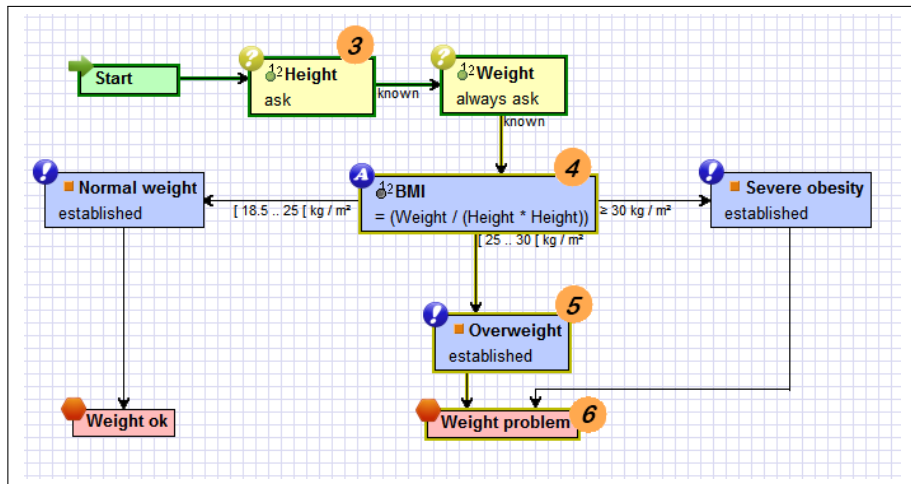


Fig. 2. The anamnesis module for acquiring data and establishing the current diagnosis.

Upon completion of the “Anamnesis” module, the appropriate successor node is chosen based on the returned result. In case of “Weight ok” the execution of the protocol ends by reaching the exit node “Finished”, as there is no superordinate module to return to. Otherwise, a proper treatment is chosen based on the history of values of the BMI. The *decision node* (7) tests the gradient of BMI values. For a declining BMI (i.e., the patient is losing weight), the previously selected therapy is continued. Otherwise, another therapy is chosen within the module

“Select Therapy”¹. Both paths reach the *snapshot node* (8). On activation of this node, the execution state of the protocol is saved and truth maintenance will not retract any conclusion beyond this point. Furthermore all active nodes on the incoming path are deactivated, to allow their repeated execution. Next, the execution is suspended by the *wait node* (9), until the given time of 7 days has lapsed. Afterwards, a second anamnesis is conducted and the current BMI is calculated based on the newly acquired body weight. If it has decreased, so will the BMI and the current therapy is continued. Otherwise, a new therapy is selected and applied until a normal body weight is obtained.

A more detailed description of the DiaFlux language and the execution engine can be found in [13].

3 Anomaly Detection

There exists a large body of research concerning the detection of anomalies by verification methods, for instance for rule bases [14], for ontologies [15], for mixed verification of rules and ontologies [8]. In general, we distinguish the following types of anomalies for knowledge bases:

1. **Redundancy** defining duplicate or subsuming elements of the knowledge base
2. **Inconsistency** caused by contradicting elements of the knowledge base
3. **Missing knowledge** are absent parts of the knowledge base, that can prevent the proper execution of the knowledge
4. **Deficiency** comprising parts of the knowledge base, that worsen the design of the knowledge

In the following, we discuss these types in more detail and we introduce particular anomalies, that explain redundant, inconsistent, deficient, and missing knowledge especially in DiaFlux models.

It is important to notice, that the following presentation of anomalies is not an exhaustive set but more or less a collection of problems, that occurred during the development of industrial knowledge bases.

3.1 Redundancy

Redundant knowledge may be removed from the knowledge base without change in the semantics of the derivation behavior. Each found anomaly, however, needs to be considered carefully by a human knowledge engineer, since some kinds of redundancy can be used to increase the robustness of the knowledge base.

¹ The gradient of a single value is 0 and a therapy is chosen for the first time.

Redundant Calculation Abstraction nodes can be used to assign a value to a finding. That value can either be a constant number or can be calculated by a formula, aggregating the values of the others findings. The assignment of a constant value is redundant when the same value is assigned more than once on a given path. The assignment of a value derived from a formula is redundant, if the second calculation will yield the same result. This is the case if the second abstraction uses the same formula and if there is no path between the first and the second calculation that leads to the acquisition of new values for the findings used in the calculation.

Redundant Test Depending on the frequency the values of a finding may change, two different kinds of actions can be used for *test nodes*, “ask” and “always ask”, respectively. The first one triggers the acquisition of data only if no value has been assigned to the finding so far. The latter demands new data each time the node is activated in the flowchart. If two test nodes are located on a connecting path and trigger an “ask” action on the same finding, the second test action is ignored and therefore redundant. In case the second node has more than one outgoing edge with different guards, the developer should consider to convert the node to a *decision node*.

3.2 Inconsistency

Inconsistent knowledge often yields unexpected and contradictory inferences during execution. Detected inconsistencies should be investigated thoroughly by the knowledge engineer and be considered for elimination in most cases.

Inconsistent Calculation As described in the anomaly *Redundant Calculation*, abstraction nodes can be used to assign a value to a finding. The assignment contains either a constant value or a formula that is evaluated. Such a calculation is inconsistent, if different values are assigned to one finding on a single, connected path of nodes. In the worst case, the assignment of the second value may force the truth maintenance system to illegally retract the followed path until the first assignment, and thus creates a truth maintenance cycle.

Inconsistent Test Action Two different types of testing actions are provided in DiaFlux for collecting data. For findings containing high frequency data (e.g. “blood pressure” in the medical domain), the testing action “always ask” is appropriate to be used; the action “ask” is appropriate for the single acquisition of data (e.g. when asking the age or sex of a patient). Using both types of testing actions for the same finding most likely hints to a design flaw. If the finding contains high frequency data, the value of the finding will not be updated upon reaching the node, that performs the “ask” action. Therefore an old value will be used, instead of acquiring new data. In the case of low frequency data, the value for the finding is acquired more often than necessary, if the action “always ask” is used.

3.3 Missing Knowledge

Some anomalies may point to unfinished areas of the knowledge base, for instance elements of the knowledge that are never used in problem-solving sessions.

Uninitialized Value Values of findings are calculated in the DiaFlux representation by using abstraction nodes. To conduct such a calculation, proper values have to be available for all findings that are included in the calculation. If at least one necessary finding is not acquired (or calculated itself) on at least one path leading to the abstraction node, then the calculation will not succeed and the execution of the path may stop at the abstraction node.

Missing Start Node A flowchart in DiaFlux can have several distinct entry points. Each one must begin with a *start node*. A flowchart not defining at least one start node, cannot be activated during execution and thus is isolated from the rest of the knowledge base.

Unconnected Node Every flowchart defines a process that begins at a *start node* and ends at an *exit node*. The activation of the nodes in between depends on the connecting edges and their respective guards. Any node (except a *start node*) that is missing an incoming edge cannot be activated during the problem-solving process. All successors of such a node are also unreachable unless they have an alternative incoming edge, which is itself connected to at least one *start node*.

Open Path End Every possible path in a flowchart has to be terminated by an *exit node*. Although, an open path end does not influence the execution of this particular flowchart, it will prevent the continuation of a superordinate flowchart. Thus, the flowchart is not returning to the super-flowchart, that called it. After reaching a *composed node* during execution, the calling flowchart awaits the termination of the called module by an *exit node*. If this does not exist, then the execution of the calling flowchart will not continue.

No Startup Flow Defined The execution of the knowledge base begins in a distinct flowchart, which has to be marked as *autostart* by the knowledge engineer. If no flowchart is marked accordingly, then none is activated at the start of a problem-solving session. Therefore, the execution will end immediately.

Unused Flowchart For improving the structure of the knowledge base, flowcharts can be nested. *Composed nodes* allow the execution of another flowchart module. A flowchart, that is neither marked as *autostart* nor is called by any *composed node* will never be executed during runtime.

Incompleteness of Edge Guards The definition of edge guards allows to select one of multiple outgoing paths at a node, depending on the current value of a finding. As the execution will continue only along an edge whose guard is evaluated to *true*, the entirety of guards defined at one node has to cover the complete range of possible values of the examined finding. Otherwise, the execution of the flowchart will stop at this node, if the current value does not match with an edge guard.

3.4 Deficiency

Deficiencies point to subtle parts in the knowledge base, that may benefit from a design improvement. The existence of such an anomaly, however, often does not affect the reasoning behavior in a bad manner.

Dead Path The possible paths through a flowchart are given by the edges between nodes. Every edge can be guarded by a condition that evaluates the values of findings entered into the system. An edge is activated, if its starting point is active and its condition evaluates to *true*. If a finding is used multiple times on a single path, then the guards at later edges have to be consistent to the possible values at that point. Otherwise such edges cannot be activated for certain values. An example is given in Figure 3.

Impossible Path When new findings are entered into the system, a truth maintenance system checks the state of all flowcharts. If the value of a finding has changed, all edges and nodes change their activation state according to the new values. In case an *abstraction node* calculates a value for a finding, that is used to guard an edge in the active path, the calculated value must not contradict that guard. Otherwise, the truth maintenance system will collapse the path to the *abstraction node* undoing its calculation. Therefore, the path starting at the *abstraction node* is impossible to continue.

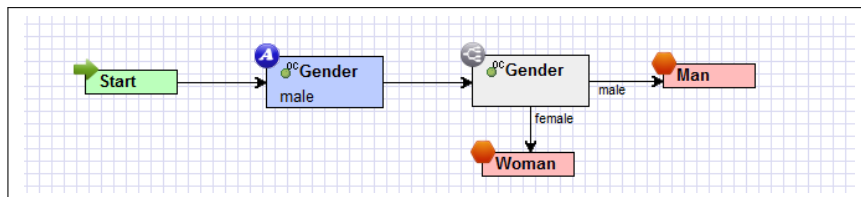


Fig. 3. A minimal example of a Dead Path. After setting the question “Gender” to “Male”, the following decision node branches depending on its value. As it can only be “Male”, the path leading to the exit node “Woman” can never be taken, and is therefore dead.

Disjointness of Edge Guards The guards on the outgoing edges of every node must be disjoint with respect to the possible outcomes of a node. If the domains of guards overlap, all belonging edges will be activated for according values. This easily happens, when defining intervals at a decision node that examine a numerical finding.

In this section, we introduced a selection of anomalies that can occur in DiaFlux knowledge bases. In the next section, we describe an implementation of a part of the shown anomalies and we report on some experiences.

4 Case Study

The DiaFlux development environment is integrated into the Semantic Wiki KnowWE [12]. KnowWE is a wiki aimed at building intelligent systems, offering methods to capture and execute strong problem-solving knowledge. A Continuous Integration (CI) tool supports the modeler during the development of the knowledge base by executing a configurable set of tests after each edit. The results of the recent build of the knowledge base are indicated to the user in an unintrusive manner. A detailed report is available on demand. The frequently running test procedures help to find modeling errors at an early stage.

We recently integrated detection algorithms for selected anomalies as described in Section 3 into the CI tool. The system was used in a couple of projects and received very positive feedback, from unexperienced as well as advanced users. A common mistake among modelers, that are new to the DiaFlux language, is to miss marking the *autostarting* flowchart. As a result the knowledge base seems to simply do nothing. In more complex knowledge bases, that are hierarchically structured and contain different possible paths of execution, the detection of anomalies like *Uninitialized Value* or *Dead Path* is very helpful as those are not only tested within each flowchart module but also across their boundaries along paths through composed nodes.

5 Conclusions

The development of knowledge-based software systems is similar to general software engineering approaches. We motivated that today's knowledge bases are often built using workflow-based languages; this especially holds in the medical domain, where existing guidelines and standard operating procedures are transferred into computer-interpretable models. In this paper, we discussed the problem of quality assurance of such models and we described the detection of anomalies in the models as an important aspect of quality assurance. We described the practical guideline language DiaFlux by an example protocol for overweight treatment. Furthermore, we introduced a selection of anomalies for this language. The selection of these anomalies is not exhaustive, but was motivated by our experiences in the development of industrial knowledge bases.

In the future, we plan to define a more exhaustive set of anomalies, including temporal ones, and relate the particular artifacts to anomalies already known in classical verification research. Often, a found defect is the start of a refactoring of the knowledge base. We are currently working also on refactoring methods for DiaFlux models, that are used to eliminate found deficiencies but also other kinds of anomalies.

References

1. Peleg, M., Tu, S., Bury, J., Ciccarese, P., Fox, J., Greenes, R.A., Miksch, S., Quaglino, S., Seyfang, A., Shortliffe, E.H., Stefanelli, M., et al.: Comparing computer-interpretable guideline models: A case-study approach. *JAMIA* **10** (2003) 2003
2. Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., de Velde, W.V., Wielinga, B.: *Knowledge Engineering and Management - The CommonKADS Methodology*. 2 edn. MIT Press (2001)
3. Baumeister, J., Seipel, D., Puppe, F.: Agile development of rule systems. In Giurca, Gasevic, Taveter, eds.: *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. IGI Publishing (2009)
4. Baumeister, J.: Advanced empirical testing. *Knowledge-Based Systems* **24**(1) (2011) 83–94
5. Simon, F., Steinbruckner, F., Lewerentz, C.: Metrics based refactoring. In: *Software Maintenance and Reengineering, 2001. 5th European Conference on*. (2001) 30–38
6. Fowler, M.: *Refactoring. Improving the Design of Existing Code*. Addison-Wesley (1999)
7. Ayel, M., Laurent, J.P.: *Validation, Verification and Test of Knowledge-Based Systems*. Wiley (1991)
8. Baumeister, J., Seipel, D.: Anomalies in ontologies with rules. *Web Semantics: Science, Services and Agents on the World Wide Web* **8**(1) (2010) 55–68
9. Aalst, W.M.P.v.d.: Workflow verification: Finding control-flow errors using petri-net-based techniques. In: *Business Process Management, Models, Techniques, and Empirical Studies*, London, UK, Springer-Verlag (2000) 161–183
10. Mersmann, S., Dojat, M.: SmartCaretm - automated clinical guidelines in critical care. In: *ECAI'04/PAIS'04: Proceedings of the 16th European Conference on Artificial Intelligence, including Prestigious Applications of Intelligent Systems*, Valencia, Spain, IOS Press (2004) 745–749
11. de Clercq, P., Kaiser, K., Hasman, A.: Computer-interpretable guideline formalisms. In ten Teije, A., Miksch, S., Lucas, P., eds.: *Computer-based Medical Guidelines and Protocols: A Primer and Current Trends*. IOS Press, Amsterdam, The Netherlands (2008) 22–43
12. Baumeister, J., Reutelshoefer, J., Puppe, F.: KnowWE: A semantic wiki for knowledge engineering. *Applied Intelligence* (2011)
13. Hatko, R., Baumeister, J., Belli, V., Puppe, F.: Diaflux: A graphical language for computer-interpretable guidelines. In: *KR4HC'11: Proceedings of the 3th International Workshop on Knowledge Representation for Health Care*. (2011)
14. Preece, A., Shinghal, R.: Foundation and application of knowledge base verification. *International Journal of Intelligent Systems* **9** (1994) 683–702
15. Gómez-Pérez, A.: Towards a framework to verify knowledge sharing technology. *Expert Systems with Applications* **11**(4) (1996)

Proposal of a Hierarchical Approach to Formal Verification of BPMN Models Using Alvis and XTT2 Methods*

Krzysztof Kluza, Grzegorz J. Nalepa, Marcin Szpyrka, Antoni Ligeza

AGH University of Science and Technology,
al. Mickiewicza 30, 30-059 Krakow, Poland
{kluza, gjn, mszpyrka, ligeza}@agh.edu.pl

Abstract BPMN is a visual notation for modeling business processes. Although there are many tools supporting it, they rarely provide formal verification of models. We propose a new approach to formal verification of BPMN models using the Alvis modeling language and the XTT2 knowledge representation. The structure of the BPMN model can be analyzed using translation to Alvis. Alvis models can be verified with dedicated tools, and their properties can be linked to the properties of the original BPMN model. On the other hand, selected BPMN elements can be verified using the XTT2 decision tables. Several BPMN elements can be translated to XTT2 and checked using the HeaRT rule engine with the HalVA verification and analysis tool. The paper constitutes an overview of the methods and concepts and presents preliminary results of our research.

1 Introduction

Business Process Model and Notation (BPMN) has recently emerged as a leading visual notation for modeling Business Processes. A BPMN model defines how the organization works by describing the ways in which operations are carried out to accomplish its intended goals. The progress in using the BPMN notation and the increasing complexity of the modeled processes make new advanced methods and tools needed.

BPMN provides a large collection of notation elements and allows for modeling various workflow structures, such as conditional operations, loops, event-triggered actions, splits and joins of sequence flow, etc. Moreover, it supports the hierarchical approach to design; thus, the process can be modeled on several abstraction levels.

The complexity of BPMN makes the formal verification of models a tough task. Although there are many tools supporting BPMN modeling, most of them do not provide any kind of formal model verification. In this paper, a new hybrid approach to formal verification of BPMN models is presented. It uses Alvis [1] and Extended Tabular Trees version 2 (XTT2) [2] methods. The considered approach is partially based on our previous research [3,4]. It extends and separates the verification process into two layers: the structure (or flow) layer and single components (mainly tasks) of the BPMN model.

*The paper is supported by the *BIMLOQ* Project funded from 2010–2012 resources for science as a research project.

This hierarchical separation provides verification of distinct properties on different abstraction levels. For the global (process structure) verification, the translation to Alvis modeling language is considered. The structure of the BPMN model can be analyzed thanks to its similarity to Alvis model, which is suitable for information systems modeling with subsystems working in parallel. For the local (model elements) verification, verification of single BPMN elements is considered. Such BPMN elements are mapped to the XTT2 knowledge representation, which can be verified using the HeaRT rule engine [5] as well as the HalVA verification and analysis tool [6].

This paper constitutes an overview of the methods and concepts, and presents preliminary results of the research aiming at formal verification of selected BPMN models using Alvis and XTT2 methods. We have limited the presentation to describing a simple yet illustrative case study of a student's project evaluation process.

The rest of the paper is organized as follows. Section 2 presents the BPMN notation and the selected case study. In Section 3 several works related to our research are presented. The BPMN model structure verification concept is introduced in Section 4, while the BPMN elements verification concept is given in Section 5. The evaluation of our approach is presented in Section 6. A short summary is given in the final section.

2 Business Process Modeling Notation

A Business Process can be defined as a collection of related tasks that produce a specific service or product for a particular customer. Business Process Model and Notation (BPMN) is a leading visual notation for modeling Business Processes. It uses a set of predefined graphical elements to depict a process and how it is performed. The current version of BPMN defines three models to cover various aspects of processes:

1. *Process Model* – describes the ways in which operations are carried out to accomplish the intended objectives of an organization. The process can be modeled on different abstraction levels: *public* (collaborative Business 2 Business Processes) or *private* (internal Business Processes).
2. *Choreography Model* – defines expected behavior between interacting business participants in the process.
3. *Collaboration Model* – can include Processes and/or Choreographies, and provides a Conversation view (which specifies the logical relation of message exchanges).

In our research, the internal Business Process Model is considered. There are four basic categories of elements used to model such processes: flow objects (*activities*, *gateways*, and *events*), connecting objects (*sequence flows*, *message flows*, and *associations*), swimlanes, and artifacts.

For the purpose of our research, only a subset of BPMN elements (flow objects and sequence flows) is considered. A task is a kind of activity, and a model defines the ways in which individual tasks are carried out. Gateways determine forking and merging of the sequence flow between tasks in a process, depending on some conditions. Events denote something that happens in the process. The icon in the event circle depicts the event type, e.g. envelope for *message event*, clock for *time event* (see Fig. 1).

Let us analyze an exemplary BPMN model of a simple student's project evaluation process. Fig. 1 depicts the evaluation process of a student's project for an Internet technologies course. However, the process can be used for any kind of a project depending on the rules which are applied to the particular tasks [7].

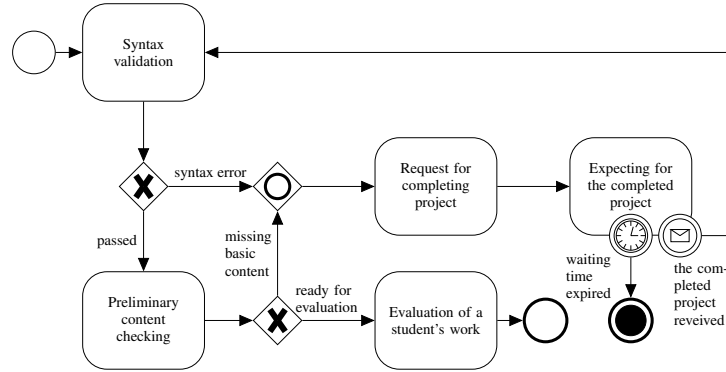


Figure 1. An example of the student's project evaluation process

In the considered example, the process is applied to the website project evaluation. At the beginning, the syntax is automatically checked. Every website code in XHTML needs to be well-formed in terms of the XML standard, and valid wrt the XHTML DTD.

If the project syntax is correct, preliminary content checking is performed. Then, if the project contains expected elementary XHTML tags, it can be evaluated and a grade can be given according to the rules defined by the teacher. On the other hand, if the project contains any syntax error or lacks some basic required content, it is requested to be completed. After receiving the completed project, the whole process starts from the syntax checking again. However, if the completed project is not received on time, the process is terminated (thus, the author of the project does not get a credit).

The example will be used for presentation of our research concerning a hierarchical approach to formal verification. We discuss the existing related works beforehand.

3 Related works

Most of the recent approaches to analysis of the BPMN models consider a restricted subset of BPMN elements in the model. They focus on checking of selected properties of the BPMN model through its transformation to a formal language.

In [8] Raedts et al. presented an approach transforming BPMN models to Petri nets, and these to the mCRL2 algebraic language. This allows for verification of the model using the mCRL2 toolset. Because the discovered problem have to be manually identified in the BPMN model, this can slow the result interpretation process.

Dijkman et al. in [9] proposed a similar approach. They presented a formal specification of the BPMN to Petri nets mapping, and thanks to this, they identified a number of deficiencies in the BPMN specification. The implementation of the approach transforms a BPMN model to a PNML file, which can be used in ProM tool in order to check

the model for absence of dead tasks and absence of incomplete process executions. One of the limitations of this approach is not supporting of OR-join gateways.

Similar research conducted by Ou-Yang and Lin [10] proposed a Petri-net-based approach to evaluate the feasibility of a BPMN model. This approach enables to reveal deadlocks and infinite loops. It consists in manually translating of the BPMN model to the Modified BPEL4WS representation, and then to Colored Petri-net XML (CPNXML). The resulted CPNXML representation can be verified using CPN Tools. The major limitations of this research are the limited assessment criteria, and lack of support of the multiple merge and split conditions in BPMN.

Another research direction concerns the translation of BPMN models to Yet Another Workflow Language (YAWL) [11], a modeling language for Business Processes. The BPMN2YAWL tool for such transformation was presented in [12]. Such model can be further checked using a YAWL-based verification tool. The recent research by Wynn et al. [13] presented the verification of YAWL models with advanced constructs, such as cancellations or OR-joins. The paper describes the mapping of a model to an extended Petri net to determine the model correctness, i.e. the following properties are verified: soundness, weak soundness, irreducible cancellation regions, and immutable OR-joins. Although in this research the process is modeled in YAWL, according to the authors, it can be applicable to BPMN as well. However, all of the YAWL approaches consider only BPMN to YAWL transformation. Thus, the errors revealed in the YAWL model can not be easily tracked in the BPMN model.

One of the recent paper in the field of BPMN model verification by Lam [14] proposed a transformation of the BPMN model to New Symbolic Model Verifier (NuSMV) language in order to do a model-checking analysis. The strength of this approach is that it has mathematical foundations and addresses the correctness issue of the transformation. However, this approach assumes a specification of Computation Tree Logic (CTL) formulas, which stipulate the required properties of the model to be checked. Therefore, it is not possible to check automatically a BPMN model of the process which does not have any properties specified using CTL.

The main drawback of these solutions is that it is difficult to map the resulting model back to the BPMN one. Although the tools reveal some errors in the model after translation, it is hard to find the corresponding place in the BPMN model and fix them.

4 BPMN model structure verification

In this section, we present the concept of BPMN model structure (global) verification. For such a verification, the model structure is translated to the Alvis modeling language.

4.1 Alvis modeling language

Alvis [1] combines the advantages of formal methods and practical modeling languages. The main differences between Alvis and more classical formal methods, especially process algebras, are: a user-friendly syntax and a visual modeling language (communication diagrams) for defining communication among agents. The main differences between Alvis and industry programming languages is a possibility of formal verification of Alvis models e.g. using model checking techniques.

The key concept of Alvis is an *agent*, which denotes any distinguished part of the system with a defined identity persisting in time. An Alvis model is a system of agents that usually run concurrently, communicate one with another, compete for shared resources etc. The dependencies among agents are described with three model layers: graphical, code and system one.

The *code layer* defines the behavior of individual agents. Each agent is described with a piece of source code. Agents can be either *active* or *passive*. *Active agents* perform some activities and each of them can be treated as a thread of control in a concurrent or distributed system. *Passive agents* do not perform any individual activity, but provide a mechanism for the mutual exclusion and data synchronization.

The *graphical layer* (communication diagram) defines connections (communication channels) among agents. A communication diagram is a hierarchical graph with nodes representing agents or parts of the model from the lower level. The diagrams allow for combining sets of agents into modules, represented as *hierarchical agents*. Active and hierarchical agents are drawn as rounded boxes while passive ones as rectangles. An agent can communicate with other agents through *ports*, drawn as circles placed at the edges of the corresponding agents. Communication channels are depicted as lines (or broken lines) with arrowheads showing the direction of communication.

From the users point of view, the *system layer* is predefined and only graphical and code layers have to be designed. The system layer is strictly connected with the system architecture and the chosen operating system. Alvis provides a few different system layers. The most universal one is denoted by α^0 and makes Alvis similar to other formal languages. The layer is based on the following assumptions: 1) each active agent has access to its own processor and performs its statements as soon as possible; 2) the scheduler function is called after each statement automatically; 3) in case of conflicts, agents priorities are taken under consideration (if two or more agents with the same highest priority compete for the same resources, the system works indeterministically).

4.2 BPMN to Alvis transformation

To verify the properties of the structure of the whole BPMN model, the model has to be transformed into Alvis model. The transformation procedure starts with preparing the *initial set of agents* which correspond to activities in the BPMN model. In the second stage, the initial set of agents is optimized. An Alvis agent can work like a buffer which collects a signal/value and then sends it to the next agent.

For each identified agent, we define its interface (ports) by considering the set of *surrounding edges* for a given BPMN activity i.e. sequence flows that go to or from the activity; however, if a sequence flow goes from the activity to a gateway, we consider the sequence flow going from the gateway. Each surrounding edge is transformed into a port of the corresponding agent. Moreover, for each port an identifier (a name) has to be added. To complete the agent definition, its behavior should be defined with additional code in the Haskell functional programming language.

Other agents in the Alvis model can be defined in very similar way. Moreover, in the considered example, a student is treated as a part of the system environment in the Alvis model. Thus, a project (its submission or resubmission), a grade and a timeout are sent via border ports, which have to be further specified. To represent the possibility of a

timeout, the agent definition should contain the Alvis statement with a time out branch. After receiving an error signal, the agent waits particular time for a revised project, and after this time a time_out signal is generated and the agent finishes its activity.

Although in Alvis a decision table activity can be represented as Haskell function, such an approach is beyond the scope of this paper. The approach considering full Alvis representation of the presented BPMN model was proposed in [4]. In the approach presented in this paper, Alvis is used only as a tool for global verification. For the purpose of local verification, the XTT2 method suits much better, and contrary to Alvis can provide a precise verification of single BPMN elements, such as gateways or tasks.

The last stage of the transformation procedure is to define communication channels in the Alvis model graphical layer, which in most cases consist in connecting pairs of ports. A more complex case is the transformation of the OR gateway, which requires to connect two pairs of ports. The complete communication diagram is shown in Fig. 2.

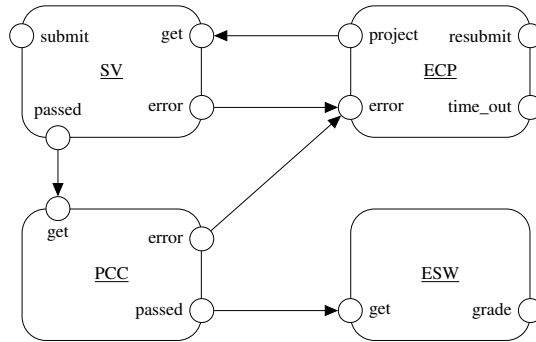


Figure 2. Alvis model – communication diagram

4.3 Verification of the model structure

The transformation of a BPMN model into an Alvis one is only a half-way to the formal verification of the model. Next, the Alvis model is transformed into a *labelled transition system* (LTS), used for formal verification. An LTS graph is an ordered graph with nodes denoting states of the considered system and edges denoting transitions among states. A state of a model is represented as a sequence of agents states. A state of an agent is four-tuple that consists of: agent mode (e.g. running, waiting), its program counter (point out the current step/statement), context information list (contains additional information e.g. the name of called procedure) and a tuple with parameters values.

There are two possible approaches to the formal verification of an LTS graph. If the graph is stored in the form of Haskell list, it is possible to add additional functions that inspect the list e.g. to find states with specified properties. On the other hand, such an LTS graph can be encoded using the *Binary Coded Graphs* (BCG) format and verified with the CADP toolbox. CADP offers a wide set of functionalities, ranging from step-by-step simulation to massively parallel model-checking. The verified properties can be divided into two groups usually called *safeness* and *liveness* ones. The former link with states properties while the latter link with an LTS graph paths' properties.

5 BPMN elements verification

Apart from the BPMN model structure analysis, checking several properties of single BPMN elements (local verification) is needed. Thus, our approach allows for verification of selected BPMN elements, which are mapped to the XTT2 knowledge representation. Thanks to the formal representation of XTT2, it is possible to verify several properties of these elements using the HeaRT rule engine [5] with the HalVA tool [6].

5.1 XTT2 rule representation

EXtended Tabular Trees v2 (XTT2) is a knowledge representation that incorporates an attributive table format. In this approach, similar rules are grouped in separated tables, and the system is split into a network of such tables representing the inference flow [15]. The XTT2 structure and rules can be modeled visually using the HQEd (HeKatE Qt Editor) rule editor. This table-based representation can be automatically transformed into HeKatE Meta Representation (HMR) which is suitable for direct execution by the HeKatE RunTime (HeaRT), a dedicated inference engine. HeaRT also provides a verification module – HeKatE Verification and Analysis (HalVA) [16]. The module implements a debugging mechanism that allows tracking system trajectory and logical verification of models. It is important to notice that formal verification is possible thanks to the formalized description in the ALSV(FD) logic of the XTT2 rules.

5.2 Gateways verification

Several problems related to selected BPMN elements may be considered. In the case of gateways, it should be checked if all the possible conditions are taken into account during the design. The proposed approach is as follows. A gateway BPMN element is translated to a table XTT2 knowledge representation – in this case it is represented as a single table. Diagram elements are translated to the XTT2 form according to appropriate logic functions. Thus, a BPMN element and its sequence flows are transformed to an XTT2 table filled with proper rules. Similar approach to the analysis of the BPMN elements and corresponding logic functions can be found in [3].

An exemplary decision table corresponding to the XOR gateway from the case study is presented in Fig. 3. The table consist of four columns. The first two, marked with (?), contain condition attributes, and the second two, marked with (->), contain the decision attributes. Each row contains a single rule that specifies the requirements for the flow.

The syntax of the resulting table can be validated in HQEd and then verified using HeaRT with HalVA. In the presented example, assuming that the *validation* attribute can take one of three values: *error*, *passed* or *warning*, the table can be verified against its completeness. It can be observed that the state in which the *validation* attribute takes the *warning* value is not included.

It is important to note that even if all model elements are validated, the whole model structure is still not grasped. Therefore, the verification of the model structure, presented in Section 4, is needed.

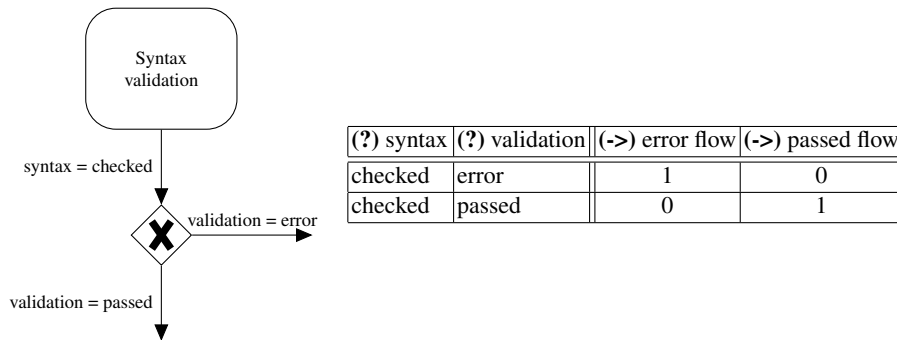


Figure 3. BPMN gateway verification

5.3 Tasks verification

Since BPMN does not specify the control logic of particular tasks, currently it has to be implemented manually. In the proposed approach it can be specified either using rules in the form of the XTT2 table or network, or as a HeaRT callback.

After specification of the task logic using the XTT2 decision tables, there is a possibility of their formal verification. Currently, HeaRT with the HalVA module allows for verification and analysis of the XTT2 table, i.e.: checking the inconsistency of a single rule, inconsistency of a pair of rules, incompleteness (lack of the ability to react for every admissible input values), subsumption of conditions and subsumption of a pair of rules, as well as identity and equivalence of rules [6].

An exemplary XTT2 decision table for the *Evaluation of a student work* task is shown in Table 1. The table can be used for evaluation a project, according to the specified rules. The output of the table is a grade for a project.

(?) implemented functionality	(?) quality	(->) grade
= basic	= low	:= satisfactory (D)
= any	= low	:= satisfactory (D)
= basic	= high	:= good (C)
= advanced	= fair	:= very good (B)
= advanced	= high	:= excellent (A)

Table 1. Decision table for student's project evaluation

In the presented table, it can be observed that there is no rule which can determine the *grade* when *implemented functionality* is *basic* and the *quality* of the project is *fair*. Thus, the verification would give the information about uncovered states (incompleteness), as well as it would inform that the second rule subsumes the first one. This is important when the system has to work correctly for any admissible input data and produce deterministic, consistent solutions.

6 Evaluation

The verification method presented in this paper is a new hybrid approach to the BPMN model verification and constitute a preliminary attempt to BPMN model execution.

From the structure point of view, the Alvis model resembles the original BPMN one. After a verification of the Alvis model, it is easy to link the model properties to the properties of original BPMN model. Contrary to the solutions presented in [9,10], our approach supports the OR-join gateway and the multiple merge and split.

From the single element point of view, the approach allows for using rule verification methods. Although this requires to specify gateway conditions using the ALSV(FD) logic and define the logic of tasks using the XTT2 representation, this can be a consistent method, complementary to the Business Processes. Moreover, the transformation from BPMN to XTT2 can be used for execution purposes in the future.

Therefore, the presented approach differs from the earlier attempts in addressing hierarchical verification of BPMN models. It allows for verification of both:

1. model structure (or flow) and
2. single elements (gateways and tasks) of the BPMN model.

In both presented cases, the BPMN elements are taken into account. However, thanks to the separation of layers, the approach provides the verification of distinct properties on different abstraction levels.

In the case of the BPMN model structure, the properties to verify can be divided into two groups: *safeness* and *liveness*. The former is related to states properties e.g. a project with correct content cannot be treated as a defective one. The latter concerns properties of LTS graph paths e.g. if the time out signal has not been generated and a project with correct content has been provide, the system must provide a suitable grade.

When it comes to the BPMN elements, there are many properties which can be verified, such as: lack of *redundancy*, *consistency*, *minimal representation*, or *completeness*.

Although the approach concerns only a small subset of BPMN, extending of this subset is expected in the future. Dedicated tools enabling automatic translation of the BPMN model to Alvis and XTT2 representations are planned to be implemented. Moreover, the formal definition of transformation rules will be developed.

7 Conclusion

The paper presents preliminary results of the research concerning verification of BPMN models. The original contribution is the proposal of a hybrid and hierarchical approach to formal verification of selected BPMN models. We propose an approach which uses the Alvis modeling language for the global verification of the model structure and the XTT2 knowledge representation for the local verification i.e. verification of single BPMN elements in the model. The presentation of the approach has been limited to the presentation of a simple, yet illustrative, case study of a student's project evaluation process. The considered example contains only a few activities, gateways, and events. However, it is possible to use the presented approach for more complex models.

References

1. Szpyrka, M., Matyasik, P., Mrówka, R.: Alvis – modelling language for concurrent systems. In Bouvry, P., Gonzalez-Velez, H., Kołodziej, J., eds.: *Intelligent Decision Systems in Large-Scale Distributed Environments*. Studies in Computational Intelligence. Springer-Verlag (2011) (in press).
2. Nalepa, G.J., Ligeza, A.: HeKatE methodology, hybrid engineering of intelligent systems. *International Journal of Applied Mathematics and Computer Science* **20**(1) (2010) 35–53
3. Kluza, K., Maślanka, T., Nalepa, G.J., Ligeza, A.: Representing BPMN diagrams with XTT2-based business rules proposal. In Brazier, F.M., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C., eds.: *Intelligent Distributed Computing V*. Studies in Computational Intelligence. Springer-Verlag (2011) in press.
4. Szpyrka, M., Nalepa, G.J., Ligeza, A., Kluza, K.: Proposal of formal verification of selected bpmn models with alvis modeling language. In Brazier, F.M., Nieuwenhuis, K., Pavlin, G., Warnier, M., Badica, C., eds.: *Intelligent Distributed Computing V*. Studies in Computational Intelligence. Springer-Verlag (2011) in press.
5. Nalepa, G.J.: Architecture of the HeaRT hybrid rule engine. In Rutkowski, L., [et al.], eds.: *Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II*. Volume 6114 of *Lecture Notes in Artificial Intelligence*. Springer (2010) 598–605
6. Nalepa, G., Bobek, S., Ligeza, A., Kaczor, K.: Halva - rule analysis framework for xtt2 rules. In Bassiliades, N., Governatori, G., Paschke, A., eds.: *Rule-Based Reasoning, Programming, and Applications*. Volume 6826 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg (2011) 337–344
7. Nalepa, G.J., Kluza, K., Ernst, S.: Modeling and analysis of business processes with business rules. In Beckmann, J., ed.: *Business Process Modeling: Software Engineering, Analysis and Applications*. Business Issues, Competition and Entrepreneurship. Nova Publishers (2011)
8. Raedts, I., Petković, M., Usenko, Y.S., van der Werf, J.M., Groote, J.F., Somers, L.: Transformation of BPMN models for Behaviour Analysis. In Augusto, J.C., Barjis, J., Nitsche, U.U., eds.: *MSVVEIS, INSTICC press* (2007) 126–137
9. Dijkman, R.M., Dumas, M., Ouyang, C.: Formal semantics and automated analysis of BPMN process models. preprint 7115. Technical report, Queensland University of Technology, Brisbane, Australia (2007)
10. Ou-Yang, C., Lin, Y.D.: BPMN-based business process model feasibility analysis: a petri net approach. *International Journal of Production Research* **46**(14) (2008) 3763–3781
11. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet another workflow language. *Information Systems* **30**(4) (2005) 245–275
12. Decker, G., Dijkman, R., Dumas, M., García-Bañuelos, L.: Transforming BPMN diagrams into YAWL Nets. In Dumas, M., Reichert, M., Shan, M.C., eds.: *Business Process Management*. Volume 5240 of *Lecture Notes in Computer Science*. Springer (2008) 386–389
13. Wynn, M., Verbeek, H., Aalst, W.v.d., Hofstede, A.t., Edmond, D.: Business process verification – finally a reality! *Business Process Management Journal* **1**(15) (2009) 74–92
14. Lam, V.S.W.: Formal analysis of BPMN models: a NuSMV-based approach. *International Journal of Software Engineering and Knowledge Engineering* **20**(7) (2010) 987–1023
15. Nalepa, G.J., Ligeza, A., Kaczor, K., Furmańska, W.T.: HeKatE rule runtime and design framework. In Giurca, A., Nalepa, G.J., Wagner, G., eds.: *Proceedings of the 3rd East European Workshop on Rule-Based Applications (RuleApps 2009)* Cottbus, Germany, September 21, 2009, Cottbus, Germany (2009) 21–30
16. Ligeza, A., Nalepa, G.J.: Rules verification and validation. In Giurca, A., Gasevic, D., Taveter, K., eds.: *Handbook of Research on Emerging Rule-Based Languages and Technologies: Open Solutions and Approaches*. IGI Global, Hershey, New York (2009) 273–301

Architecture for the Use of Synergies between Knowledge Engineering and Requirements Engineering - Extended Abstract *

José del Sagrado, Isabel M. del Águila, and Francisco J. Orellana

Dpt. Languages and Computation,
Ctra Sacramento s/n, 04120 University of Almería, Spain
{jsagrado, imaguila, fjorella}@ual.es

Expert knowledge is involved in every software development project since developers must face numerous decision tasks during requirements management, analysis, design, and implementation stages. Therefore, if expert knowledge could be properly modelled and incorporated in the different processes of software development as well as in the CASE tools that support these processes, that would mean a great advantage for any software development.

In software development, requirements stage is considered a good application domain for Artificial Intelligence (AI) techniques because of requirements nature. Software requirements express and establish the needs and constraints that contribute to the solution of a real world problem [7]. However, requirements tend to be imprecise, incomplete and ambiguous[3] and has a big impact in whole development stages [5, 15, 1, 2]. Therefore, the use of AI techniques in order to improve requirements stage will favorably affect the whole software life cycle, but we need a seamless integration of Requirement Engineering (RE) and AI techniques to exploit the benefits of collaboration between these two knowledge areas [10].

Besides, the biggest breakthrough in requirement management is when you stop thinking of documents and start thinking about information. Here, is where CARE (Computer-Aided Engineering Requirement) tools help us in order to be able to handle all of this information. InSCo Requisite is an academic web CARE tool, developed by DKSE group at the University of Almería, which aids during the requirement development stage [11].

This work presents the architecture for the seamless integration of a CARE tool to manage requirements (i.e. InSCo Requisite) with some AI techniques (i.e. Bayesian networks [12, 6] and metaheuristics). Specifically, a Bayesian network, called Requisites [13], is used in the requirement validation task in order to validate the Software Requirements Specification (SRS) of a software development project, they has been successfully applied in SE, [9, 4, 8, 13]. Metaheuristic techniques (Simulated Annealing, Genetic Algorithms and Ant Colony Systems) are used in the problem of selecting the subset of requirements among a whole

* This is a long abstract of the paper published in Lozano, J.A., Gámez, J.A., Moreno, J.A. (eds.) LNAI Series, Current Topics in Artificial Intelligence. 14th Conference of the Spanish Association for Artificial Intelligence, CAEPIA 2011, La Laguna, Spain, November 8-11, 2011, Selected Papers.

set of candidate requirements proposed by a group of stakeholders, that will be included in the development of a final software product [14].

The RE workflow depicted in Figure 1 shows an organization of the tasks that must be done in a software development project during RE stage. Requirements are elicited or gathered from users, next they are specified in a document or its electronic equivalent, known as Software Requirements Specification (SRS). CARE tools provide environments that make use of databases, allowing an effective management of the requirements of any software project. Requirements validation checks whether the elicited and specified requirements present inconsistencies; if the information is incomplete or if there are ambiguities in the system definition. Requisite Bayesian network provide developers an aid, under the form of a probabilistic advice (i.e. an estimation of the degree of revision for the SRS), helping them at the time of making a decision about the stability of the current requirements specification. Finally, requirements selection task has as main objective to choose, from all the requirements defined in the specification, the subset of requirements that will be implemented.

Bayesian networks and metaheuristic techniques have demonstrated to obtain interesting results through different tests data [13, 14]. However, it is difficult to put them in practice in real software projects. We strongly believe that having these AI techniques available in a CARE tool would be considerably helpful for any development team, making them more accessible even for non-expert people. However, IA techniques and the CARE tools have been developed independently of each other. Therefore, it is necessary to define a communication interface between them preserving the independent evolution of both areas and achieving a synergic benefic effect between them. This seamless synergic architecture is shown in Figure 1. The architectural pattern distinguish between three logically separated layers (see Fig. 1) : the presentation (i.e. interface layer), the application processing (i.e. service layer), and the data management (i.e. data layer).

The interface is a web environment accessed from a web browser. Data layer is in charge of storing and managing the electronic representation of SRS handled by InSCo Requisite tool and the knowledge base that contains the Bayesian network Requisites. Service layer is composed by the CARE tool (i.e. InSCo Requisite), the AI techniques used to address requirements validation (i.e. Bayesian network Requisites) and requirements selection (i.e. metaheuristics algorithms) tasks. Communication interface connect CARE and knowledge-based tools passing the required information needed for the execution of the appropriated processes. Thus, requirement validation receives metrics on the SRS and returns an estimation of the degree of revision for SRS; requirement selection receives resources effort bound and specific measures on individual requirements and set the set of requirements in order to be implemented. All of these communication processes are performed through XML files.

The purpose of this work is to define a three-layer architecture which: a) allows the seamless collaboration between RE tasks and some AI techniques (Bayesian networks, simulated annealing, genetic algorithms and ant colony sys-

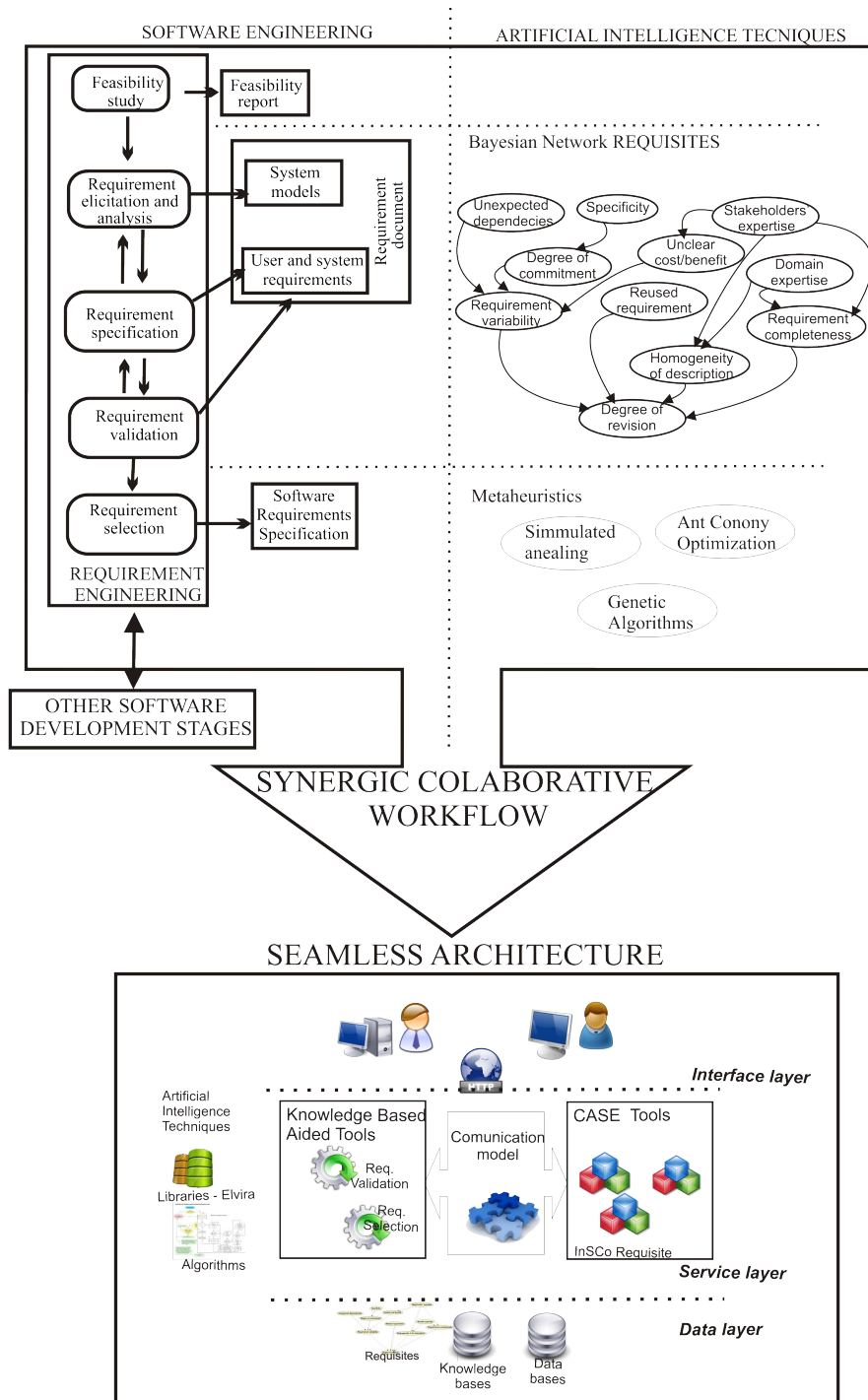


Fig. 1. Seamless synergic architecture.

tems) in order to perform a software development project; b) facilitates their parallel and independent evolution.

Acknowledgments. This work was supported by the Spanish Ministry of Science and Innovation under project TIN2010-20900-C04-02 and by the Junta of Andalucía under project TEP-06174.

References

1. Standish Group: Chaos Report. Technical report, Standish Group International (1994)
2. Johnson, J.: CHAOS chronicles v3.0. Technical report, Standish Group International (2003)
3. Cheng, B.H., Atlee, J.M.: Research directions in requirements engineering. In: Future of Software Engineering, FOSE'07, pp. 285-303. Institute of Electrical and Electronics Engineers, Minneapolis, Minnesota (2007)
4. Fenton, N., Neil, M., Marsh, W., Hearty, P., Marquez, D., Krause P., Mishra, R.: Predicting software defects in varying development lifecycles using Bayesian nets, *Information and Software Technology* 49(1) 32-43 (2007)
5. Glass A.R.L.: Facts and Fallacies of Software Engineering. Pearson Education, Inc., Boston, MA (2002)
6. Jensen F.V.: Bayesian Networks and decision graphs. Springer-Verlag, New York (2001)
7. Kotonya, G., Sommerville, I.: Requirements Engineering: Processes and Techniques. Wiley (1998)
8. Lauria, E.J., Duchessi, P.J., A Bayesian Belief Network for IT implementation decision support. *Decision Support Systems* 42(3), 1573-1588 (2006)
9. de Melo A.C. , Sanchez, A.J.: Software maintenance project delays prediction using Bayesian Networks. *Expert Systems with Applications* 34(2), 908-919 (2008)
10. Meziane, F., Vadera, S. (eds.): Artificial intelligence applications for improved software engineering development: new prospects. IGI Global, Hershey, New York (2010)
11. Orellana, F.J., Cañadas, J., del Águila, I.M., Túnez, S.: INSCO requisite - a Web-Based RM-Tool to support hybrid software development. In: International Conference of Enterprise Information System ICEIS (3-1), pp. 326-329. Barcelona, Spain (2008)
12. Pearl, J.: Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufman, San Mateo, CA (1988)
13. del Sagrado, J. , del Águila, I.M.: A Bayesian Network for Predicting the Need for a Requirements Review. In: Meziane, F., Vadera, S. (eds.): Artificial intelligence applications for improved software engineering development: new prospects, pp. 106-128. IGI Global, Hershey, New York (2010)
14. del Sagrado, J., del Águila, I. M., Orellana, F. J.: Requirement selection: Knowledge based optimization techniques for solving the next release problem. In: 6th Workshop on Knowledge Engineering and Software Engineering (KESE 2010), pp. 40-51. CEUR-WS, Karlsruhe, German (2010)
15. Sommerville I.: Software Engineering. Addison-Wesley Longman Publishing Co., Inc., Boston (2006)

Mashups: Behavior in Context(s)

Emilian Pascalau

Hasso Plattner Institute at University of Potsdam,
Germany,
emilian.pascalau@hpi.uni-potsdam.de

Abstract. The World Wide Web (WWW) has left behind the dot-com bubble and changed into something new. The move to the Internet as a platform and the shift from transaction-based Web pages to interaction-based ones opened the way to a whole new environment. Future Internet is a generative environment that fosters innovation, through an advance of technologies and a shift in how people perceive, use and interact with it. Nowadays the abilities to create new information have far exceeded the abilities to manage it. There exist a huge amount of data and potential that is still unused and undiscovered.

Mashups are a new paradigm emerged from Web 2.0 that tries to empower users with some sort of freedom to tackle this huge amount of potential provided nowadays by the web. Current approaches however are still restrictive in many ways e.g. are data oriented, and platform dependent. Hence this paper introduces a new perspective on mashups. Here a mashup is seen as a plan that a user and an engine need to follow in order to achieve a desired goal. As such a mashup comprises contextual information and the necessary behavior related to the context(s) described, in order to fulfill desired goal. Subsequently a mashup is defined here as behavior in context(s).

1 Introduction

Web 2.0 [12] is *no longer a bleeding edge but rather a leading edge now* [15] and has become integral part of life and business. Participation is one aspect which pushes forward Web 2.0 [3]. In the last five years Web 2.0 technologies (i.e. social networking sites, blogs, wikis) have spread widely among consumers. Sites such as Facebook attract more than 100 million visitors a month [3]. There is a shift from processes towards users. Users want their problems and their requirements to be taken into account; they want to be part of the conversation. Continuously changing business models do not fit anymore the old and stiff approaches. Processes must be in accordance with the reality, and reality means people. It means that processes and system behavior have to be in accordance with what users require and with their needs. This issue is also underlined by process mining [16] approaches that look at event logs and see that the processes that actually get executed are different compared to the original blueprints. Companies need to change to what customers/users actually do.

Harnessing collective intelligence, wisdom of the crowds, easy consumption, web as innovation platform, context are requirements that need to be tackled to allow people to use their imagination without too much restriction in order to fulfill by themselves their goals. As argued in [13] the Web is more than just data, is about knowledge, context, behavior and most important is about people.

This paper proposes a new definition for the mashup concept from a user's perspective. Thus a mashup is behavior in context(s). This particular perspective is a high level one addressing mashups at a conceptual level opposed to current approaches that are mostly application and technology oriented (see for instance Yahoo Widgets¹). The framework discussed here allows users to model a mashup as a map containing context(s) and behavior description required to fulfill a specific goal. In consequence this approach implies business rules, business processes, business concepts and vocabularies that describe the businesses and users' goal themselves rather than a possible IT system that might support it. The framework is formalized using the Unified Modeling Language (UML).

The reason for such a framework is manifold. To name just a few: (1) users are able to define their own applications in order to fulfill their needs; (2) because the framework unifies several paradigms (behavior, context etc) reasoning can be performed in an unified over all these; (3) models described can be extended, modified and maintained in an unified way as well; (4) companies can learn their customers' needs as these mashups expose behavior as well as the context(s) in which exposed behavior is performed; (5) these mashups can act very easily as prototypes for possible future major implementations; (6) statistics can be provided about the usage of mashups and about the system(s) involved in relationship with social networking platforms and so forth.

2 Conferences Calendar Example

The Conferences Calendar example has been first introduced in [13]. Such a calendar is user specific since for example some users might be interested in web related conferences, others in semantic web, others in rules or business processes conferences. Contextual information is mashed up to fulfill a user goal; hence specific information about conferences is stored in a calendar context. At least two services are required: one that deals with conferences and one that offers a calendar. From a technological point of view these services might not be compatible with each other e.g. might not use the same definition language.

For scientists in the filed of IT the DbWorld Mailing List is the well known place where they can search for an IT conference. A series of information are provided here, but most important are the subject, deadline and the web page of the event published. From a technical perspective DBWorld does not provide an API to allow programmatically access and interrogation of the service. In consequence with respect to current mashups approaches this service is useless. On the other hand Google Calendar is one of the most known Google Apps

¹ <http://manual.widgets.yahoo.com>

services and the service that provides the calendar context for the use case. The information of interest for a calendar is the title of the event, the date and description of an event. This information is found also in a Google Calendar. Opposed to the DbWorld service, Google provides for this service beside the regular web page representation also an API to access the contents.

The usual way to achieve this goal, of having conferences stored in Google Calendar by their deadline is manually: (1) the user is required to maintain two open tabs in the browser; (2) even though there might be several entries that comply with a search term, the user must deal with the events one by one as DBWorld does not provide built in search functionality; (3) the user has to move between the open tabs several times, in order to store only one event in the calendar, since just one piece of information can be copied and pasted at a time (e.g. the title of the event). An important aspect is that both services as well as users interact with each other.

3 The Framework

As argued the framework discussed here it is defined from a user perspective and is formalized using the de facto standard modeling language UML.

Next subsections discuss the main concepts of the framework required to define the *mashup* concept.

3.1 Concept

Ackoff [1] defines an abstract system as one all of whose elements are concepts. Because the framework has to deal with a high degree of generality it has at the core the abstract notion of *concept*. A concept is a cognitive unit of meaning - an abstract idea or a mental symbol.

Languages, number systems are examples of abstract systems. Numbers are concepts but the symbols that represent them, numerals are physical things [1]. Humans deal both with conceptualizations as well as with physical things. However the reasoning process involves only conceptualizations.

Although many of the ontological approaches (see for instance OWL [6]) use as the upper level entity the *thing* notion for the framework discussed here the notion of *concept* is at the top.

The OMG specification for Semantics of Business Vocabulary and Business Rules Specification (SBVR) [10] uses as top entity the *concept* notion. Definition 1 is the SBVR definition for the concept notion.

Definition 1. *Concept.* *Unit of knowledge created by a unique combination of characteristics.*

There are two ways to recognize entities. Basically in software engineering when dealing with typed languages, entities are recognized by their types (class name). The other way around is based on a set of characteristics. Take for instance a *car*. Stating the concept's name *car* someone will be able to tell you

the characteristics of a car, that it has for wheels, that it has an engine etc. Nevertheless stating the characteristics of the concept that it has 4 wheels and an engine, the answer will be a *car*. The reasoning architecture [13] that uses this framework tackles both approaches.

3.2 Context

The notion of context is of interest in cognitive psychology, in linguistics and computer sciences. In the field of computer sciences notions of context have appeared in several areas such as artificial intelligence, machine learning, data bases and software development. In some of these areas the notion of context appears in the form of views, aspects, information for concept classification, means to partition knowledge in manageable sets, or as an abstraction mechanism to partition information into possibly overlapping parts [2].

Dey in [5] defines context as *any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves*. Similarly for Coutaz et al. [4] the context is a *structured and unified view of the world in which the system operates*.

Context is under permanent change, is episodic, personal and hence subjective interpretations and experiences of the communicative context [17],[5]. Analyti et al. discuss in [2] a general framework to harness the notion of context in conceptual modeling. A full mathematical apparatus has been defined to tackle issues such as containment and relationships between contexts. According to them "context in an information base can be seen as a higher-order conceptual entity that groups together other conceptual entities on which we want to focus" [2]. More precisely a context is a set of objects within which each object is associated with a set of names.

For the framework discussed here the notion of context adheres to the mathematical apparatus defined in [2], however here a context is a set of concepts not objects. Nonetheless our definition is compliant with Analyti et al. definition.

Basically a context is a set of concepts (concepts according to Definition 1).

Definition 2. Context. *A context consists of a context identifier and a set of concepts identifiers.*

Recalling the simple mechanism that has been discussed in subsection 3.1, a context is identified by recursively identifying all the constituent concepts.

The notion of context supports a series of features as they have been defined in [2]: (1) concept sharing or overlapping contexts; (2) context-dependent concept names; (3) context dependent references; (4) context sharing; (5) context-dependent reachability; (6) synonyms, homonyms,onyms.

Beside these features the notion of context is enhanced also with attribution, generalization and classification.

3.3 Behavior

For this particular approach behavior comprises rules and processes. It is described by users in relationship with related context(s).

When dealing with human like behavior a single system (mind) produces all aspects of behavior [8]. *It is one mind that minds them all* [8]. Even if such a system has parts, modules, components or whatever they mesh together to produce behavior.

The mind is the control unit that guides the behaving organism in its complex interactions with the dynamic real world [8]. Both the behaving organism as well as the environment behaves through time with a series of interactions between them. [8] continues by stating that these transactions or interactions are embedded in a sequence such that each becomes part of the context within which further action follow.

Newell underlines a set of requirements that behavior must comply with [8]: (1) it has to be flexible as a function of the environment; (2) flexible in such a sense that it can deal with goals; (3) real time; (4) according with the context.

Behavior of an entity is the set of events, actions and messages that that entity produces. Behavior is conditioned by the context and it is expressed either as rules or as processes. Hence an event is any observable occurrence of a phenomena. An action as stated in [7] is represented with the keyword *do* and is represented as *function from a time and an action, to the time at the end of the action*. Acokff defines behavior in [1] in terms of system as a system change which initiates other events implying that behavior consists of events *whose consequences are of interest* [1].

According to the PRR [9] specification a production rule is a statement that specifies the execution of one or more actions in the case that its conditions are satisfied. An Event Condition Action (ECA) rule is a production rule triggered by an event. Thus the form of an ECA rule is: `on [events] if [conditions] then [action-list]`.

While Weske argues in [18] that a *"business process consists of a set of activities that are performed in coordination in an organizational and technical environment"*, Ackoff on the other hand defines a process as a sequence of behavior that constitutes a system and has a goal producing function [1].

Hence behavior is goal oriented, is context(s) related and is expressed as rules and processes.

3.4 Mashup

This section unifies and puts together previously discussed aspects in order to define the mashup concept. In consequence a mashup is a map which describes the context(s) and related behavior that a user needs to do in order to achieve a desired goal. Such a mashup is defined from a user perspective. Coutaz et al.' [4] view of *context-as-process* is related to the idea I discuss here. However their approach is not from a user perspective but rather from an IT system perspective. Nonetheless at least two issues are addressed by having context

related to behavior. First *context-as-process* view allows for greater flexibility than *context-as-state* as utility and usability are derived from information exchange and interaction [4], [14]. Secondly there is no mismatch risk between system's interaction model and the mental model that a user might have about the system[4]. This approach uses actually as interaction model for the system the one that a user defines as a mashup. Moreover as discussed in [14] context provides meaning to processes. For example one could deal with a sell/buy process, a very generic one. But whenever contextual information is added, the *meaning* of a process could be totally different, as there is a big difference between selling tomatoes and selling e.g. chemical products. To support even more this idea SBVR specification [10] states that a body of shared meaning that a community has is represented in concepts, fact types (relationships between concepts) and business rules (constraints on concepts and fact types).

Definition 3 (mashup). *Mashup.* A mashup is a set of contexts and behavior. Behavior consists of rules and processes.

Figures 1, 2, 3, 4 formalize the framework. These models comply with the definitions previously discussed.

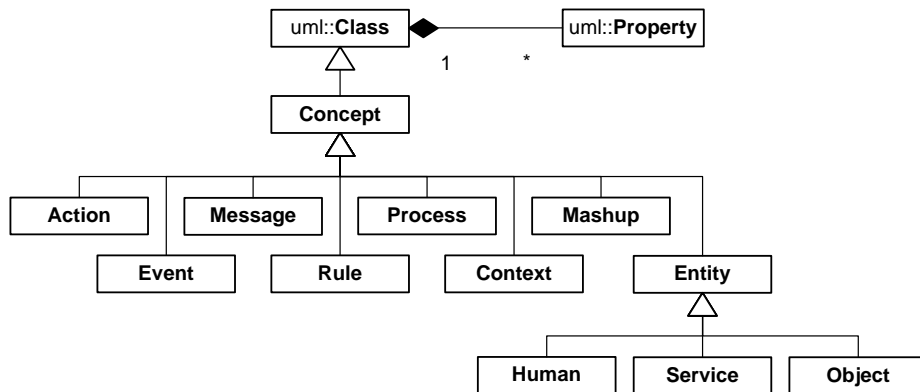


Fig. 1. Concepts

Based on Definition 1 a concept is a unit of knowledge created by a unique combination of characteristics. Thus as depicted in Figure 1 every element of the framework is a concept. In addition although not visually represented a *Concept* is also a *Concept*. In this way the reasoning process can involve any of the concepts defined in a unified way.

Figure 2 depicts the general framework. Hence the *Mashup* concept contains 1 or more *Contexts*. Further a *Mashup* can contain *Processes*, *Rules* or a combination of those two. A *Context* is basically a collection of *Concepts*. In addition a *Context* could have subcontexts. A *Context* refers to an *Entity*.

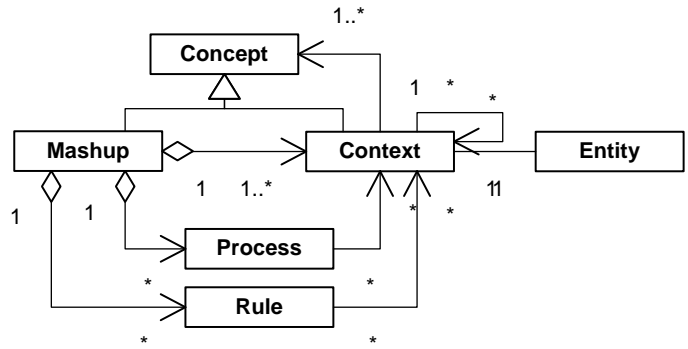


Fig. 2. Mashup Concept

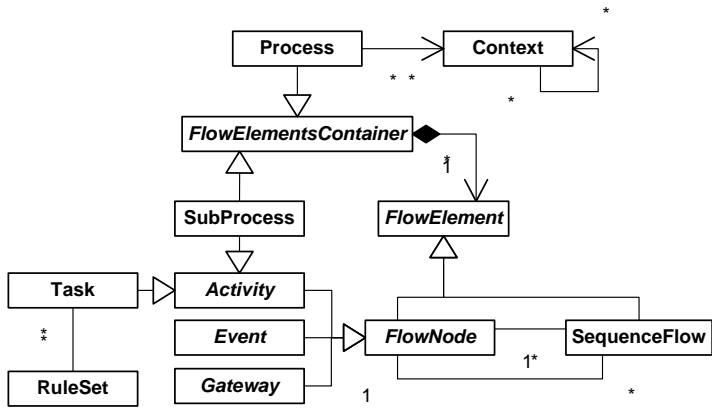


Fig. 3. Business Process, based on BPMN 2.0 specification [11]

The Process Concept is further expanded in Figure 3. The definition is based on the BPMN 2.0 specification. Thus a process is a `FlowContainer` and contains `FlowElements` and `SequenceFlows`. Furthermore a `FlowElement` is either an `Activity`, a `Gateway` or an `Event`. An `Activity` is subclassed by a `SubProcess`, meaning that a `Process` might have subprocesses, and by a `Task`. A `Task` is an atomic `Activity`. However this model introduces the following relationships, which were not previously contained by the BPMN 2.0 specification: the execution of a `Task` can mean the execution of `RuleSets`; `Processes` are related to `Contexts`. This particular relationship can provide as discussed in [14] meaning to processes.

The model depicted in Figure 4 is compliant with the OMG PRR specification [9] and is the basic model for a Rule. A Rule similar to a `Process` is related to a `Context`. It can be triggered by an `Event`, in the case of an `Event Condition Action (ECA) Rule`. It can be conditioned by a set of conditions. Conditions concern `Concepts`. Actions are the result of rule execution. With respect to execution beside the regular rule conflict resolution mechanism the engine using

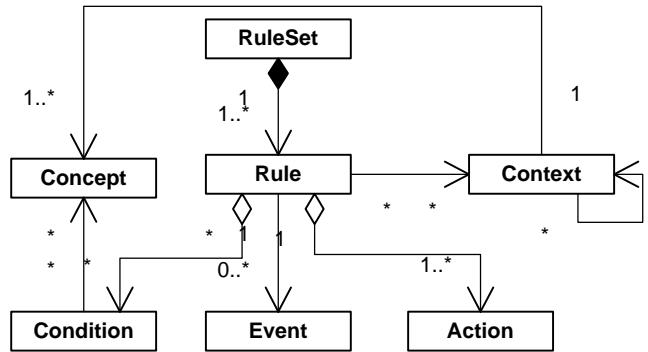


Fig. 4. Rules

this framework uses processes to order the execution of actions in relationship with events.

4 Using the Framework

Recall the example discussed in Section 2. Several contexts are involved in this particular mashup: the DBWorld context, the Google Calendar context and the calendconf context. Figure 5 depicts all the involved contexts mashed together. According to Definition 2 a context contains a set of concepts. In addition as argued in Section 3.2 identifying a context means identifying all the constituent concepts. Let's take for instance the Google Calendar context. This one refers to the Google Calendar Entity. This entity is uniquely identified by its URL. The context contains a **Create Event** button. While this concept in relationship with the entity is enough for one user, someone else could use a different set of concepts to identify the same context. Furthermore **Create Event** button is identified by a set of characteristics. The most evident one is the name: **Create Event**. Figure 6 depicts an excerpt of the framework instantiation.

I was arguing that behavior is in a strong relationship with the context. The most simple example: to be able to create a calendar event in Google Calendar a user needs to click on the **Create Event** button. A more complex one (see Figure 7) is the process of searching for a particular conference in DbWorld. From DbWorld the subject, deadline and web page are the concepts of interest. With respect to behavior in this context, one user could be interested both in the subject and deadline when searching for a conference. On the other hand another one could be interested only in the subject.

While the framework allows reasoning over all the constitutes elements similar to human cognition and as such empowering users with the ability to define behavior in fine details an user friendly modeling platform for the non technical users is desired. Widgets based, pipes based platforms have proven to be easy to use. Similar to those approaches a visual modeling platform for the framework

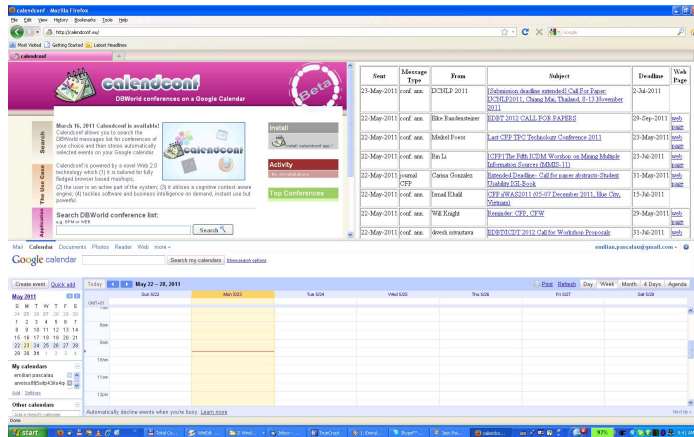


Fig. 5. Calendconf Mashup

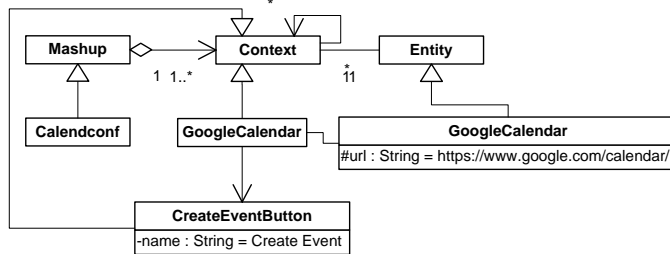


Fig. 6. Framework Instantiation - an excerpt

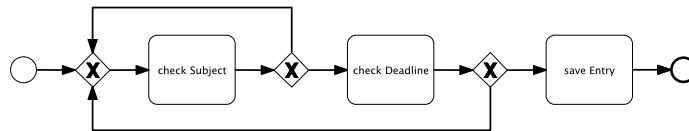


Fig. 7. Search for a Conference Process

discussed here is under development. Currently mashups are defined declaratively using JSON notation. Nonetheless the running version of the example discussed here can be accessed at <http://calendconf.eu>.

5 Conclusions

This paper discussed a new perspective for the mashup concept. While the Web 2.0 mashup paradigm has been mostly currently addressed from a technical perspective and strongly application oriented, the framework formalized here concerns a high level perspective and defines a mashup as behavior in context(s).

Further improvements of the framework concerns reuse of mashups with an emphasis on inheritance. As argued in [14] this is not a straight forward process as here behavior is defined using UML, hence as static constructs. In consequence UML class inheritance can not be used as it is but special types of inheritance mechanisms are required.

References

1. R.L. Ackoff. Towards a System of System Concepts. *Management Science*, 17(11), 1971.
2. Anastasia Analyti, Manos Theodorakis, Nicolas Spyrtatos, and Panos Constantopoulos. Contextualization as an independent abstraction mechanism for conceptual modeling. *Information Systems*, 32(1):24–60, 2007.
3. Michael Chul, Andy Miller, and Roger P. Roberts. Six Ways to make Web 2.0 work. *Business Technology, The McKinsey Quarterly*, pages 1–6, February 2009.
4. Joelle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM*, 48(3):49–53, 2005.
5. Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. Technical Report GIT-GVU-99-22, Georgia Institute of Technology, 1999.
6. W3C OWL Working Group. OWL 2 Web Ontology Language. <http://www.w3.org/TR/owl2-overview/>, 2009.
7. Patrick J. Hayes. A Catalog of Temporal Theories. Technical Report UIUC-BI-AI-96-01, University of Illinois, 1996.
8. Allen Newell. *Unified Theories of Cognition*. Harvard University Press, 1994.
9. OMG. Production Rule Representation (PRR), Beta 1. Technical report, OMG, November 2007.
10. OMG. Semantics of Business Vocabulary and Business Rules Specification. <http://www.omg.org/spec/SBVR/>, January 2008.
11. OMG. Business Process Model and Notation (BPMN). FTF Beta 1 for Version 2.0. <http://www.omg.org/spec/BPMN/2.0>, August 2009.
12. T. O’Reilly. What Is Web 2.0. Design Patterns and Business Models for the Next Generation of Software. *Communications and Strategies*, 1st quarter(65):17, 2007.
13. Emilian Pascalau. Towards TomTom like systems for the web: a novel architecture for browser-based mashups. In *Proceedings of the 2nd International Workshop on Business intelligence and the WEB (BEWEB11)*, pages 44–47. ACM New York, NY, USA, 2011.
14. Emilian Pascalau and Clemens Rath. Managing Business Process Variants at eBay. In Jan Mendling and Mathias Weske, editors, *Proceedings of the 2nd International Workshop on BPMN, BPMN2010*. Springer, 2010.
15. The Economist Intelligence Unit. Serious business. Web 2.0 goes corporate. *The Economist*, pages 1–20, 2007.
16. W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow Mining: A Survey of Issues and Approaches. *Data and Knowledge Engineering*, 47(2):237–267, 2003. <http://is.tm.tue.nl/staff/wvdaalst/publications/p206.pdf>.
17. Teun A van Dijk. Cognitive Context Models and Discourse. *Proceedings and Debates of the 102d Congress*, 137(84):189–226, June 1991.
18. M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag Berlin Heidelberg, 2007.

Model-Driven Rich User Interface Generation from Ontologies for Data-Intensive Web Applications

Joaquín Cañadas¹, José Palma² and Samuel Túnez¹

¹ Dept. of Languages and Computation. University of Almeria. Agrifood Campus of International Excellence, ceiA3. Spain

`jjcanada@ual.es`, `stunez@ual.es`

² Dept. of Information and Communications Engineering. University of Murcia. Spain
`jtpalma@um.es`

Abstract. Building data-intensive Web applications is a complex task widely explored during the last decade. Many approaches have been proposed, mainly based on conceptual models as well as on domain ontologies and knowledge models. This work describes a method for rich user interface development for data-intensive Web applications based on OWL2 ontologies, which applies model-driven engineering to derive a user interface from the domain ontology, incorporating modern rich components for Web-based interfaces. A tool supporting the ideas presented in this paper has been developed.

Keywords: user interfaces, model-driven engineering, ontologies

1 Introduction

Data-intensive Web applications are Web-based information systems for accessing and maintaining large amounts of structured data, typically stored in database management systems [4]. Its development involves the definition of data models representing the information structure of the problem domain, as well as the design of user interfaces (UIs) to enable end-users to properly interact with the system when managing the collection of data, e.g. UIs for data presentation, data acquisition and data querying.

In general, the usage of data models for domain specification suffers of poor expressivity. To address this problem, ontologies are typically applied to domain modeling in which a conceptualization of a particular domain is given. Ontology formalisms such as OWL2 (Web Ontology Language 2) [22] are the backbone of Semantic Web and are growing in importance in software development [9]. Ontologies can describe the relevant concepts, relations and properties of an application domain adding assertions and constraints, increasing the amount of knowledge that can be represented by data models.

In this work we describe a model-driven method for deriving rich Web-based UIs for data management from domain models based on domain ontologies.

We apply Model-Driven Architecture (MDA) [15], or using the general concept, Model-Driven Engineering (MDE) [20], as the software development approach in which models are used as first class entities and transformations between models and from models to code can be defined and executed. MDA/MDE is currently being applied in many domains, such Web Engineering [14], Ontology Engineering [8], and UI development [11].

Recent technologies for improving end-user experience in Web 2.0 include the so called Rich Internet Applications (RIAs) [7] which provide advanced and more interactive UIs, similar to desktop applications, while minimizing network traffic overhead and increasing user usability and efficiency [23]. In our approach, the UI derived from OWL2 is based on frameworks of reusable components for RIAs. To enrich the UI generation, a presentation model tightly coupled to the domain ontology provides modeling support to customize presentation features of UIs for data-intensive Web applications. For practical implementation, two different Java-based frameworks for RIAs have been considered.

A tool supporting the method presented has been developed using MDE tools provided by the Eclipse Modeling Project³, and it is supported by the TwoUse Toolkit [17] for ontology creation and management.

The rest of this paper is organized as follows: Section 2 reviews related work. Next, Section 3 introduces RIAs frameworks for Web-based UI implementation. The presentation model and the proposed mapping from OWL2 to UI components are detailed in Section 4. Finally, main conclusions and future work are summarized.

2 Related work

There have been many earlier approaches on UI generation based on models and MDE. We can distinguish between those from the field of Web Engineering and those from the field of Human-Computer Interaction (HCI).

Web Engineering approaches the design and development of Web applications based on conceptual models [3], focusing on content, navigation and presentation models as the most relevant concerns in Web applications design [19]. Based on these models, the full Web application can be developed applying a model-driven approach, including the presentation layer composed of web pages, web forms, links, and so on.

Web Engineering offer rather mature and established methodologies for traditional Web applications, and the UI layer has been explicitly addressed in most approaches. But when we move to Semantic Web information systems, methodologies are still in a development phase [1]. Examples like SHDM [12], Hera [21] and WebML+Sem [1] offer a wide support for ontology languages, basically RDF (Resource Description Framework) and OWL, and focus on different semantic web technologies such as semantic model description, advanced query support, flexible integration, ontology reasoning, and more, but leaving UI aside. In this paper we address that open issue.

³ <http://www.eclipse.org/modeling/>

In HCI field a number of model-driven approaches for UI development have also arisen [18]. They are commonly based on models created with extensions of UML (Unified Modeling Language) for UIs modeling [5], but can also use textual formats based on XML (eXtensible Markup Language), as is widely explored in [11].

Among these approaches, the PEGASUS method [13] presents an effort to supply end-users with mechanisms for authoring Web-based applications using ontologies to specify knowledge for building data models together with presentation models. Moreover, it enables the generation of a Web UI from the ontology in basic HTML (HyperText Markup Language) and JSP (JavaServer Pages) code. In our approach we also use ontologies as domain models and a presentation model, but we focus on current rich Web UI generation using modern components of RIA frameworks which are richer in functionality than basic HTML and JSP.

In a recent work [6], the same authors provide a way of modeling UIs based on semantic models of domain problem, deriving a Web application for displaying content. The method is based on document transformation through a set of XSLT (Extensible Stylesheet Language Transformations) applied to XML files to generate documents for the UI. The result can incorporate AJAX (Asynchronous JavaScript And XML) components to have a better interactive result. Our proposal has similarities with this approach in sense that we also focus UI development with rich AJAX components, but has important differences with respect to the model-driven approach applied since we use MDE transformation languages and tools for defining the approach instead of XSLT transformations, as well as we derive a UI not only for displaying content but also for content acquisition and querying.

3 Frameworks for RIAs in JavaEE

Modern UI development requires the usage of extensive software libraries and frameworks, and code becomes rather platform-specific [11]. In this work we focus on Java Platform Enterprise Edition (JavaEE⁴) technology and the JavaServer Faces framework (JSF) [10] as implementation platform. To fit once of the most important characteristics of RIAs, a richer interaction is achieved adding AJAX technology to provide improved user experience. Several rich UI frameworks for JSF applications are available, some of them are well established such as RichFaces⁵ (JBoss project), ICEfaces⁶ (ICEsoft project), MyFaces⁷ (Apache project), ADF Faces⁸ (Oracle project) and Google Web Toolkit⁹ (Google project). In this

⁴ <http://java.sun.com/javaee>

⁵ <http://www.jboss.org/richfaces>

⁶ <http://www.icefaces.org/>

⁷ <http://myfaces.apache.org/>

⁸ <http://www.oracle.com/technetwork/developer-tools/adf/>

⁹ <http://code.google.com/webtoolkit/>

work, only RichFaces and ICEfaces have been considered, although the approach can easily be adapted to other frameworks.

JBoss RichFaces is an advanced JSF based framework that provides a complete range of rich AJAX enabled UI components. RichFaces is made up of two component tag libraries: *rich:ajax* represents core AJAX functionality, and *rich:rich* represents self contained and advanced components such as calendars, datatables, trees and more (see the RichFaces showcase¹⁰ for details). Current version, RichFaces 4.0, can be used in any container compatible with JSF 2.0.

ICEsoft ICEfaces is an integrated AJAX application framework that enables JavaEE application developers to easily create and deploy thin-client RIAs in Java. ICEfaces 2 is the current version of the open-source framework based on the JSF 2.0 standard. It offers a vast set of rich components included in the *ice:ice* tag library, to create rich advanced UIs (see the ICEfaces showcase¹¹ for details).

4 Model-driven rich user interface generation

4.1 Process overview

Our approach is based on the assumption that a UI can be induced from the ontology classes, properties and assertions. Since OWL2 semantics is richer than semantics of UI components, only a part of OWL2 can be represented in the UI and supported by the proposed mapping.

Fig. 1 shows the model-driven schema proposed for deriving rich Web UI from OWL2 ontologies. The process starts with the specification of an OWL2 ontology of the problem domain. In a first step, a presentation model with default presentation values is derived from the ontology applying a model-to-model (M2M) transformation. Developers can customize this presentation model to drive a better UI generation. Later, a model-to-text (M2T) transformation produces the final code.

The model-driven process proposed was implemented using MDE tools of Eclipse Modeling Project, and it is supported by the TwoUse Toolkit¹² for OWL2 authoring and management. The TwoUse Toolkit is a free, open source tool bridging the gap between Semantic Web and MDE, that supports OWL2 authoring based on the Ontology Definition Metamodel (ODM) [16]. In this environment, metamodels are defined with EMF¹³ (Eclipse Modeling Framework) in ecore format. The ODM metamodel is provided by TwoUse whereas the Presentation metamodel has been designed by the authors of this work.

M2M transformation is designed with ATL¹⁴ (Atlas Transformation Language), where as final code is generated by a M2T transformation implemented in JET¹⁵ (Java Emitter Templates). As a result, the code for the rich Web UI

¹⁰ <http://richfaces.org/showcase>

¹¹ <http://component-showcase.icefaces.org>

¹² <http://code.google.com/p/twouse/>

¹³ <http://www.eclipse.org/modeling/emf/>

¹⁴ <http://www.eclipse.org/m2m/atl/>

¹⁵ <http://www.eclipse.org/modeling/m2t/?project=jet>

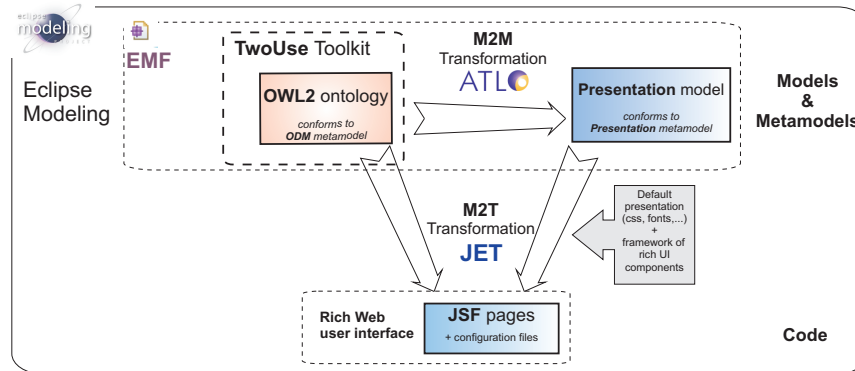


Fig. 1. MDE schema for rich Web user interface generation

is obtained as a set of JSF web pages based on the selected framework of rich components, RichFaces or ICEfaces. Default configuration is injected to provide presentation templates for pages, styles and css files.

4.2 The Presentation metamodel

To drive a powerful UI generation, the Presentation model captures features of UI components used in data-intensive applications, for example, in what other they are to appear, their visual appearance and layout, and more. Basically we propose three main types of presentation elements in the UI for data-intensive Web applications:

- a *menu*, including a hierarchy or tree with the class taxonomy
- a *list* page per ontology class, listing all instances of the class,
- a *form* page per class for viewing and editing instances of the class.

Fig. 2 shows the Presentation metamodel using a simplified UML class diagram notation. It defines the primitives that can be used in the modeling language, that is, in presentation models. Metaclasses in this metamodel are designed to allow the extension of presentation models by adding new features or modifying existing ones, enabling the process to evolve. For that purpose, a *Class* is related to a *MenuItem* which stores the feature(s) for displaying the class in the menu page, and to a *TableList* which stores the feature(s) for displaying the class in the list page. Similarly, a *Property* is related to a *FormField* which stores the features for displaying the property as a field in the form page, and to a *ListColumn* which stores the features for displaying the property as a column in the list page.

Tables 1 and 2 describe the main metaclasses and attributes that can be specified in a presentation model. How they are used in the generation of UI elements is explained in following section.

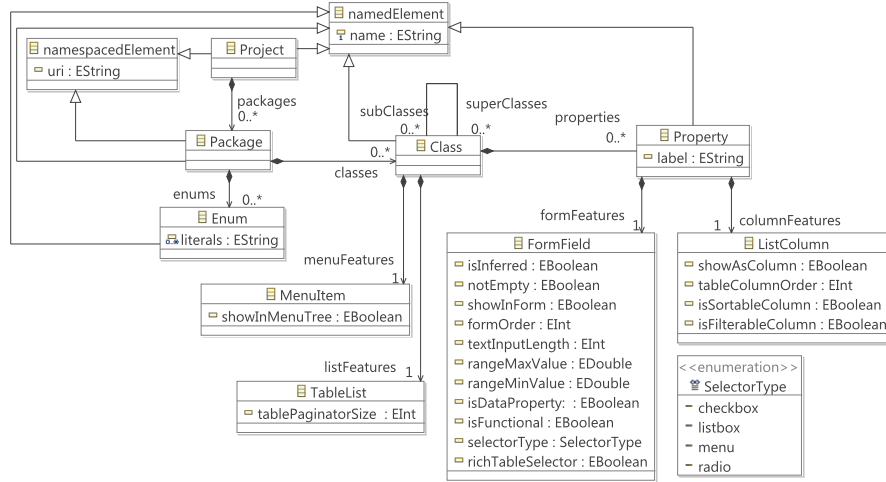


Fig. 2. Presentation metamodel

Table 1. Presentation metamodel: Class features

Metaclass	Feature	Type	Used in Page	Description	Default value
Class	name	string	all	name of the class	OWL2
	subClasses	Class[0..*]	menu	collection of subClasses	OWL2
	superClasses	Class[0..*]	menu	collection of superClasses	OWL2
	properties	Property[0..*]	form, list	collection of properties having the class as domain	OWL2
	menuFeatures	MenuItem	menu	link to presentation features for a class in the menu page	
	listFeatures	TableList	list	link to presentation features for a class in the list page	
MenuItem	showInMenuTree	boolean	menu	true if the class is showed in the menu (tree)	true
TableList	tablePaginatorSize	integer	list	rows per page in the list	10

4.3 Mapping OWL2 to user interface components

This section describes how the menu tree, list and form pages are derived.

Menu tree. Domain concepts are represented in OWL2 as named classes, which can have subclasses, conforming a hierarchy of classes. To display such a hierarchy a tree is commonly used. For that purpose, RichFaces provides a *rich:tree* component which renders a tree control on the page. Similarly, ICEfaces has the *ice:tree* component that displays hierarchical data as a tree of branches and leaf nodes. Only classes with the presentation feature *showInMenuTree* true-valued are shown in the tree.

Form page. For displaying and editing instances of a particular class, a rich form with the properties of the class is generated. In OWL2 each property has a domain and a range, and two types of properties are distinguished: datatype properties, relations between instances of classes and primitive data

Table 2. Presentation metamodel: Property features

Metaclass	Feature	Type	Used in Page	Description	Default value
Property	<input type="checkbox"/> <i>name</i>	string	form, list	name of the property	OWL2
	<input type="checkbox"/> <i>label</i>	boolean	form, list	text to show in the property field (form) or the table column (list)	prop. name
	<input checked="" type="checkbox"/> <i>fromFeatures</i>	FormField	form	link to presentation features for a property in a form field	
	<input checked="" type="checkbox"/> <i>columnFeatures</i>	ListColumn	list	link to presentation features for a property as a column of the table list	
FormField	<input type="checkbox"/> <i>isInferred</i>	boolean	form	true if the value is not editable	false
	<input type="checkbox"/> <i>notEmpty</i>	boolean	form	true if the value can not be empty	true
	<input type="checkbox"/> <i>showInForm</i>	boolean	form	true if the form includes a field for the property	true
	<input type="checkbox"/> <i>formOrder</i>	integer	form	field order in the form	null
	<input type="checkbox"/> <i>textInputLength</i>	integer	form	field size for the property value	30
	<input type="checkbox"/> <i>rangeMaxValue</i>	real	form	maximum value allowed for number type properties	null
	<input type="checkbox"/> <i>rangeMinValue</i>	real	form	minimum value allowed for number type properties	null
	<input type="checkbox"/> <i>isDataProperty</i>	boolean	form	true if the property type is primitive, false otherwise (object property)	OWL2
	<input type="checkbox"/> <i>isFunctional</i>	boolean	form	true if the property cardinality is as much one, false otherwise	OWL2
	<input type="checkbox"/> <i>selectorType</i>	SelectorType	form	for object properties, kind of selector showing related instances	menu
<input type="checkbox"/> <i>richTableSelector</i>	boolean	form	for object properties, use a rich datatable for selecting related instances	false	
ListColumn	<input type="checkbox"/> <i>showAsColumn</i>	boolean	list	true if the property is showed as a column in the list	true
	<input type="checkbox"/> <i>tableColumnOrder</i>	int	list	column order of the property in the table	null
	<input type="checkbox"/> <i>isSortableColumn</i>	boolean	list	true if the table can be sorted by column values	true
	<input type="checkbox"/> <i>isFilterableColumn</i>	boolean	list	true if the table can be filtered by column values	false

types (e.g. integer or string); and object properties, relations between instances of two classes. The default behavior in OWL determines that a property can relate an instance of the domain to multiple instances of the range, but defining the property as *functional* the relation is from an individual to only one primitive value (functional datatype property) or individual (functional object property). Similarly, a cardinality constraint can set the property range to 1. The form page for a particular class contains fields for all the properties having the class as domain and with a true value in the presentation feature *showInForm*.

The mapping of object properties to the UI implies that for non-functional properties, common selectors are used, such as *selectManyCheckbox*, *selectManyListbox* and *selectManyMenu* widgets, whereas for functional properties, *selectOneListbox*, *selectOneMenu* and *selectOneRadio* widgets can be used.

The widget used for an object property is set in the presentation feature *selectorType*. In case that a property has a true value in the presentation feature *isInferred*, a selector widget is not generated for it because its value can not be modified by users.

When an object property has the feature *richTableSelector* to true, then the values to be selected are shown in tables instead of in selectors. Rich-Faces provides the *rich:extendedDataTable* component for a powerful selection of one or many items. Similarly, ICEfaces includes the *ice:rowSelector* tag in the *ice:dataTable* component to provide that functionality. Fig. 3 shows an example of multiple data table selection.

ID	First Name	Last Name	Phone
140	Joshua	Brown	555-4579
150	Christopher	Brown	555-4580
160	Matthew	Brown	555-4581
170	Ryan	Jones	555-4582
180	Jason	Jones	555-4583
200	Kevin	Jones	555-4584
220	Daniel	Jones	555-4585
210	Matthew	Jones	555-4586

Navigation controls: Home, Previous, 1, 2, 3, 4, Next, End

Fig. 3. Multiple data table selection (3 rows selected)

To fulfill the list of values to be selected, individuals of the range type are listed. When an object property has the same class as domain and range types, then reflexive or irreflexive property axioms must be considered. In a irreflexive object property, the self individual (domain) is not included in the list of values to be selected (range), whereas in a reflexive one, all individuals are listed.

OWL2 provides several class extension constructs to define unnamed anonymous classes. The “oneOf” expression enables the definition of an enumerated class through the list of individuals that constitute the instances of the class. When the range of a property is an enumerated defined through a “oneOf” anonymous class, the enumeration literals obtained from the individuals linked to the enumeration are used as the list of allowed-values that can be selected. An example is the *hasSex* functional property from the *Person* class to the enumerated class with values $\{female, male\}$. In this case, one of the three *selectOne* selectors can be used in the interface.

It is possible to further constrain the range of a property with property restrictions. The “has-Value” restriction specifies an anonymous class based on the existence of particular property value. Other classes can be a subclasses of such a property restriction. As example, the *Woman* class is a subclass of “*hasSex has female*” property restriction, and similarly, the *Man* class is subclass

of “*hasSex has male*”. In the UI, this is mapped to default values that can not be editable by end-users.

List page. Finally, for listing all the instances or individuals of each named class, a page with a rich data table component is generated. Table columns are those properties of the class with a true value in the *showAsColumn* feature of the presentation model, and the column position in the table is established by the *tableColumnOrder* value. Both RichFaces and ICEfaces provide a *datatable* component with advanced functionality including a *paginator* widget for viewing the table as multiple pages of rows instead of as one large table, a *sortColumn* feature allowing the user to sort of data in the table, and *filterValue* feature for filtering data rows (only available in RichFaces). The corresponding features in the presentation model allow to customize these elements.

5 Conclusions and future work

In this work a model-driven method for generating rich Web UIs from OWL2 domain ontologies was presented, continuing a research focused on model-driven development of Web applications from ontologies and rules [2]. Our approach is based on the assumption that a UI for a data-intensive application can be induced from the domain ontology classes, properties and axioms. To obtain an enhanced result, a presentation model captures presentation features related to the UI. Since UI development is platform-specific task, JavaEE and JSF technologies for Web application development were chosen as target implementation in our research. Two frameworks of rich UI components were considered, although the approach can be extended to other frameworks. The proposal is tested with a proof of the concept tool.

The extension of the proposal to cover a larger set of OWL2 elements is considered as future work, as well as enhanced UI functionality to provide full Semantic Web information system generation from ontologies. Enriching the ontology with SWRL rules and analyzing how rules can affect to the UI is also considered as future work, focusing on how rules can provide UI adaptivity.

Acknowledgments. The authors wish to thank the Spanish Ministry of Education and Science for funding received under projects TIN2009-14372-C03-01 and PET2007-0033, and the Andalusian Regional Government under project P06-TIC-02411.

References

1. Brambilla, M., Facca, F.M.: Building semantic web portals with WebML. In: Web Engineering, 7th International Conference, ICWE 2007, Como, Italy. Lecture Notes in Computer Science, vol. 4607, pp. 312–327. Springer (2007)
2. Cañadas, J., Palma, J., Túnez, S.: A MDD approach for generating rule-based web applications from OWL and SWRL. In: 3rd Workshop on Transforming and Weaving Ontologies in Model Driven Engineering (TWOMDE 2010). vol. 604. CEUR Workshop Proceedings, Málaga, Spain (June 2010)

3. Ceri, S., Fraternali, P., Matera, M.: Conceptual modeling of data-intensive web applications. *Internet Computing, IEEE* 6(4), 20–30 (2002)
4. Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., Matera, M.: *Designing Data-Intensive Web Applications*. Morgan Kaufmann, 1 edn. (Dec 2002)
5. Cerny, T., Song, E.: A profile approach to using UML models for rich form generation. In: *Information Science and Applications (ICISA), 2010 International Conference on*. pp. 1–8 (2010)
6. Chavarriaga, E., Macías, J.A.: A model-driven approach to building modern semantic web-based user interfaces. *Advances in Engineering Software* 40(12), 1329–1334 (2009)
7. Driver, M., Valdes, R., Phifer, G.: Rich internet applications are the next evolution of the web. Tech. rep., Gartner Research Note. G (2005)
8. Gašević, D., Djurić, D., Devedžić, V.: *Model Driven Architecture and Ontology Development*. Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
9. Gašević, D., Kaviani, N., Milanović, M.: Ontologies and software engineering. In: Staab, S., Studer, R. (eds.) *Handbook on Ontologies*, pp. 593–615. Springer Berlin Heidelberg (2009)
10. Geary, D., Horstmann, C.S.: *Core JavaServer Faces*. Prentice Hall, 2 edn. (2007)
11. Hussmann, H., Meixner, G., Zuehlke, D.: *Model-Driven Development of Advanced User Interfaces*. Springer-Verlag New York Inc (2011)
12. Lima, F., Schwabe, D.: Application modeling for the semantic web. In: *LA-WEB*. pp. 93–102. IEEE Computer Society (2003)
13. Macías, J.A., Castells, P.: Providing end-user facilities to simplify ontology-driven web application authoring. *Interacting with Computers* 19(4), 563–585 (2007)
14. Moreno, N., Romero, J.R., Vallecillo, A.: An overview of Model-Driven Web Engineering and the MDA. In: *Web Engineering: Modelling and Implementing Web Applications*, pp. 353–382. Springer London (2008)
15. Object Management Group: *MDA Guide Version 1.0.1*. OMG document: omg/2003-06-01 (2003)
16. Object Management Group: *Ontology Definition Metamodel. Version 1.0*. OMG (2009), available at <http://www.omg.org/spec/ODM/1.0/>
17. Parreiras, F.S., Staab, S.: Using ontologies with UML class-based modeling: The TwoUse approach. *Data & Knowledge Engineering* 69(11), 1194–1207 (Nov 2010)
18. Pérez-Medina, J.L., Dupuy-Chessa, S., Front, A.: A survey of model driven engineering tools for user interface design. In: *Task Models and Diagrams for User Interface Design, 6th International Workshop, TAMODIA 2007, Toulouse, France. Lecture Notes in Computer Science*, vol. 4849, pp. 84–97. Springer (2007)
19. Rossi, G., Pastor, O., Schwabe, D., Olsina, L. (eds.): *Web Engineering: Modelling and Implementing Web Applications*. Human-Computer Interaction Series, Springer London, London (2008)
20. Schmidt, D.C.: Guest Editor’s Introduction: Model-Driven Engineering. *Computer* 39(2), 25–31 (2006)
21. Vdovjak, R., Frasincar, F., Houben, G.J., Barna, P.: Engineering Semantic Web Information Systems in Hera. *J. Web Engineering* 2(1-2), 3–26 (2003)
22. W3C OWL Working Group: *OWL 2 Web Ontology Language: Document Overview*. W3C Recommendation (2009), available at <http://www.w3.org/TR/owl2-overview/>
23. Wright, J.M., Dietrich, J.: Requirements for rich internet application design methodologies. In: *Web Information Systems Engineering - WISE 2008, 9th International Conference, Auckland, New Zealand. Lecture Notes in Computer Science*, vol. 5175, pp. 106–119. Springer (2008)

Loki – Presentation of Logic-based Semantic Wiki

Weronika T. Adrian, Grzegorz J. Nalepa

AGH University of Science and Technology,
al. A. Mickiewicza 30, 30-059 Krakow, Poland
{wta,gjn}@agh.edu.pl

Abstract. TOOL PRESENTATION: The paper presents a semantic wiki, called Loki, with strong logical knowledge representation using rules. The system uses a coherent logic-based representation for semantic annotations of the content and implementing reasoning procedures. The representation uses the logic programming paradigm and the Prolog programming language. The proposed architecture allows for rule-based reasoning in the wiki. It also provides a compatibility layer with the popular Semantic MediaWiki platform, directly parsing its annotations.

1 Motivation

Semantic wikis enrich standard wikis with the semantic information expressed by a number of mechanisms. Three basic questions every semantic wiki needs to address are¹: 1) how to annotate content?, 2) how to formally represent content?, 3) how to navigate content? In last several years multiple implementations of semantic wikis have been developed, including IkeWiki², OntoWiki³, Semantic MediaWiki⁴, and AceWiki⁵. The summary of semantic wiki systems is available⁶.

From the knowledge engineering point of view, simply expressing semantics is not enough. A knowledge-based system should provide both effective knowledge representation and processing methods. In order to extend semantic wikis to knowledge-based systems, ideas to use rule-based reasoning and problem-solving knowledge have been introduced. An example of such a system is the KnowWE semantic wiki⁷. The system allows for introducing knowledge expressed with decision rules and trees related to the domain ontology.

Logic-based Wiki [5], or *Loki* for short, uses the logic programming paradigm to represent knowledge in the wiki. The main design principles are as follows: 1) provide an expressive underlying logical representation for semantic annotations and rules, 2) allow for strong reasoning support in the wiki, 3) preserve backward compatibility with existing wikis, namely Semantic MediaWiki.

¹ See <http://aran.library.nuigalway.ie/xmlui/handle/10379/574>.

² See <http://ikewiki.salzburgresearch.at/>.

³ See <http://ontowiki.net/Projects/OntoWiki>.

⁴ See http://semantic-mediawiki.org/wiki/Semantic_MediaWiki.

⁵ See <http://attempto.ifi.uzh.ch/acewiki/>.

⁶ See http://semanticweb.org/wiki/Semantic_Wiki_State_Of_The_Art.

⁷ See www.is.informatik.uni-wuerzburg.de/en/research/applications/knowwe.

2 Architecture

A prototype implementation of Loki, called PIWiki (Prolog Wiki), has been developed [2,4]. The main goal of the system design is to deliver a generic and flexible solution. Thus, instead of modifying an existing wiki engine or implementing a new one, an extension of the DokuWiki⁸ system has been developed.

The architecture can be observed in Fig. 1. The stack is based on a simple runtime including the Unix environment with the Unix filesystem, Apache web server and PHP stack. Using this runtime, a standard DokuWiki installation is run. The Loki functionality is implemented with the use of Dokuwiki plugins allowing to enrich the wikitext with semantic annotations and Prolog clauses, as well as run the SWI-Prolog interpreter. It is also possible to extend the wikitext with explicit semantic information encoded using RDF and OWL representation. This layer uses the Semantic Web library provided by SWI-Prolog.

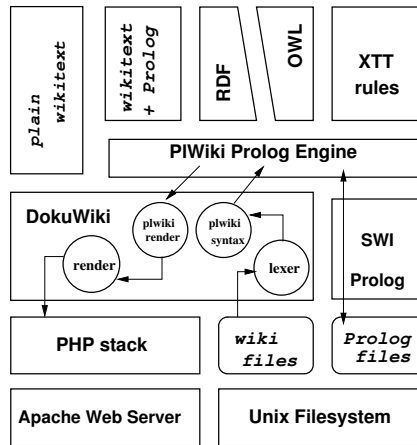


Fig. 1. Loki Architecture – PIWiki Implementation

3 Features

The main idea of Loki consists in representing the semantic annotations in a formalized way, and simultaneously enriching them with an expressive rule-based knowledge representation. Both semantic annotations and Prolog clauses (facts and rules) may be embedded within in the wiki content. The resulting knowledge base is homogeneous from the logical point of view.

SMW Support There are three main methods of semantic annotations in Semantic MediaWiki (SMW) that are supported in Loki: categories, relations and attributes. Loki also supports the query language used in SMW.

⁸ See <http://www.dokuwiki.org>.

Categories, relations, attributes and queries are represented by appropriate Prolog terms. Technically, SMW markup is converted to Prolog code and then saved in a Loki file related to a Wiki page. Examples (annotations written on a page for "The Call of Cthulhu" book) are as follows, with the SMW syntax given first, and the Prolog representation below.

```
[[category:book]]
    wiki_category('book','the_call_of_cthulhu').
**Author**: [[author:h_p_lovecraft]]
    wiki_relation('the_call_of_cthulhu','author','h_p_lovecraft').
**Date**: [[date:=2011]]
    wiki_attribute('the_call_of_cthulhu','date','2011').
{{#ask: [[category:book]] [[author:h_p_lovecraft]]}}
    wiki_category('book',Page),
    wiki_relation(Page,'author','h_p_lovecraft').
```

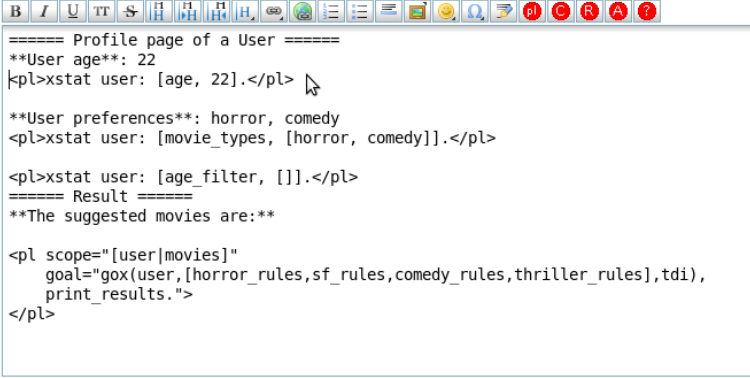
Semantic Web Standards Support RDF annotations can be embedded directly in the XML serialization. They are parsed by the SWI-Prolog *semweb/rdflib* library, and turned to the internal representation. Within the wikitext, a SPARQL query (SELECT, ASK or DESCRIBE) may be posed. The query scope is the whole wiki system. Analogously to the SMW-like queries, SPARQL ones are also translated to Prolog goals and then executed by the wiki engine.

Semantic information gathered in Loki may be exported to the RDF/XML format. The exported file consists of a header with used namespaces, metadata of the exported page, and optionally ontological information about categories, relations and attributes. Categories are exported as OWL Classes, relations between pages as Object Properties, and attributes as Datatype Properties. Information about subcategories and subproperties is exported with the use of `rdfs:subClassOf` and `rdfs:subProperty`.

Rule-based Reasoning An optional rule layer is provided using the HeaRT [3] runtime for the XTT2 framework [6]. XTT2 (eXtended Tabular Trees v2) knowledge representation uses attributive table format. Rules based on the same attributes are grouped within tables, and the system is split into a network of such tables representing the inference flow. XTT2 rules can be serialized into a HMR (HeKatE Meta Representation) format, supported in Loki.

An example rule: `xrule a/1: [age in[18to100],movie_types sim[comedy]]==> [age_filter set union(age_filter,[adult_comedy]):comedy_rules.` would be interpreted as: for users who are older than 18 and like comedies adjust the *age_filter* attribute and redirect the inference to *comedy_rules* table.

HeaRT (HeKatE RunTime), a dedicated inference engine for the XTT2 rule bases, has been added to Loki as a part of the plugin responsible for parsing Prolog. HMR code is embedded on wiki pages within the `<p1></p1>` tags (see Fig. 2). To run reasoning, a `<p1 scope="" goal="">` tag is used. If the goal is a valid HeaRT command, the reasoning is performed by the engine, the result is calculated and rendered on a wiki page. Embedding HeaRT in Loki is currently in an experimental phase and is not provided with the current release.



```
==== Profile page of a User =====
**User age**: 22
<pl>xstat user: [age, 22].</pl>

**User preferences**: horror, comedy
<pl>xstat user: [movie_types, [horror, comedy]].</pl>

<pl>xstat user: [age_filter, []].</pl>
==== Result =====
**The suggested movies are:**

<pl scope="[user|movies]"
  goal="gox(user,[horror_rules,sf_rules,comedy_rules,thriller_rules],tdi),
  print_results.">
</pl>
```

Fig. 2. Goal query on user profile page

4 Summary

In the paper, a semantic wiki called Loki has been presented. An essential feature of the system is a strong rule-based reasoning thanks to a coherent knowledge representation. In the system, standard semantic annotations are mapped to the Prolog knowledge base, in which also rule-based reasoning is specified. Moreover, a custom rule-based engine using decision tables and trees is provided. Loki allows for the development of modularized knowledge bases with the use of a wiki. In future, Loki is planned to be used as a platform for knowledge evaluation [1].

References

1. Baumeister, J., Nalepa, G.J.: Verification of distributed knowledge in semantic knowledge wikis. 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. pp. 384–389. FLAIRS, AAAI Press, Menlo Park, California (2009)
2. Nalepa, G.J.: PIWiki – a generic semantic wiki architecture. In: Nguyen, N.T., Kowalczyk, R., Chen, S.M. (eds.) Computational Collective Intelligence. Semantic Web, Social Networks and Multiagent Systems, First International Conference, ICCCI 2009, Wroclaw, Poland, October 5-7, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5796, pp. 345–356. Springer (2009)
3. Nalepa, G.J.: Architecture of the HeaRT hybrid rule engine. In: Rutkowski, L., [et al.] (eds.) Artificial Intelligence and Soft Computing: 10th International Conference, ICAISC 2010: Zakopane, Poland, June 13–17, 2010, Pt. II. Lecture Notes in Artificial Intelligence, vol. 6114, pp. 598–605. Springer (2010)
4. Nalepa, G.J.: Collective knowledge engineering with semantic wikis. *Journal of Universal Computer Science* 16(7), 1006–1023 (2010)
5. Nalepa, G.J.: Loki – semantic wiki with logical knowledge representation. In: Nguyen, N.T. (ed.) Transactions on Computational Collective Intelligence III, Lecture Notes in Computer Science, vol. 6560, pp. 96–114. Springer (2011)
6. Nalepa, G.J., Ligeza, A.: HeKatE methodology, hybrid engineering of intelligent systems. *International Journal of Applied Mathematics and Computer Science* 20(1), 35–53 (2010)