10th Workshop on

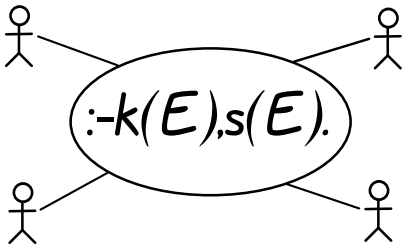
Knowledge Engineering and Software Engineering (KESE10)

at the

21st European Conference on Artificial Intelligence (ECAI 2014)

Prague, Czech Republic, August 19, 2014

Grzegorz J. Nalepa and Joachim Baumeister (Editors)



Technical Report No. 492, Würzburg University, Würzburg, Germany, 2014

The KESE Workshop Series is available online: http://kese.ia.agh.edu.pl

Technical Reports of the Würzburg University: http://www.informatik.uni-wuerzburg.de/forschung/technical_reports

Preface

Grzegorz J. Nalepa and Joachim Baumeister

AGH University of Science and Technology
Kraków, Poland
gjn@agh.edu.pl
—
denkbares GmbH
Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
joachim.baumeister@denkbares.com

Research questions and practical exchange between Knowledge Engineering for intelligent systems and Software Engineering of advanced software programs have been fruitfully discussed over the last years. Many successful examples demonstrate the clear symbiosis between these two research areas.

In 2005 the KESE workshops took place for the first time in Koblenz at the 28th German Conference on Artificial Intelligence (KI-2005). In 2014 the KESE10 workshops was collocated with the 21st European Conference on Artificial Intelligence (ECAI 2014) in Prague, Czech Republic on August 19. This year we solicited contributions having the following topics:

- Knowledge and software engineering for the Semantic Web,
- Knowledge and software engineering for Linked Data,
- Ontologies in practical knowledge and software engineering,
- Business systems modeling, design and analysis using KE and SE,
- Practical knowledge representation and discovery techniques in software engineering,
- Context and explanation in intelligent systems,
- Knowledge base management in KE systems,
- Evaluation and verification of KBS,
- Practical tools for KBS engineering,
- Process models in KE applications,
- Software requirements and design for KBS applications,
- Software quality assessment through formal KE models, and
- Declarative, logic-based, including constraint programming approaches in SE.

As from the beginning the workshop series shows a healthy mixture of advanced research papers showing the direction to the next years and practical papers demonstrating the actual applicability of approaches in (industrial) projects and concrete systems. This year six regular, and two short papers were accepted to the workshop. Moreover, two tool presentations were also included. We decided to organize the presentations during the workshop into three topical sessions.

The first session, entitled *Knowledge Modeling* (Chair: Grzegorz J. Nalepa) included two papers and one tool presentation. Freiberg and Puppe discuss the

use of patterns in engneering of knowledge-based systems. In their tool presentation Furth and Baumeister demonstrate an ontology debugger integrated in a semantic wiki. Hatko et al. apply behaviour-driven development for medical applications.

The second session, entitled *Business Processes in KE&SE (Chair: Thomas Roth-Berghofer)* included three papers. Nguyen and Le-Thanh discuss semantic aspects of workflows. Bobek et al. present a recommender system for business process design using a Bayesian network. In their paper Sanfilippo et al. provide remarks on an ontological analysis of BPMN.

The third session *Systems and Tools (Chair: Martina Freiberg)* included four papers. Ostermayer et al. presented a custom connector architecture for Prolog and Java. Then Ślażyński et al. shared their experiences in migrating rule inference engines to mobile platforms. In their paper Bach et al. discussed knowledge modeling aspects with the open source tool myCBR. Finally, Kluza et al. presented a new wiki-based tool for SBVR.

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. We would like to thank the Chairmans that supported the KESE Chairs during the event. For the submission and reviewing process we used the EasyChair system, for which the organizers would like to thank all the developers of the system. Last but not least, we would like to thank the organizers of the ECAI2014 conference for hosting the KESE10 workshop.

Grzegorz J. Nalepa
Joachim Baumeister

Workshop Organization

The 10th Workshop on Knowledge Engineering and Software Engineering
(KESE10)
was held as a one-day event at the
21st European Conference on Artificial Intelligence
(ECAI 2014)
on August 19 2014 in Prague, Czech Republic

Workshop Chairs and Organizers

Grzegorz J. Nalepa, AGH UST, Kraków, Poland
Joachim Baumeister, denkbares GmbH, Germany
Krzysztof Kaczor, AGH UST, Kraków, Poland

Programme Committee

Klaus-Dieter Althoff, University Hildesheim, Germany
Isabel María del Águila, University of Almeria, Spain
Thomas-Roth Berghofer, University of West London, UK
Kerstin Bach, Verdande Technology AS, Norway
Joachim Baumeister, denkbares GmbH/University Würzburg, Germany
Joaquín Cañadas, University of Almeria, Spain
Adrian Giurca, BTU Cottbus, Germany
Jason Jung, Yeungnam University, Korea
Rainer Knauf, TU Ilmenau, Germany
Mirjam Minor, Johann Wolfgang Goethe-Universität Frankfurt, Germany
Pascal Molli, University of Nantes - LINA, France
Grzegorz J. Nalepa, AGH UST, Kraków, Poland
José Palma, University of Murcia, Spain
Alvaro E. Prieto, Univesity of Extremadura, Spain
Dietmar Seipel, University Würzburg, Germany
José del Sagrado, University of Almeria, Spain

Table of Contents

Knowledge Modelling (Chair: Grzegorz J. Nalepa).

Pattern-driven Knowledge Systems Engineering 1
Martina Freiberg, Frank Puppe

An Ontology Debugger for the Semantic Wiki KnowWE (Tool Presentation) 13
Sebastian Furth, Joachim Baumeister

Behaviour-Driven Development for Computer-Interpretable Clinical Guidelines 24
Reinhard Hatko, Stefan Mersmann and Frank Puppe

Business Processes in KE&SE (Chair: Thomas Roth-Berghofer).

Ensuring the Semantic Correctness of Workflow Processes: An Ontological Approach 30
Thi-Hoa-Hue Nguyen, Nhan Le-Thanh

Integration of Activity Modeller with Bayesian Network Based Recommender for Business Processes 42
Szymon Bobek, Grzegorz J. Nalepa and Olgierd Grodzki

Towards an Ontological Analysis of BPMN 53
Emilio M. Sanfilippo, Stefano Borgo and Claudio Masolo

Systems and Tools (Chair: Martina Freiberg).

CAPJA - A Connector Architecture for Prolog and Java 59
Ludwig Ostermayer, Frank Flederer and Dietmar Seipel

Migration of Rule Inference Engine to Mobile Platform. Challenges and Case Study. 71
Mateusz Slażyński, Szymon Bobek and Grzegorz J. Nalepa

Knowledge Modeling with the Open Source Tool myCBR 84
Kerstin Bach, Christian Severin Sauer, Klaus-Dieter Althoff and Thomas Roth-Berghofer

SBVRwiki (Tool Presentation) 95
Krzysztof Kluza, Krzysztof Kutt and Marta Woźniak

Pattern-driven Knowledge Systems Engineering

Martina Freiberg and Frank Puppe

Department of Artificial Intelligence and Applied Informatics, Institute of Computer Science, University of Würzburg, Am Hubland, D-97074 Würzburg, Germany
freiberg/puppe@informatik.uni-wuerzburg.de

Abstract. Despite increasing relevance in research- and industrial contexts, the implementation of knowledge-based systems (KBS) still is a challenging task. We motivate, that patterns—basically a renowned means for providing reusable solutions for similar problems—can drastically leverage development efforts and time. In this paper, we propose a framework for pattern-driven, encompassing KBS development, consisting of: Tailored usability criteria for a clear delimitation of KBS solutions, a basic pattern specification template, and a collection of foundational KBS UI patterns. We further describe practical experiences with the approach, entailing: The reference implementation of several patterns with the tailored development tool ProKEt, their design- and usability-related evaluation, and empirical evidence of applying pattern-driven KBS development in actual projects.

1 Introduction

Despite increasing relevance in research- and industrial contexts, knowledge-based systems (KBS) engineering still denotes a challenging task. In general software engineering, patterns are renowned for describing proven solutions and preventing common pitfalls, thus fostering reuse and strongly leveraging overall development efforts. To date, various pattern collections for general UI- and interaction design are proposed, including [7, 8, 9]; also, many resources are available freely on the web.

In contrast to standard web pages or query forms, KBS do not solely build on strictly predefined question sequences; rather, they characteristically live from follow-up items—flexibly included interview items that become relevant only during the questioning session and based on the concrete user input. Also, KBS often require the prominent integration of additional information for elaborating the knowledge base / interview items more clearly and of (in-place) explanations of the results. This calls for tailored UI and interaction solutions that best support those requirements.

We motivate, that also KBS engineering can strongly profit from fitted patterns that subsume such proven (and optimally evaluated) KBS solutions; this can strongly support and refine requirements engineering, and leverage encompassing KBS development. First steps into that direction have been already taken with regards to the knowledge base, e.g., [6]. As counterpart, we propose tailored

patterns, that capture foundational KBS interaction and UI design solutions regarding various contexts and target objectives; to the best of our knowledge, no similar efforts have been published so far.

The rest of the paper is organized as follows: In Section 2, we introduce a basic KBS pattern specification framework: A short definition of relevant terms, tailored (usability-related) classification criteria, and a KBS UI pattern specification template. A collection of four framing patterns, that can be fine-tuned into a total of ten pattern variants, is proposed in Section 3. Practical experiences related to pattern-driven KBS development are reported in Section 4, and a summarizing conclusion and promising future work are presented in Section 5.

2 KBS UI Pattern Specification Framework

Before proposing a set of usability-related KBS delimitation criteria in Section 2.1 and sketching a basic pattern specification template in Section 2.2, we clarify some basic terms used in the remaining paper.

Forward- & backward progression: *Forward progression* starts with an empty solution set; from one or more init questions, such a KBS then questions in all directions, depending on the particularly implemented indication mechanisms. In contrast, a *backward progression* KBS is initialized with a target solution and poses only those questions that contribute to the final state of that chosen solution.

Multiplex consultation- & clarification KBS: *Multiplex consultation KBS* use forward progression, whereas *clarification KBS* base on backward progression. Clarification KBS can further be used with two application focuses: Consultation focussed—i.e., the clarification KBS is started empty, and all contributing questions are posed. Or justification focussed—then, such a system is called for justifying a solution that already has been derived in the preceding session, thus corresponding to an elaborate, interactive solution explanation.

2.1 Tailored, Usability-related KBS Classification Criteria

Today, diverse UI design- and usability guidelines and standards are available, such as Nielsen's heuristics [5] or the universal design guidelines of Lidwell [4]. However, those typically are defined rather generally as to be applicable for diverse interactive software system types. Due to their specifics as mentioned in the introduction, KBS require more tailored criteria; those then can be used for clearly delimitating their specific characteristics—as, e.g., for the pattern specification in this work—or for rating KBS solutions regarding their quality and usability. We propose eight tailored, usability-related classification criteria as follows:

1. Compactness: *How many interview items are presented simultaneously?*

2. Comprehensibility: *Is support provided for understanding specialist, complex, or ambiguous knowledge base contents (additional explanations, surrounding, contextual questions), and in learning something about the domain?*

3. Descriptiveness: *Does the KBS suggest how respective questions/answers influence the final result of the session, e.g., by indicating the score (change)?*

4. Efficiency: *How long does a characteristic session take and how many interview items need to be processed?*

5. Explorability (Participation): *Are users enabled to deviate from the suggested questioning sequence, are potential expert shortcuts provided?*

6. Intuition (usage): *Are the applied presentation/interaction forms familiar or otherwise self-descriptive? If not, are particularly novice users supported (instructions, tutorials, examples)?*

7. Transparency: *Is the current state (i.e., state of questions, results, overall progress) clearly and comprehensibly mediated at any time?*

8. Clear Arrangement/Aesthetics: *Does the overall design exhibit certain aesthetics, e.g., by using a small number of virtual lines & basic symmetry?*

2.2 KBS UI Pattern Specification Template

Table 1 summarizes a basic template for specifying KBS UI patterns in a unified and clear manner. All variations of a base pattern exhibit some similar characteristics, e.g., the core KBS objective. They vary regarding the specific realization of the UI/interaction, the extent of adhering to KBS classification criteria (see Section 2.1), the target users, knowledge specifics, and the imposed consequences. In the following pattern descriptions, we only provide a summarizing template item *variations* that subsumes specifics regarding the UI/interaction and required knowledge; we further delimitate the differences regarding the classification criteria and the target users in Table 2; consequences, as well as details on the example implementations, are omitted here due to space restrictions.

Pattern Section	Description
Problem Statement	Specifies the problem, that is solved by this pattern, based on the the tailored KBS usability criteria as described in Section 2.1.
Solution	Describes the general (UI and interaction) solution that all variants of this pattern apply, e.g., the characteristic base interaction.
Variations	Variations of the fundamental pattern, differing regarding: The targeted *user types*, the specific *UI realization*, *knowledge specifics*, (*consequences*, and *example implementation details*—not elaborated in this paper).

Table 1: Basic template for specifying fundamental KBS UI patterns.

3 KBS UI Pattern Collection

We propose four basic KBS UI patterns: *Questionnaire*, *Interview*, *Clarifier*, and *Clarifier Hybrid*, along with variants. In earlier research, we already introduced

three basic interaction categories for KBS, see [1]; there, *Adaptive Entry* maps to *Questionnaire*, *Guided Entry* to *Interview*, and *Autonomous Entry* to *Clarifier*. The patterns basically are intended independent from a specific knowledge representation—in the sense that for the pattern/UI it is irrelevant whether a rule-base or a covering model derives the solution ratings; however, some patterns are favorable regarding specific knowledge characteristics—e.g., CheckList Questionnaire requires all questions to be mappable on a fixed answer set; we included short remarks on such specifics in the variations' descriptions.

3.1 Questionnaire Pattern

Problem Statement: The KBS should *compactly* display a greater part of the KB, offer a *highly transparent UI*, *intuitive usage*, and a c*ertain extent of explorability*; comprehensibility, is no key requirement for the core UI itself.

Solution: Questionnaire KBS resemble paper- or web-based questionnaire forms. Depending on the particular UI style, many to all indicated interview objects are displayed simultaneously and typically ordered in some form of grid-based layout. Questionnaire may suggest (visually), but does not necessarily prescribe, an optimal interrogation sequence and thus fosters explorative usage. A certain comprehensibility can be achieved by adding auxiliaries—such as informative popups with additional explanations for interview items. Per default, Questionnaire uses forward progression, c.f., Section 2

Variations: Box-, Daily-, and CheckList Questionnaire.

a. Box Questionnaire closely adheres to the design of standard questionnaires by using established, familiar question presentation forms—e.g., checkboxes and radio buttons for choice questions, see Figure 1, I; thereby, each question is rendered within a distinct box, resulting in a very regular layout, but some waste of UI space.

b. Daily Questionnaire, originally inspired by daily newspapers, realizes a more flat, juxtaposed presentation style for questions and answers, c.f., Figure 1, II; therefore, each question along with its answer options is placed in a line, implying a less regular yet more compact layout than the Box variant.

c. CheckList Questionnaire mirrors paper-based check lists by representing answer options by columns that simply are ticked off, see Figure 1, III. Therefore, all choice questions need to be mappable to a fixed answer set; including further types, e.g., numerical questions, is possible, yet best results regarding efficient interaction and compactness are achieved with choice questions only.

3.2 Interview Base Pattern

Problem Statement: The KBS UI should be *highly intuitive* and *easily comprehensible*, thus specifically supporting novice users/domain laymen; in turn, compactness, descriptiveness, efficiency, explorability, as well as UI transparency can be neglected.

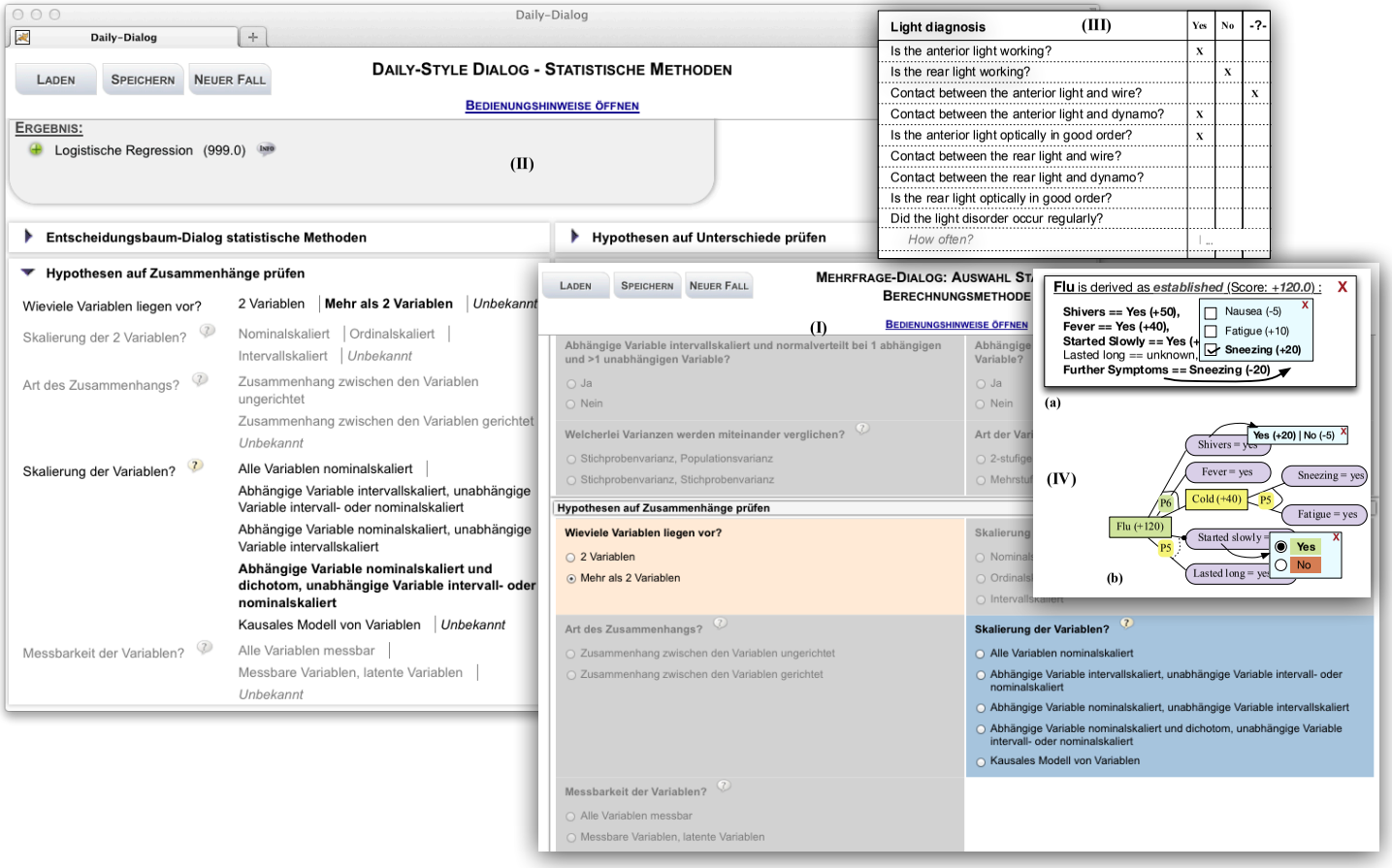


Fig. 1: KBS Patterns—examples 1: Box Questionnaire (I) and Daily Questionnaire (II) as implemented in ProKEt, CheckList Questionnaire (III), and Form Add-on Clarifier (IV) as preliminary sketch.

Solution: Interview imitates human conversations by presenting always a single question—or a group of related questions—at a time; additional information are available anytime quickly and easily, e.g., by directly integrating it near to the corresponding interview object. Interview typically prescribes the interrogation sequence in a rather fixed manner. The basic lack of transparency can be alleviated by integrating auxiliaries such as an interview object history—listing the already processed interview items—and a progress information display.

Variations: Strict-, Grouped-, and Hierarchical Interview.

d. Strict Interview displays only a single question at a time together with its additional information, see Figure 2, I. Thus, optimally, the KB should provide suitable auxiliary information for each interview item. Further, a sophisticated indication mechanism is advisable for keeping the possible interrogation paths at solid lengths, especially regarding large KBs.

e. Grouped Interview sequentially displays groups of (optimally topically related) questions or single questions; thus, it is a bit more efficient than Strict Interview, and offers more explorability, as the particular sequence within question groups is not prescribed. The UI uses a similar basic frame as Strict Interview, where Questionnaire variants are used for rendering the groups, c.f., Figure 2, II.

f. Hierarchical Interview offers an interactively navigable tree UI specifically for decision tree knowledge, c.f., [6]. Answer options are presented as width-spanning tree nodes, c.f., Figure 2, III. Clicking on answer nodes induces their expansion by the answer options of the follow up question. Solutions are represented by distinct nodes at the final nesting levels; thus, the tree path from outer nodes to the solution particularly enables users to 'read' the justification of a solution from the visual tree structure. Auxiliary information is presented in a dedicated side panel—either on click or by hovering the nodes.

3.3 Clarifier Base Pattern

Problem Statement: Selected single solutions in highly expertise domains should be investigated exclusively and in-depth. The KBS UI should be *compact*, *transparent*, *descriptive*, and offer skill-building ability induced by a *high comprehensibility* of the contents; users should be enabled to increase efficiency in contributing their personal knowledge, e.g., for using shortcuts regarding the interrogation sequence (*explorability/participation*). Intuitive usage, in contrast, is no key requirement.

Solution: Clarifier characteristically uses backward knowledge, see Section 2, for investigating only a single issue at a time. Therefore, Clarifier renders the target solution and all contributing questions—i.e., that potentially alter the solution rating in any way—simultaneously and offers means to adapt answers and thus investigate the consequences on the solution.

Variations: Hierarchical Clarifier and Form Add-on Clarifier.

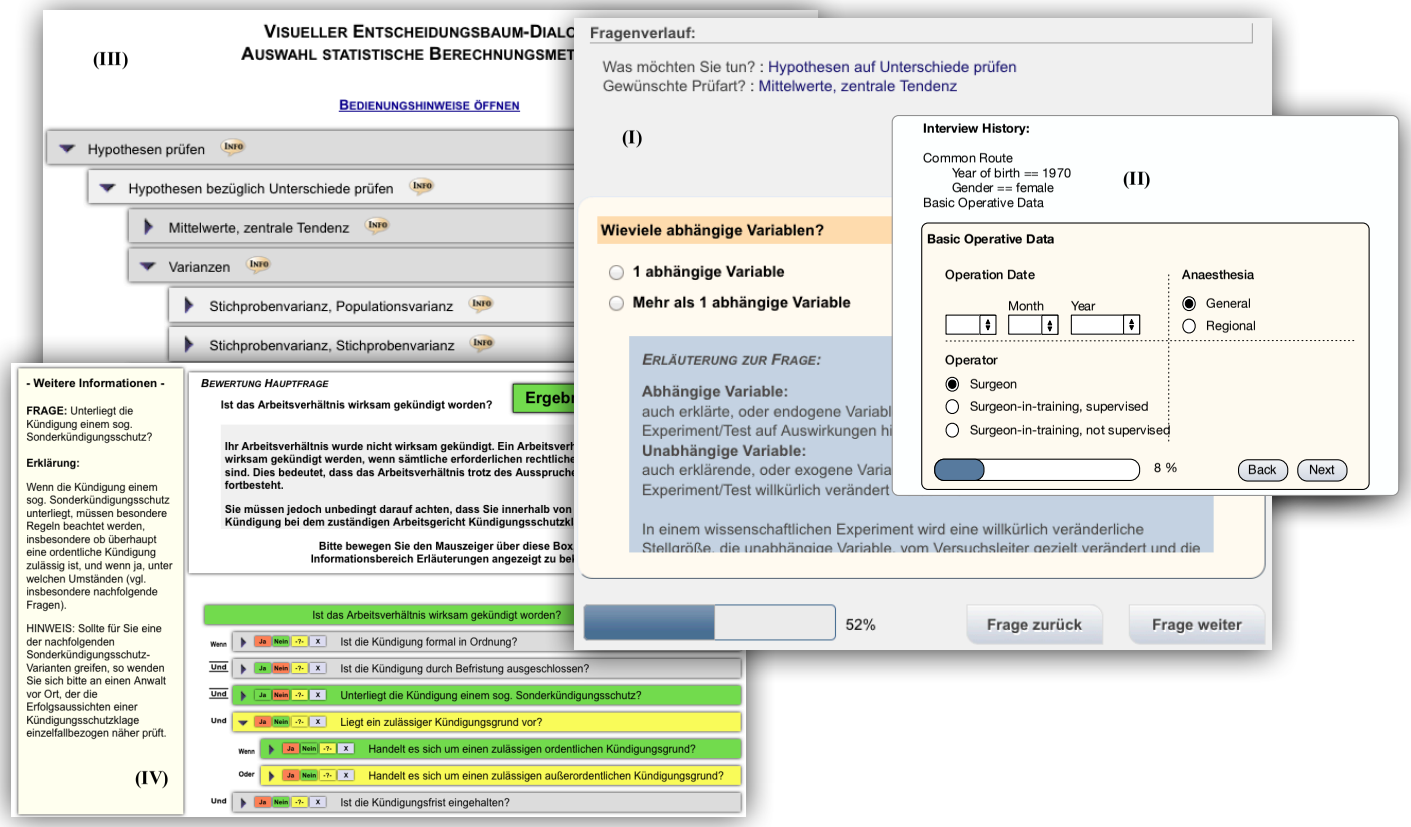


Fig. 2: KBS Patterns—Examples 2: Strict Interview (I), Hierarchical Interview (III), and Hierarchical Clarifier (IV) as implemented in ProKEt, Grouped Interview (II) as preliminary sketch.

g. Hierarchical Clarifier exhibits its strengths optimally with backward knowledge that is refined over several abstraction levels. It displays question and answers options within one node of its tree-style UI, see Figure 2, IV. The topmost node corresponds to the target solution, and is followed by nodes that correspond to the directly solution-relevant questions. Each question node again can be followed recursively by further question levels where the children denote a more fine-granular partition—one or several questions—of its parent. Thus, users decide whether to answer the more abstract top-level questions; or whether to implicitly answer them by expanding them and processing the children—child answers then are propagated recursively back to the parents.

h. Form Add-on Clarifier adds minimalistic consultation widgets to static base justification presentation forms, such as finding lists or rule graphs, c.f., Figure 1, IV a&b. Clicking the interview objects in the justification automatically triggers compact (popup) consultation widgets; those contain all answer options for the respective question, potentially also indicating the value that is added to the solution. This allows for interactively adapting answers and thus exploring and clarifying a selected solution based on its general justification view.

3.4 Clarifier Hybrid Pattern

Problem Description: A more intuitively usable and easily comprehensible UI representation for using clarification knowledge is desired.

Solution: Clarifier Hybrids merge intuitive, comprehensible KBS UIs with backward knowledge for supporting especially also novice or laymen users in using clarification KBS. Both Questionnaire and Interview patterns are suitable for using backward knowledge. The base implementation of Clarifier Hybrid then corresponds to the variants described in Sections 3.1 and 3.2; in contrast to that, the targeted backward knowledge is processed, which might—depending on the actually selected UI variant—require some additions to the base UI; for example, widgets for explicitly navigating the hierarchical refinement levels.

Variations: Clarification Interview and Clarification Questionnaire.

3.5 Pattern Variants—Detailed Delimitation

Table 2 summarizes the fine-grained delimitation of the proposed patterns according to the tailored classification criteria, introduced in Section 2.1; the extent of their realization is rated from low (L) to high (H). Further, corresponding target user characteristics for each pattern are specified by classifying the domain-expertise and the frequency of usage. Thus, the delimitation table serves as quick reference which pattern to apply in what context. If, e.g., a KBS solution is requested that is both highly intuitive usable, and—given the appropriate knowledge—highly comprehensible, also and especially for first-time users, then the Strict Interview pattern suits best, c.f., Table 2.

		Questionnaire			Interview			Clarifier		Hybrid	
		3.1.a	3.1.b	3.1.c	3.2.d	3.2.e	3.2.f	3.3.g	3.3.h	3.4	3.4
Usability Criteria	**compact**	M	H	H	L	M	H	H	H	L	M
	comprehensible	M	M	M	H	H	L-M	M-H	M	M-H	M-H
	descriptive	L	L	L	L	L	L	H	H	L	L
	efficient	M	H	H	L	M	M	M	L-M	M	L-M
	explorable	H	H	H	L	L	M-H	H	H	L	H
	intuitive	H	M	H	H	M	L-M	L	M	M-H	H
	transparent	H	H	H	M	M	M	H	H	L-M	H
	clear arranged	H	M	H	L	M	L-M	M	L-M	L	M
Users	**expertise: laymen**	X	X	X	X	X	(X)	/	(X)	X	X
	experienced	X	X	X	X	X	X	X	X	X	X
	expert	(X)	X	X	/	(X)	X	X	X	(X)	X
	usage: one-time	X	X	X	X	X	(X)		(X)	X	X
	frequent	X	X	X	(X)	X	X	X	X	(X)	X

Table 2: KBS UI pattern delimitation according to tailored usability-related KBS classification criteria with extent of their realization—High (H), medium (M), and low (L)—and further, regarding two user characteristics—expertise, and usage frequency.

4 Practical, Pattern-Driven KBS Development

Our practical experiences related to KBS UI patterns encompass: Support of pattern-driven KBS development by the tailored tool ProKEt [3]; a usability assessment of selected pattern reference implementations with ProKEt; and several current projects where KBS patterns/pattern-driven development was beneficial.

4.1 ProKEt: Tool Support for Pattern-driven KBS development

We already introduced ProKEt as a tailored KBS engineering tool that specifically supports template-based development in [3]: By realizing the KBS UI framework through defining highly modular (HTML) templates with varying complexity, that can recursively be assembled into more comprehensive ones. That main mechanism still persists, yet the collection of supported templates and readily available KBS patterns has been extended. Currently supported are Box- and Daily Questionnaire, along with encompassing options for simple property-based fine-tuning—e.g., regarding whether to hide non-indicated items; Strict- and Hierarchical Interview, with an optional interview history and progress bar; a configurable solution panel display for all those variants; and Hierarchical Clarifier, with a tailored add-on information display and a specific solution panel variant.

4.2 Evaluation of Selected KBS UI Patterns

In early 2014, an encompassing evaluation was conducted regarding ProKEt reference implementations of the following KBS UI patterns: Box-/Daily Questionnaire, Strict-/Hierarchical Interview, and Hierarchical Clarifier. Therefore, first

an expert evaluation was conducted by 30 HCI students, using heuristic evaluation according to Nielsen [5] and the cognitive walkthrough technique [10]; the basic goal was to assess the demo implementations regarding their basic quality and usability. Afterwards, in total 248 computer science students participated in a more comparative user study, where certain given problem descriptions were to be solved with each of the tested KBS; there, students were further instructed to fill in a short questionnaire for collecting some basic unified values—e.g., regarding the overall utility of the KBS—and to provide informal feedback.

Evaluation Item	**Daily**	**Box**	**HInterv**	**SInterv**	**HClari**
Success Rate = Correctly solved/all cases	**88.71**	**91.53**	**27.02**	**20.16**	**88.31**
Questionnaire Items					
Q1. Overall utility of the system	**2.04**	**1.93**	**3.63**	**3.06**	**2.72**
Q3. Belief in the result's correctness	**1.68**	**1.76**	**4.13**	**3.86**	**3.03**
Q4. KB quality=content,structure	**2.24**	**2.16**	**3.73**	**3.08**	**2.82**
Q5. Knowledge mediation	**3.79**	**3.77**	**3.64**	**3.18**	**2.78**
Q6. Perceived ease of use	**1.95**	**1.57**	**3.30**	**2.28**	**3.03**
Q7. Efficiency of the KBS	**2.01**	**1.84**	**3.45**	**2.86**	**2.83**
Q9. Rating of the KBS UI design	**3.58**	**2.60**	**3.84**	**2.57**	**3.12**

Table 3: Results of the comparative user study regarding five selected KBS patterns, ratings from 1 (very good) to 6 (very bad). Not explicitly listed are Q2 which concerned the acquired solution (mirrored in the success rate), and Q8 (resolution of the screen).

Table 3 summarizes the questionnaire-based results of the comparative study (rating scale: 1/very good – 6/very bad). The first major finding is a strong correlation between KB quality Q4 and each of KB utility Q1 (0.9968), KBS efficiency Q7 (0.9813), and perceived correctness of the result Q3 (0.9571), correlation coefficient given in parentheses. Further, KB quality Q4 correlates quite strong with the overall KBS success (0.8325); thus overall, the KB quality can be assumed one major influencing factor regarding the overall KBS perception. This in turn explains the bad overall results for both Interview variants, despite their way more positive rating in the expert usability assessment: Both variants used a qualitatively rather unfavorable statistical KB—confirmed strongly also by subjective remarks. Yet, regarding Strict Interview, at least the basic tendency of the expert assessment—which confirmed a highly intuitive overall UI/interaction—was confirmed, see Q6 and Q9.

Box Questionnaire obviously received the best ratings, closely followed by the Daily variant; along with provided subjective remarks this indicates, that the more structured presentation of the Box variant was favored over the compact Daily layout, thereby consenting with the basic expert assessment findings. Apart from underlining this tendency, however, subjective remarks specifically criticized the more space consuming presentation of Box Questionnaire and the general lack of structure in Daily; those comments also revealed more approval

for Daily regarding its ease of use, simplicity, and efficiency. Thus, we suspect an even further increase in the overall rating of Daily in case it is further enhanced regarding its presentation—including, e.g., a clearer distinction of question and answer items, a more prominent highlighting of answered items, and providing overall visual structure by indicating separators between question/answer lines.

Regarding Hierarchical Clarifier, the ratings may seem improvable; yet, this KBS addressed a highly complex KB from the domain of protection against unlawful dismissal, with an equally comprehensive problem description of the dismissal conditions, the correctness of which was to be rated by the KBS. Thus, an utility value of 2.68 and even the more a success rate of 88.31 % are particularly good results in the given context of a highly expertise domain but domain laymen users. Especially the descriptive and transparent style of Hierarchical Clarifier, mirroring the derived question/solution ratings directly in the UI may have supported that result; it most likely fostered the overall trustworthiness Q3 (compared to the Interview variants).

As a general important insight it excelled clearly, that the evaluation of a KBS UI always is inherently coupled with the applied KB and the respective problem to be solved.

4.3 Case Studies with pattern-driven KBS Development

Pattern-driven development along with the tool ProKEt already proved highly beneficial in actual projects. First, we noticed a strong support of the requirements engineering process. In the Mediastinitis project—where a documentation KBS for the structured input of operation data is realized, c.f., [3]—the patterns, and their ProKEt reference implementations, provided a visual and interactive means for gathering the user requirements more precisely and quickly. Thus, it was easy to experiment with several Questionnaire variants—two distinct Box layouts and one Daily layout—and to let the user formulate his requirements based on practically investigating the ProKEt pattern demos.

Another advantage is the fostered reuse of KBS solutions. In the EuraHS project, c.f. also [3], nearly the same constraints and conditions existed as in Mediastinitis. Thus, it was quickly agreed that a similar KBS solution would fit best in that case, too. There, the availability of the Questionnaire reference implementation in ProKEt drastically accelerated the initial setup of a first functional demo system—which was gradually refined, particularly regarding the KB, later, yet the project initiation itself was highly accelerated and eased.

Similarly, in the JuriSearch project, see [2]—aiming at providing clarification consultation modules for diverse legal topics—we could easily experiment with a Hierarchical Clarifier and a (preliminary) hybrid Clarification Interview variant regarding the most suiting solution.

5 Conclusion

In this paper, we motivated the benefits of pattern-driven development in the context of KBS. For practical support, we proposed a pattern specification frame-

work, based on tailored KBS (usability) criteria and a pattern template, and we introduced a collection of 10 fundamental KBS UI/interaction patterns. Further, we reported practical experiences with the approach: Using the tailored KBS engineering tool ProKEt for creating reference implementations of the patterns, evaluating five of them regarding their design and usability, and empirical experiences with pattern-driven KBS development from current projects.

Despite denoting an exciting approach for leveraging encompassing KBS development, there are several aspects worth investigating in the future: First, the general extension of the tool ProKEt as to entirely support those patterns that are specified theoretically only so far. Second, a thorough usability- and design-related base evaluation of the not yet assessed patterns. Third, an in-depth inquiry of one assumption that has emerged in the conducted study: That a structural enhancement of Daily Questionnaire may entail even better results compared to Box Questionnaire. Fourth, follow-up studies for investigating patterns in direct comparison for delimitating core characteristics even clearer—e.g., the pure Clarifier variants vs. one or both Clarifier Hybrids. Another goal is the actual application of further selected patterns in current (and potential future) projects; e.g., using Clarification Interview as a first-time user alternative for the Hierarchical Clarifier in the JuriSearch project.

References

1. Freiberg, M., Baumeister, J., Puppe, F.: Interaction Pattern Categories—Pragmatic Engineering of Knowledge-Based Systems. In: Proceedings of the 6th Workshop on Knowledge Engineering and Software Engineering (KESE-2010) at the 33rd German Conference on Artificial Intelligence (2010)
2. Freiberg, M., Puppe, F.: iTree: Skill-building User-centered Clarification Consultation Interfaces. In: Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD 2012). SciTePress Digital Library (2012)
3. Freiberg, M., Striffler, A., Puppe, F.: Extensible prototyping for pragmatic engineering of knowledge-based systems. Expert Systems with Applications 39(11), 10177 – 10190 (2012)
4. Lidwell, W., Holden, K., Butler, J.: Universal Principles of Design. Rockport Publishers Inc., 2nd edn. (2010)
5. Nielsen, J.: Heuristic Evaluation. In: Nielsen, J., Mack, R.L. (eds.) Usability Inspection Methods, pp. 25–62. John Wiley & Sons, New York (1994)
6. Puppe, F.: Knowledge Formalization Patterns. In: Proceedings of PKAW 2000, Sydney Australia (2000)
7. Rogers, Y.: Interaction Design—beyond human-computer interaction. John Wiley & Sons, Ltd, 3rd edn. (2011)
8. Schmettow, M.: User interaction design patterns for information retrieval systems (2006)
9. Tidwell, J.: Designing Interfaces — Patterns for Effective Interaction Design. O'Reilly Media Inc. (2006)
10. Wharton, C., Rieman, J., Lewis, C., Polson, P.: Usability inspection methods. chap. The Cognitive Walkthrough Method: A Practitioner's Guide, pp. 105–140. John Wiley & Sons, Inc., New York, NY, USA (1994)

An Ontology Debugger for the Semantic Wiki KnowWE
(Tool Presentation)

Sebastian Furth[1] and Joachim Baumeister[1,2]

[1] denkbares GmbH, Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
{firstname.lastname}@denkbares.com
[2] University of Würzburg, Institute of Computer Science, Am Hubland, 97074 Würzburg, Germany

Abstract. KnowWE is a semantic wiki that provides the possibility to define and maintain ontologies and strong problem-solving knowledge. Recently, the ontology engineering capabilities of KnowWE were significantly extended. As with other ontology engineering tools, the support of ontology debugging during the development of ontologies is the deficient. We present an Ontology Debugger for KnowWE that is based on the delta debugging approach known from Software Engineering. KnowWE already provides possibilities to define test cases to be used with various knowledge representations. While reusing the existing testing capabilities we implemented a debugger that is able to identify failure-inducing statements between two revisions of an ontology.

1 Introduction

In software engineering changing requirements and evolving solutions are well-known challenges during the software development process. Agile software development [3] became popular as it tackles these challenges by supporting software engineers with methods based on iterative and incremental development.

In the field of ontology engineering the challenges are similar. It is extremely rare that the development of an ontology is a one-time task. In most cases an ontology is developed continuously and collaboratively. Even though this insight is not new and tool support has improved in recent years, a mature method for agile ontology development still is a vision.

The idea of continuous integration (CI) has been adapted for the development of knowledge systems (cf. Baumeister et al. [1] or Skaf-Molli et al. [14]). CI for knowledge system uses automated integration tests in order to validate a set of modifications. In software engineering the continuous integration is applied by unit and integration tests to mostly manageable sets of changes, which often is sufficient to isolate bugs. In accordance to Vrandečić et al. [16], who adapted the idea of unit testing to ontologies, we consider unit tests for ontologies to be difficult to realize. Additionally in ontology engineering changes can be rather complex, e.g. when large amounts of new instance data is extracted from texts and added automatically to an ontology.

As abandoning a complete change set because of an error is as unrealistic as tracing down the failure cause manually, a method for isolating the fault automatically is necessary. We developed a debugging plugin for the semantic wiki KnowWE that is able to find the failure-inducing parts in a change set. The debugger is based on the Delta Debugging idea for software development proposed by Zeller [18].

The remainder of this paper is structured as follows: Section 2 describes the delta debugging approach for ontologies; in Section 3 we present the developed plugin for KnowWE in detail. Section 4 contains a short case study while Section 5 concludes with a discussion and the description of future work.

2 Delta Debugging for Ontologies

2.1 Prerequisites

The proposed Delta Debugging approach assumes that we are able to access different revisions of an ontology. Additionally, we assume that a mechanism for the detection of changes exists. We call the difference between two revisions the set of changes C. The detection can be realized for example by utilizing revision control systems like SVN or Git or by manually calculating the difference between two snapshots of an ontology. Even more sophisticated change detection approaches are possible, like the one proposed by Goncalves et al. [5] for the detection of changes in OWL ontologies on a semantic level.

Definition 1 (Changes). *Let $C = \{c_1, c_2, \ldots, c_n\}$ be the set of changes c_i provided by a change detection mechanism.*

Definition 2 (Revision). *Let O_1 be the base revision of an ontology and $O_2 = O_1 \cup C$ a revision of the ontology, where the set of changes C have been applied.*

With respect to a given test one of the revisions has to pass while the other one has to fail a specific test (Axiom 1).

Definition 3 (Test Function). *A function $test : O \rightarrow BOOLEAN$ determines for a given ontology whether it passes ($TRUE$) a specified test procedure or not ($FALSE$).*

Axiom 1 *For a given test function test and the revisions O_1 and O_2 of the ontology $test(O_1) = TRUE$ and $test(O_2) = FALSE$ holds.*

2.2 Tests for Ontologies

Test outcome The Delta Debugging approach we propose is not limited to a certain test procedure as long as it can assert a boolean value as defined in Definition 3 and Axiom 1. In software engineering the outcome of a test can also be undefined. Zeller [18] pointed out three reasons why this can happen:

Failure 1 *Integration: When a change relies on earlier changes that are not included in the currently focused change set, the change may not be applied.*

Failure 2 *Construction: When applying all changes a program may have syntactical or semantical errors which avoids the construction of the program.*

Failure 3 *Execution: A program can not be executed.*

As ontology engineering is basically about adding/removing/changing triples, these failures can hardly occur—at least on the syntactical level. Incorrect statements can usually not be added to a repository and therefore should not detected as a valid/applicable change by the change detection mechanism. Additionally triples do syntactically not depend on other triples and therefore can be added to and removed from a repository independently. Finally ontologies are not executed in the way a program is, what relaxes the execution failure. On the semantical level, however, integration and construction failures are very likely to occur but they do not result in an undefined test outcome, but a failing test—which is the desired behavior.

Example tests A test could consider for example the result of a SPARQL query. A concrete implementation could compare the actual query result with an explicitly defined (expected) result. Another realization could use SPARQL's ASK form.

When dealing with an ontology that makes heavy use of semantics like OWL, a reasoner like Pellet [13] could be utilized in a test to check whether an ontology is consistent and/or satisfiable.

A test does not even have to test the ontology itself, as in task-based ontology evaluation [8] the outcome of the ontology's target application could be considered. Testing with sample queries for semantic search applications is an example, where a given ontology is expected to provide certain results in an otherwise unchanged semantic search engine.

Regardless of the actual implementation the definition of test cases should be a substantial and integral part of the underlying ontology engineering methodology. We described the TELESUP project [4] that aims for a methodology and tool for ontology development in a self-improving manner and emphasizes on the early formulation of test cases.

2.3 The Delta Debugging Algorithm

We propose a delta debugging algorithm (Algorithm 1) for ontologies that is basically a divide-and-conquer algorithm recursively tracing down the faulty parts of an ontology.

The input of the recursive algorithm is the base revision of the ontology O_1 that is known to pass the specified test procedure *test*. Additionally the set of changes C between this base revision and the failing revision O_2 is provided.

Algorithm 1 The delta debugging algorithm for ontologies.

```
function DeltaDebug(O_1, C, test)
    if C.length is 1 then
        return C
    end if
    r ← {}
    for all c_i in Divide(C) do
        O_t ← O_1 ∪ c_i
        if test(O_t) is FALSE then
            r ← r + DeltaDebug(O_1, c_i, test)
        end if
    end for
    return r
end function
```

If the considered change set only contains one change then this is the failure-inducing change by definition. Otherwise the helper function Divide slices the set of changes in i new change sets. The function may use heuristics or exploit the semantics of the ontology to divide the initial change set. In the following each change set c_i proposed by the Divide function is applied to the base revision O_1 of the ontology. If the resulting revision of the ontology O_t does not pass the specified test procedure, then the change set is recursively examined in order to find the triple responsible for the failure. As more than one recursive call of the DeltaDebug algorithm can return a non empty set of failure inducing changes, the final result may contain more than one triple.

The shown version of the algorithm returns all changes that applied to the base revision O_1 cause the failure. It additionally assumes monotonicity, i.e. a failure occurs as long as the responsible changes are contained in the change set. A more sophisticated handling of interferences will be subject of future work.

3 Implementation

3.1 KnowWE

We have implemented the delta debugging algorithm as an extension of the semantic wiki KnowWE [2]. KnowWE provides the possibility to define and maintain ontologies together with strong problem-solving knowledge. Ontologies can be formulated using the RDF(S) or OWL languages. KnowWE provides different markups for including RDF(S) and OWL: proprietary markups, turtle syntax, and the Manchester syntax. KnowWE compiles ontologies incrementally, i.e. only those parts of an ontology get updated that are affected by a specific change. This is possible as KnowWE's incremental parsing and compiling mechanism is able to keep track of which markup is responsible for the inclusion of a specific statement. Thus statements can easily be added to or removed from the repository when a specific markup has been changed.

3.2 Change Detection Mechanism

We use a dedicated change log to keep track of all changes applied to an ontology in KnowWE. Each time a change is applied to the repository KnowWE's event mechanism is used to fire events that inform about the statements that have been added to and removed from the repository. For every change a log entry is created. Listing 1.1 shows an example of log entries, that indicate the removal (line 1) and addition (line 2) of statements at the specified timestamps.

Listing 1.1. Example change log

```
1  -;1401619927398;si:abraham;rdfs:label;Abraham Simpson
2  +;1401619927401;si:abraham;rdfs:label;Abraham Simson
```

The change detection mechanism can now be realized by accessing the log file and asking for the changes between two points in time. The ontology revisions O_1 and O_2[3] can be constructed by reverting all changes between a specified start point and the currently running revision of the ontology (HEAD). The set of changes C between these two revisions can be extracted directly from the log file.

3.3 Tests in KnowWE

In order to realize the *test* function, we have introduced a Java interface called `OntologyDeltaDebuggerTest` which requires implementors to realize the method `boolean execute(Collection<Statement> statements)`.

We have implemented a sample test that checks whether a revision of an ontology is able to provide specified results for a SPARQL query. We exploit the already existing possibilities of KnowWE to formulate and execute labeled SPARQL queries. In the following, we use an exemplary ontology inspired by the comic characters "The Simpsons"[4].

Listing 1.2. Example for an expected SPARQL result.

```
1  %%SPARQL
2  SELECT ?s
3  WHERE {
4    ?s rdf:type si:Human;
5       si:gender si:male;
6       rdfs:label ?name .
7    FILTER regex(str(?name), "Simpson")
8  }
9  @name: maleSimpsons
10 %

12 %%ExpectedSparqlResult
13 |si:abraham
14 |si:homer
15 |si:bart
16 @sparql: maleSimpsons
17 @name: maleSimpsonsExpected
18 %
```

[3] The revision O_2 is constructed to check whether Axiom 1 holds.
[4] http://en.wikipedia.org/wiki/The_Simpsons

Additionally we use KnowWE's feature to define expected results for a specified query. This can be done by adding the expected results to a table and referencing a labeled SPARQL query. For the convenient formulation a special markup has been introduced. Listing 1.2 shows an example where `si:homer` and `si:bart` are the expected results of the SPARQL query with the label "maleSimpsons". In order to access the formulated expected results, the markup also gets a label ("maleSimpsonsExpected"). The actual test is instantiated using this label, which allows accessing the expected results as well as the underlying SPARQL query.

3.4 Ontology Debugger

The Delta Debugger for Ontologies is realized by the markup `OntologyDebugger` that allows for the convenient configuration of the debugger. The configuration is done by specifying the base revision O_1 using the `start` annotation, optionally the revision O_2 can be specified using the `end` annotation. If not specified the current revision of the ontology (HEAD) is considered as O_2. Using the annotation `expected` the label of the expected SPARQL result is defined.

Listing 1.3. Example for the definition of an Ontology Debugger.

```
1  %%OntologyDebugger
2    @expected: maleSimpsonsExpected
3    @start: 1401267947599
4  %
```

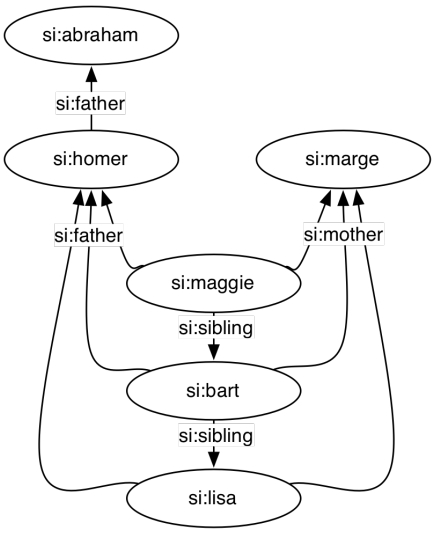
The so defined ontology debugger instance is rendered like depicted in Figure 1. A tool menu allows the execution of the debugger, a progress bar is used to visualize the running process. The actual implementation of the delta debugging algorithm for ontologies has been realized as `LongOperation` that is a feature of KnowWE's framework architecture, which allows for executing long operations in background without having the user to wait for the result. When the long operation has finished, then the failure-inducing changes are returned and displayed. An error message is rendered instead, if a failure occurs during the execution of the debugger, e.g. because the test is undefined or Axiom 1 does not hold for the specified revisions.

Fig. 1. The ontology debugger in KnowWE.

4 Case Study

4.1 The Simpsons Ontology

Fig. 2. An excerpt of the ontology showing the relationships of the Simpson family.



In the following we describe a small case study that illustrates the functionality of the presented ontology debugging extension for KnowWE. Therefore we use an example ontology that has been developed for tutorial purposes and covers various facts of the popular comic television series "The Simpsons". The ontology contains several classes like `Human` or `Building`, as well as properties like `parent` or `owns`. Additionally, some instances of the defined classes are included. We do not present the entire ontology but concentrate on some relevant parts.

Figure 2 shows relationships of the Simpsons family, e.g. that Homer (`si:homer`) is the father (`si:father`) of Bart (`si:bart`), Lisa (`si:lisa`) and Maggie (`si:maggie`), who are also marked as siblings (`si:sibling`). The sibling property was initially defined as `owl:TransitiveProperty`, i.e. a triple that explicitly states that Lisa is sibling of Maggie is not necessary. We have also defined that `si:father` is a sub-property of `si:parent`, which has an inverse property `si:child`. Listing 1.4 describes a SPARQL query for all children of Homer (`si:homer`) and Marge (`si:marge`).

Listing 1.4. SPARQL query for the children of Homer and Marge.

```
%%SPARQL
SELECT ?kid
WHERE {
  ?kid rdf:type si:Human .
  si:homer si:child ?kid .
  si:marge si:child ?kid .
}
@name: simpsonsKids
%
```

An expert on the Simpson family knows that Bart, Lisa and Maggie are the expected result of this query. So this knowledge can be defined in KnowWE as an expected SPARQL result (Listing 1.5), which than can be used as a test case for the ontology.

Listing 1.5. Expected results of the query for the children of Homer and Marge.

```
%%ExpectedSparqlResult
|si:maggie
|si:bart
|si:lisa
@name: simpsonsKidsExpected
@sparql: simpsonsKids
%
```

Listing 1.6 is another example containing a SPARQL query for all siblings of Maggie (`si:maggie`) and the definition of the expected result (`si:bart` and `si:lisa`).

Listing 1.6. A test case for Maggie's siblings.

```
%%SPARQL
SELECT ?sibling
WHERE {
  BIND (si:maggie as ?kid) .
  ?kid si:sibling ?sibling .
  FILTER (?kid != ?sibling) .
}
@name: maggiesSiblings
%

%%ExpectedSparqlResult
|si:bart
|si:lisa
@name: maggiesSiblingsExcpected
@sparql: maggiesSiblings
%
```

For this case study various changes have been applied to the ontology (see Listing 1.7) and broke it finally, i.e. the SPARQL results do not return the expected results: Bart can not be retrieved as sibling of Maggie, and apparently Homer and Marge do not have any children.

Listing 1.7. Changes applied to the Simpsons ontology.

```
-;1401267947600;si:snowball;rdfs:label;Snowball
+;1401267947605;si:santas_little_helper;rdfs:label;Santa's little helper@en
+;1401267947605;si:snowball;rdfs:label;Snowball II
-;1401268045755;si:child;owl:inverseOf;si:parent
+;1401283675264;si:sibling;rdf:type;owl:IrreflexiveProperty
-;1401283841549;si:relatedWith;rdf:type;owl:ReflexiveProperty
+;1401283841552;si:relatedWith;rdf:type;owl:SymmetricProperty
-;1401283907308;si:sibling;rdf:type;owl:TransitiveProperty
-;1401287487640;si:Powerplant;rdfs:subClassOf;si:Building
```

In order to find the failure-inducing changes, we have defined two ontology debugger instances that utilize the test cases defined above. Listing 1.8 shows their definitions. Revision 1401267947599 is the base revision O_1 for both instances as we know that the queries had been working before we started changing the ontology.

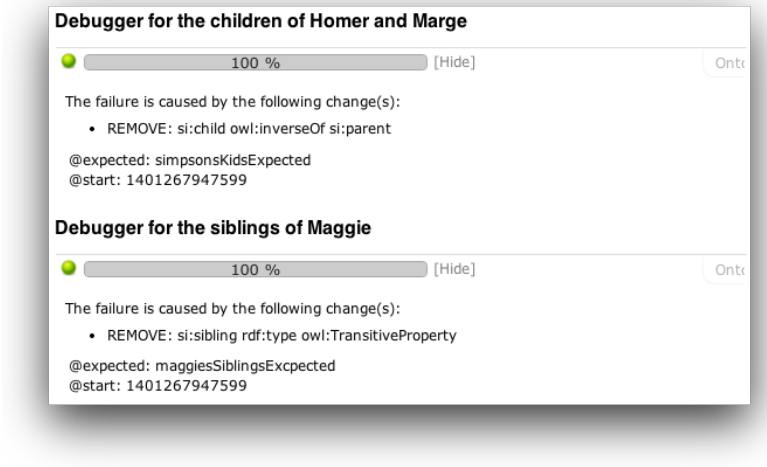
Listing 1.8. Changes applied to the Simpsons ontology.

```
%%OntologyDebugger
  @expected: simpsonsKidsExpected
  @start: 1401267947599
%

%%OntologyDebugger
  @expected: maggiesSiblingsExcpected
  @start: 1401267947599
%
```

After manually triggering the debugging process, the ontology debugger instances return the correct results. As depicted in Figure 3 the first ontology debugger instance identified that removing the statement declaring `si:child` as inverse property of `si:parent` has caused the failure that the children of Homer and Marge could not be retrieved. The second instance reports that Bart is not identified as sibling of Maggie because the transitivity (`owl:TransitiveProperty`) has been removed from the `si:sibling` property.

Fig. 3. The result of running the ontology debugger.



We ran the case study on an Apple MacBook with 3 GHz Intel Core i7 processor and 8 GB RAM; the example ontology contained 2,518 triples. The ontology debugger returned each result after about 1.2 seconds on.

4.2 Ontology of an Industrial Information System

The ontology debugger was also utilized to trace down a failure-inducing change in an ontology for an industrial information system. The ontology comprises more than 700,000 triples and makes heavy use of OWL2-RL semantics. In this scenario the debugger returned the correct hint for the failure-inducing change after 5 minutes. The manual tracing of the error would have costed many times over the presented automated approach.

5 Conclusion

In this paper we presented an extension for KnowWE that adds support for ontology debugging by using a divide-and-conquer algorithm to find failure-inducing changes. Our current implementation is working on the syntactical level of an ontology and uses a provided test case in combination with a change detection mechanism and a heuristic for dividing the change set. However, the design of the algorithm and the software allows for the incorporation of more sophisticated methods that may consider semantics to leverage the debugging to a semantic level.

There has already been work on considering semantics for the debugging of ontologies. Schlobach et al. [10] coined the term *pinpointing* which means reducing a logically incorrect ontology, s.t. a modeling error could be more easily detected by a human expert. They also proposed algorithms that use pinpointing to support the debugging task [11]. Ribeiro et al. [9] proposed the usage of Belief Revision to identify axioms in OWL ontologies that are responsible for inconsistencies. Wang et al. [17] proposed a heuristic approach that considers OWL semantics in order to explain why classes are unsatisfiable. Shchekotykhin et al. [12] proposed an interactive debugging approach to address the problem that in OWL ontologies more than one explaination for an error can exist and additional information is necessary to narrow down the problem. As debugging OWL ontologies is closely related to the justification of entailments the work in this field must also be considered. See for example Horridge et al. [6] or Kalyanpur et al. [7]. However, Stuckenschmidt [15] questions the practical applicability of several debugging approaches of OWL ontologies with respect to scalability and correctness.

While already functional we consider the current implementation of our ontology debugging plugin for KnowWE as early work. For the future we plan several enhancements to the debugger, like the replacement of the change log by a change detection mechanism that is based on standard wiki functionality providing direct access to different revisions of an ontology. As mentioned above we want to improve the handling of interferences, check the monotonicity assumption for different ontology languages and plan to examine the applicability of OWL debugging approaches. As KnowWE also allows for the definition of strong problem-solving knowledge the generalization of the debugger to other knowledge representations will be subject of future work.

Acknowledgments

The work described in this paper is supported by the Bundesministerium für Wirtschaft und Energie (BMWi) under the grant ZIM KF2959902BZ4 "SELESUP – SElf-LEarning SUPport Systems".

References

1. Baumeister, J., Reutelshoefer, J.: Developing knowledge systems with continuous integration. In: Proceedings of the 11th International Conference on Knowledge Management and Knowledge Technologies. p. 33. ACM (2011)
2. Baumeister, J., Reutelshoefer, J., Puppe, F.: KnowWE: a Semantic Wiki for knowledge engineering. Applied Intelligence 35(3), 323–344 (2011)
3. Cockburn, A.: Agile Software Development (2002)
4. Furth, S., Baumeister, J.: TELESUP Textual Self-Learning Support Systems. In: under review (2014)
5. Gonçalves, R.S., Parsia, B., Sattler, U.: Analysing the evolution of the NCI thesaurus. In: Computer-Based Medical Systems (CBMS), 2011 24th International Symposium on. pp. 1–6. IEEE (2011)
6. Horridge, M., Parsia, B., Sattler, U.: Laconic and precise justifications in OWL. In: The Semantic Web-ISWC 2008, pp. 323–338. Springer (2008)
7. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: The Semantic Web, pp. 267–280. Springer (2007)
8. Porzel, R., Malaka, R.: A task-based approach for ontology evaluation. In: ECAI Workshop on Ontology Learning and Population, Valencia, Spain. Citeseer (2004)
9. Ribeiro, M.M., Wassermann, R.: Base revision for ontology debugging. Journal of Logic and Computation 19(5), 721–743 (2009)
10. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: IJCAI. vol. 3, pp. 355–362 (2003)
11. Schlobach, S., Huang, Z., Cornet, R., Van Harmelen, F.: Debugging incoherent terminologies. Journal of Automated Reasoning 39(3), 317–349 (2007)
12. Shchekotykhin, K., Friedrich, G., Fleiss, P., Rodler, P.: Interactive ontology debugging: two query strategies for efficient fault localization. Web Semantics: Science, Services and Agents on the World Wide Web 12, 88–103 (2012)
13. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: science, services and agents on the World Wide Web 5(2), 51–53 (2007)
14. Skaf-Molli, H., Desmontils, E., Nauer, E., Canals, G., Cordier, A., Lefevre, M., Molli, P., Toussaint, Y.: Knowledge continuous integration process (k-cip). In: Proceedings of the 21st international conference companion on World Wide Web. pp. 1075–1082. ACM (2012)
15. Stuckenschmidt, H.: Debugging OWL Ontologies-A Reality Check. In: EON (2008)
16. Vrandečić, D., Gangemi, A.: Unit tests for ontologies. In: On the Move to Meaningful Internet Systems 2006: OTM 2006 Workshops. pp. 1012–1020. Springer (2006)
17. Wang, H., Horridge, M., Rector, A., Drummond, N., Seidenberg, J.: Debugging OWL-DL ontologies: A heuristic approach. In: The Semantic Web–ISWC 2005, pp. 745–757. Springer (2005)
18. Zeller, A.: Yesterday, my program worked. Today, it does not. Why? In: Software EngineeringESEC/FSE99. pp. 253–267. Springer (1999)

Behaviour-Driven Development for Computer-Interpretable Clinical Guidelines

Reinhard Hatko[1], Stefan Mersmann[2], and Frank Puppe[1]

[1] Institute of Computer Science, University of Würzburg, Germany
[2] Dräger Medical GmbH, Lübeck, Germany

Abstract. In this paper, we propose an approach for specification and analysis of Computer-Interpretable Clinical Guidelines (CIGs) that was inspired by Behaviour-Driven Development. The expected behaviour of a CIG is specified by requirements in natural language. On the one hand, those requirements are used as design input for guideline development. On the other hand, they can be checked against time-oriented data recorded during testing sessions of the implemented CIG.

1 Introduction

Continuously improving quality and efficiency of health care delivery are major goals in medicine, especially in critical care. Approaches from process engineering that identify, organize and standardize therapeutic workflows have been established to meet these goals. Evidence-based workflows in medicine are called clinical guidelines (CGs). CGs are standardized therapeutic plans that reflect best practices for treating patients within a certain medical scope. The impact of CGs on patient outcome had been investigated and published through several clinical studies, e.g., [5]. A logical next step was the integration of standardized health care processes into medical technology by enabling CGs to be executed by medical devices as Computer-Interpretable Clinical Guidelines (CIGs).

Recently, we demonstrated the applicability of the Semantic Wiki KnowWE for the development of a CIG for automated mechanical ventilation [4]. The visual CIG language DiaFlux [3] was used to implement the guideline logic. However, the described development process lacked a specification of CIG behaviour at an abstract level on which the implementation was based.

The rest of the paper describes an approach to fill this gap: The next section introduces Behaviour-Driven Development (BDD) in Software Engineering. Section 3 shows its applicability in the area of CIGs. A case study in progress is presented in Section 4. The paper concludes with a summary in Section 5.

2 Behaviour-Driven Development

Behaviour-Driven Development (BDD) is an agile Software Engineering (SE) methodology, that conforms to the *Test-First* principle: Test cases are developed previous to the software system itself. In BDD, executable test cases are directly derived from requirements stated in natural language, by employing a

pattern matching approach. This enables stakeholders to actively participate in the definition of requirements, which are a fundamental design input for the system under development.

In contrast to other Test-First approaches (like Test-Driven Development), BDD focuses on the creation of acceptance tests rather than unit tests. The former ones assure the acceptance of the system by its users, as the system's ability to fulfill their needs with respect to a *business value* is tested. The latter ones perform testing on a lower system level by checking individual program constructs. This has the potential to reduce the number of defects at an early stage of the software development process.

```
Scenario: trader is not alerted below threshold
    Given a stock of symbol STK1 at a threshold of 10.0
    When the stock is traded at 5.0
    Then the alert status should be OFF

Scenario: trader is alerted above threshold
    Given a stock of symbol STK1 at a threshold of 10.0
    When the stock is traded at 11.0
    Then the alert status should be ON
```

Fig. 1: Two scenarios from the stock market domain, expressed in Gherkin.

BDD frameworks exist for several programming languages, e.g. JBehave[3] (Java) or RSpec [1] (Ruby). As a commonality, those frameworks offer some kind of Domain-Specific Language (DSL) for defining scenarios, and a mechanism to derive executable test code by some sort of pattern matching, usually regular expressions. One such DSL which is employed in multiple frameworks is called the *Gherkin* language [1]. Figure 1 shows an exemplary usage of Gherkin. The scenarios consist of the following three groups:

- The keyword `GIVEN` sets up the test environment by creating the specific context which is necessary to execute the test.
- Second, the functionality under test is executed (keyword `WHEN`).
- Lastly, the outcome of the previous step is compared against the expected one (keyword `THEN`).

Each group can contain several steps that are joined together using the keyword `AND`. Each step in turn, is backed by a step template (also called *support code*), that converts the text into executable program code using regular expressions, e.g. to extract method parameters. Figure 2 shows the step templates needed to execute both scenarios shown in Figure 1.

As each step is individually backed by support code, steps can arbitrarily be combined to create new acceptance tests. Thanks to the comprehensibility of Gherkin and the use of natural language, scenarios can easily be created also

[3] http://jbehave.org

```
public class TraderSteps {
    private Stock stock;

    @Given("a stock of symbol $sym at a threshold of $thr")
    public void aStock(String sym, double thr) {
        stock = new Stock(sym, thr); }

    @When("the stock is traded at $price")
    public void theStockIsTradedAt(double price) {
        stock.tradeAt(price); }

    @Then("the alert status should be $stat")
    public void theAlertStatusShouldBe(String stat) {
        ensureThat(stock.getStatus().name(), equalTo(stat)); }
}
```

Fig. 2: The according support code for the scenarios in Figure 1, expressed in Java and JBehave.

by non-programmers, e.g. the future users of the software themselves. That way, a *ubiquitous language* [2] is created, that improves communication between all participants involved in the software development process, e.g., product owner, software engineers, testers, and users.

3 Specification of Clinical Guidelines

In the remainder of this section, we describe an approach for the specification of automated CIGs following the BDD methodology. The CIG behaviour is specified as a set of scenarios expressed in the Gherkin language:

- The `GIVEN` group of a scenario describes the precondition in terms of the patient's current state, and possibly also its past progression.
- The expected therapeutic action, i.e. the output of the CIG for the given patient state, is expressed in the `WHEN` part of the scenario.
- The `THEN` group contains assumptions about the future (improved) patient state, based on the therapeutic action. As the effects will take some time until they are measurable, usually a temporal annotation is included.

There is a main difference between the SE acceptance tests as described in the previous section and the CIG scenarios. While the former ones are *actively* preparing the test context and executing the function under test, the latter ones can only *"passively"* be evaluated on data a patient delivered. The CIG scenarios are interpretable descriptions of patterns that are supposed to occur in patient data, given that the CIG has correctly been implemented based on an error-free specification.

After implementing the CIG based on the defined scenarios, test cases can be recorded, e.g. using a patient simulator [4]. The data consists of time-stamped recordings of the patient state and the according therapy over the course of

the session. Therapeutic actions are changes of the medical device's settings applied automatically according to the underlying CIG. By checking the scenarios against a test case, two types of errors can be discovered: First, the designated therapeutic action (`WHEN`-group) may not occur due to a bug in the implementation ("error of omission"), although the current patient state fulfills the precondition of the scenario (`GIVEN`-group). Second, if the precondition as well as the action of a scenario are met, but not the expected change in the patient's state, the assumptions about the state may not be correct in the first place. While the first error is most likely rooted in the implementation of the CIG, the second kind of error may arise from an improper specification itself.

4 Case Study

We have implemented an extension for the Semantic Wiki KnowWE that supports the described approach. Each step template consists of a regular expression containing capturing groups enriched by a type system, and a formal condition with the according placeholders. When using step templates, the placeholders are filled with terms from a predefined ontology, that contains the data definitions of the CIG (cf. Figure 3, left side). The ontology is created as the foundation for the fully formalized CIG [4]. Scenarios are expressed by arbitrarily combining steps. If a step can not be matched against a template, an error is reported. This leads to the definition of new templates, incrementally increasing the range of expressible scenarios as needed (cf. Figure 3, right side).

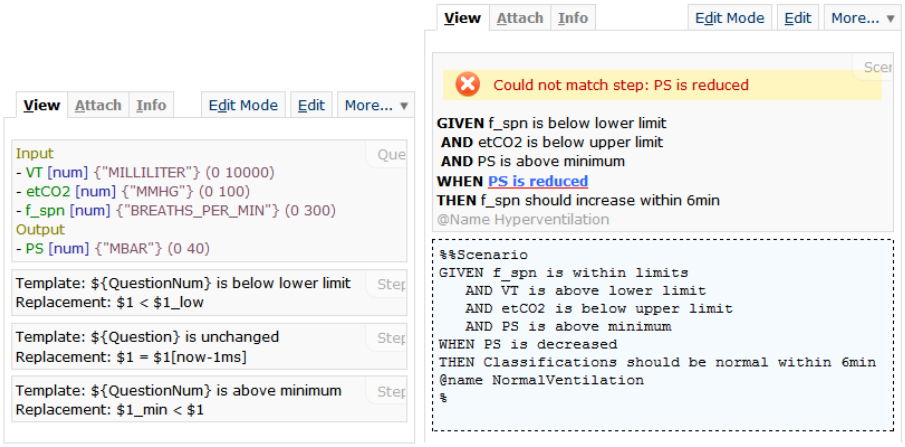


Fig. 3: Two KnowWE wiki pages: The left page contains parts of the data ontology and step templates. The right one contains two scenarios in display and in editing mode. Steps that cannot be matched result in an error.

The results of the scenario analysis with respect to a test case are visualized on an interactive timeline, that can be panned and zoomed, cf. Figure 4. Each band corresponds to exactly one scenario. At each instant of time, an icon represents the result, if the scenario's precondition is fulfilled: omitted therapeutic action (red warning sign), unfulfilled postcondition (yellow warning sign), and fully satisfied scenario (green checkmark), respectively. The timeline is integrated into KnowWE's testing framework. Replaying the test case to any instant in time is possible to introspect CIG execution for debugging purposes.

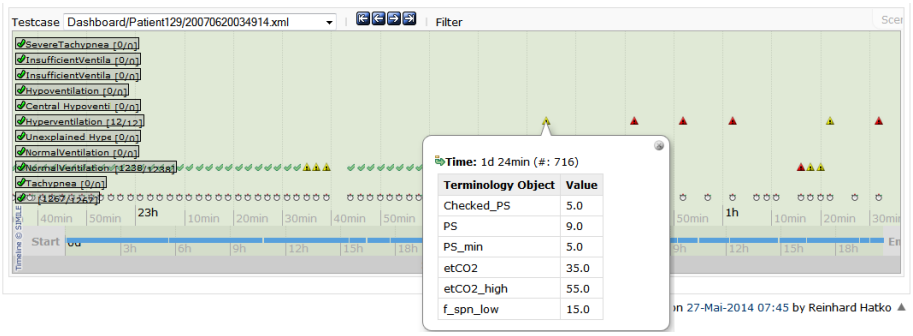


Fig. 4: Analysis results of the defined scenarios with respect to a temporal test case are depicted on an interactive timeline. For debugging purposes, the test case can be replayed until the currently selected instant of time by clicking the green arrow inside the tooltip.

For the evaluation of the presented approach, we are currently working on a case study with a real world CIG and patient data from a previous study conducted at the Department of Anesthesiology and Intensive Care Medicine, University Medical Center Schleswig-Holstein, Campus Kiel [8]. During this study, 150 patients have been automatically weaned from mechanical ventilation by the automatic weaning system SmartCare/PS® [6]. SmartCare/PS is a knowledge-based software option for Dräger's mechanical ventilators Evita XL and Evita Infinity V500. SmartCare/PS stabilizes the patient's spontaneous breathing in a respiratory comfort zone, while gradually reducing the inspiratory support of the ventilator until the patient is ready for extubation. The system's underlying CG is further depicted in [7].

This study does not investigate the usage of the derived specification as a design input. It focuses on the applicability of the approach in terms of usability and expressiveness, and its usage for analysing the test cases regarding the scenarios. The CIG behaviour has been specified using natural language scenarios as described herein. By exploiting strict naming conventions regarding the labeling of related data, e.g. measurements and their corresponding upper and lower limits, only a rather limited set of about 15 step templates needed to be de-

fined. This allows for fast requirements elicitation and also reduces maintenance efforts.

5 Conclusion

In this paper, we have described an approach inspired by Behaviour-Driven Development for specification and analysis of Computer-Interpretable Clinical Guidelines. Requirements stated by medical experts in natural language are used as design input for the development of CIGs and their analysis using test cases. We demonstrated the applicability of Behaviour-Driven Development for CIGs using an extension for the Semantic Wiki KnowWE. Currently, we are conducting a case study focusing on the analysis aspect. A real world CIG and test cases from a real patient study are used for evaluation. So far, the approach has shown its applicability in terms of usability and expressiveness. In the near future, the results of the scenarios will be analysed by medical experts.

References

1. Chelimsky, D., Astels, D., Dennis, Z., Hellesoy, A., Helmkamp, B., North, D.: The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends. The Pragmatic Programmers, United States (2011)
2. Evans, E.: Domain-Driven Design: Tackling Complexity in the Heart of Software. Addison-Wesley Longman, Amsterdam, 1 edn. (2003)
3. Hatko, R., Baumeister, J., Belli, V., Puppe, F.: DiaFlux: A graphical language for computer-interpretable guidelines. In: Riaño, D., ten Teije, A., Miksch, S. (eds.) Knowledge Representation for Health-Care, Lecture Notes in Computer Science, vol. 6924, pp. 94–107. Springer, Berlin / Heidelberg (2012)
4. Hatko, R., Schädler, D., Mersmann, S., Baumeister, J., Weiler, N., Puppe, F.: Implementing an Automated Ventilation Guideline Using the Semantic Wiki KnowWE. In: ten Teije, A., Völker, J., Handschuh, S., Stuckenschmidt, H., d'Aquin, M., Nikolov, A., Aussenac-Gilles, N., Hernandez, N. (eds.) EKAW. Lecture Notes in Computer Science, vol. 7603, pp. 363–372. Springer (2012)
5. Lellouche, F., Mancebo, J., Jolliet, P., Roeseler, J., Schortgen, F., Dojat, M., Cabello, B., Bouadma, L., Rodriguez, P., Maggiore, S., Reynaert, M., Mersmann, S., Brochard, L.: A multicenter randomized trial of computer-driven protocolized weaning from mechanical ventilation. American journal of respiratory and critical care medicine 174(8), 894–900 (Oct 2006)
6. Mersmann, S., Dojat, M.: SmartCare/PS-Automated Clinical Guidelines in Critical Care. In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 16th Eureopean Conference on AI, ECAI'2004. pp. 745–749. IOS Press 2004, Valencia, Spain (2004)
7. Neumann, A., Schmidt, H.: SmartCare®/PS The automated weaning protocol (2010), http://www.draeger.net/media/10/01/35/10013505/smartcare_auto_weaning_protocol_booklet.pdf
8. Schädler, D., Engel, C., Elke, G., Pulletz, S., Haake, N., Frerichs, I., Zick, G., Scholz, J., Weiler, N.: Automatic control of pressure support for ventilator weaning in surgical intensive care patients. American journal of respiratory and critical care medicine 185(6), 637–44 (Mar 2012)

Ensuring the Semantic Correctness of Workflow Processes: An Ontological Approach

Thi-Hoa-Hue Nguyen and Nhan Le-Thanh

WIMMICS - The I3S Laboratory - CNRS - INRIA
University of Nice Sophia Antipolis
Sophia Antipolis, France
nguyenth@i3s.unice.fr,Nhan.LE-THANH@unice.fr

Abstract. Workflow verification has been known as an important aspect of workflow management systems. Many existing approaches concentrate on ensuring the correctness of workflow processes at the syntactic level. However, these approaches are not sufficient to detect errors at the semantic level. This paper contributes to ensure the semantic correctness of workflow processes. First, we propose a formal definition of semantic constraints and an $O(n^3)$-time algorithm for detecting redundant and conflicting constraints. Second, by relying on the CPN Ontology (a representation of Coloured Petri Nets with OWL DL ontology) and sets of semantic constraints, workflow processes are semantically created. And third, we show how to check the semantic correctness of workflow processes with the SPARQL query language.

Keywords: Ensuring, Ontology, Semantic Constraint, Semantic Correctness, SPARQL query, Workflow Process

1 Introduction

Workflows have drawn an enormous amount of attention from the research community and the industry. Over the years, workflow technology has been applied extensively to the business area. So far, various researchers have focused on process specification techniques [1], [2] and on conceptual models of workflow [3], [4]. However, specifying a real-world business process is mostly manual and is thus prone to human error, resulting in a considerable number of failed projects. Therefore, model quality, correctness and re-usability become very important issues. Although numerous approaches have been already developed to ensure workflow correctness at the syntactic level (e.g., avoiding deadlocks, infinite cycles, etc.). At the semantic level there may exist errors. Recently, few researches focus on checking the semantic conformance of workflow processes.

To check the semantic correctness of a model, we consider semantic constraints as *domain specific restrictions on a business process which need to be conformed during the process is executed*. Let us take an example: In a process for the Order Management activity, when an order is accepted, an order confirmation must be sent out later; If no order confirmation is scheduled to be sent, or this activity is not done in the right position, a semantic error occurs.

The work in [5] provides a very useful inspiration for our work, but it does not discuss how to formulate semantic constraints and also does not mention about the control-flow perspective in process models. Our objective is to support workflow designers in generating semantically rich workflow processes which allow syntactic and semantic verification. In this paper, we restrict ourselves to ensure the semantic correctness of workflow processes. Our contributions are:

- Giving a formal method to describe a variety of semantic constraints;
- Proposing an algorithm to check redundant and conflicting semantic constraints;
- Developing an ontology for annotating semantic constraints and representing control flow-based business workflow processes based on that ontology;
- Showing how to use the SPARQL query language [6] to check the semantic correctness of workflow processes.

This paper is organized as follows: in Section 2, we discuss related work. A short introduction to the CPN Ontology, which is defined to represent Coloured Petri Nets (CPNs) with OWL DL, is given in Section 3. Section 4 proposes a formal definition of semantic constraints for business processes. An algorithm used for checking redundant and conflicting semantic constraints is presented. We then develop a semantic conformance-oriented ontology. In Section 5, we present the creation of correspondences between those two ontologies to develop workflow processes. These workflows enable designers to check their semantic correctness. Five semantic verification issues of a workflow process are introduced in Section 6. Finally, Section 7 concludes the paper with an outlook on the future research.

2 Related Work

Today, software systems that automate business processes have become more and more advanced. Various researchers have paid attention to the problem of ensuring the correctness of process models. Many methods have been done in workflow verification. Most of them focus on the control-flow perspective, such as [7], [8], [9], [10], [11] to prevent errors (e.g., avoiding deadlocks, infinite cycles) at the syntactic level. Nevertheless, they check mainly the conformance of a workflow process based on the principle that if the constraints on data and control flow are met during execution, the workflow is correct.

Recently, a number of researches have grown beyond the scope of pure control-flow verification. Some approaches analyze the final state of the whole workflow [12] or consider running processes [13], [14], [15]. In [13], semantic constraints are defined over processes which are used to detect semantic conflicts caused by violation only of dependency and mutual exclusion constraints. They presented techniques to ensure semantic correctness for single and concurrent changes at process instance level. With regard to ontological approaches, aspects of semantic correctness are considered in few researches, such as [5], [16], [17]. The approach of [17] requires both the annotation of preconditions and effects to

ensure the models are semantically correct. In [5], the SPARQL query language is used to check the semantic correctness of ontology-based process representations. Individual model elements were annotated with concepts of a formal ontology, and constraints were formalized as SPARQL queries. However, these approaches depend on the availability of an ontology and a bulky annotation of process models.

We know that the ontology-based approach for modelling business process is not a new idea. There are some works made efforts to build business workflow ontologies, such as [18], [4], [19] to support (semi-)automatic system collaboration, provide machine-readable definitions of concepts and interpretable format. However, the issue that relates to workflow verification at the syntactic level is not mentioned. By extending the state-of-the-art, we use the Web Ontology Language to define the CPN Ontology for representing CPNs with OWL DL ontology and the semantic conformance-oriented ontology. Our ontological approach enables to create high quality and semantically rich workflow processes.

3 Representation of Coloured Petri Nets with OWL DL ontology

In this Section, we introduce the CPN Ontology [20] defined for business processes modelled with CPNs. The purpose is to ensure the syntactic correctness of workflow processes and to facilitate business process models easy to be shared and reused.

On one hand, Coloured Petri Nets (CPNs) [21] have been developed into a full-fledged language for the design, specification, simulation, validation and implementation of large software systems. Consequently, modelling business processes with CPNs supports workflow designers easy to verify the syntactic correctness of workflow processes [8]. On the other hand, **OWL DL** [22], which stands for OWL Description Logic, is equivalent to Description Logic $\mathcal{SHOIN}(\mathcal{D})$. OWL DL supports all OWL language constructs with restrictions (e.g., type separation) and provides maximum expressiveness while keeping always computational completeness and decidability. Therefore, we choose OWL DL language to represent the CPN Ontology. We believe that the combination of CPNs and OWL DL provides not only semantically rich business process definitions but also machine-processable ones. Figure 1 depicts the core concepts of the CPN ontology.

The CPN Ontology comprises the concepts: **CPNOnt** defined for all possible CPNs; **Place** defined for all places; **Transition** defined for all transitions; **InputArc** defined for all directed arcs from places to transitions; **OutputArc** defined for all directed arcs from transitions to places; **Token** defined for all tokens inside places (We consider the case of one place containing no more than one token at one time); **GuardFunction** defined for all transition expressions; **CtrlNode** defined for occurrence condition in control nodes; **ActNode** defined for occurrence activity in activity nodes, **Delete** and **Insert** defined for all expressions in input arcs and output arcs, respectively; **Attribute** defined for all

$$
\begin{aligned}
&CPNOnt \equiv \geq 1 hasTrans.Transition \sqcap \geq 1 hasPlace.Place \\
&\quad \sqcap \geq 1 hasArc.(InputArc \sqcup OutputArc) \\
&Place \equiv connectsTrans.Transition \sqcap \leq 1\ hasMarking.Token \\
&Transition \equiv connectsPlace.Place \sqcap = 1 hasGuardFunction.GuardFunction \\
&InputArc \equiv \geq 1 hasExpresion.Delete \sqcap \exists hasPlace.Place \\
&OutputArc \equiv \geq 1 hasExpression.Insert \sqcap \exists hasTrans.Transition \\
&Delete \equiv \forall hasAttribute.Attribute \\
&Insert \equiv \exists hasAttribute.Attribute \\
&GuardFunction \equiv \geq 1 hasAttribute.Attribute \sqcap = 1 hasActivity.ActNode \\
&\quad \sqcup = 1 hasControl.CtrlNode \\
&Token \equiv \geq 1 hasAttribute.Attribute \\
&Attribute \equiv \geq 1 valueAtt.Value \\
&CtrlNode \equiv \leq 1 valueAtt.Value \\
&ActNode \equiv = 1 valueAtt.Value \\
&Value \equiv valueRef.Value
\end{aligned}
$$

Fig. 1: CPN ontology expressed in a description logic

attributes of individuals); **Value** defined for all subsets of $I_1 \times I_2 \times \ldots \times I_n$ where I_i is a set of individuals.

Properties between the concepts in the CPN Ontology are also specified in Figure 1. For example, the concept $CPNOnt$ is defined with three properties, including $hasPlace$, $hasTrans$ and $hasArc$. It can be glossed as 'The class $CPNOnt$ is defined as the intersection of: (i) any class having at least one property $hasPlace$ whose value is restricted to the class $Place$ and; (ii) any class having at least one property $hasTransition$ whose value is restricted to the class $Transition$ and; (iii) any class having at least one property $hasArc$ whose value is either restricted to the class $InputArc$ or the class $OutputArc$'.

4 Semantic Constraints for Business Processes

As mentioned previously, our work aims at representing workflow processes modelled with CPNs in a knowledge base. Therefore, in this Section, we focus on ensuring their quality by guaranteeing their semantic correctness.

4.1 Definition of Semantic Constraints

By taking account domain experts in support of modellers at build time, a set of semantic constraints is specified, which then is used to develop a corresponding workflow. According to [13], there are two fundamental kinds of semantic constraints, including mutual exclusion constraints and dependency constraints. For interdependent tasks, e.g., the presence of task A indicates that task B must be included, however, task B can be executed while task A is absence. In fact, there may exist tasks that are coexistent. This refers to the coexistence constraints. Consequently, we propose three basic types: mutual exclusion constraints, dependency constraints and coexistence constraints.

Definition 1 (Semantic Constraint).

Let T be a set of tasks. A semantic constraint:

$c = (constraintType, appliedTask, relatedTask, order, description, [Equivalence])$ *where:*

- $constraintType \in \{mExclusion, dependency, coexistence\}$;
- $appliedTask \in T$;
- $relatedTask \in T$;
- $order \in \{before, after, concurrence, notSpecified\}$;
- *description is an annotation of the constraint;*
- *Equivalence is a set of tasks which are equivalent to task appliedTask.*

In Definition 1, the first parameter *constraintType* denotes the type of a semantic constraint. Each value of *constraintType* refers to the relationship between the executions of the source task denoted by the second parameter *appliedTask* and the target task denoted by the third parameter *relatedTask*. Parameter *order* specifies the order between the source and target tasks in a process model. The first four parameters are very important when defining a semantic constraint. The fifth parameter, *description*, is used for describing the constraint. *Equivalence* is an optional parameter, which contains a set of tasks (if any) being equivalent to the source task.

Let us continue the example of a process for the Order Management activity. The process is determined as follows: After receiving an order, two tasks have to do in parallel are *authenticate client* and *check availability*. If both of these tasks result "true", the order is accepted and an order confirmation is sent out. Otherwise, an order refusal is sent out. Some semantic constraints of the process are formed as follows:

```
c1 = (dependency, authenticate client, receive request, before,
receiving an order has to be performed before authenticating
client, {authenticate purchaser});
c2 = (dependency, check availability, receive request, before,
receiving an order has to be performed before checking
availability);
c3 = (coexistence, authenticate client, check availability,
concurrence, client authentication and checking availability
are performed in parallel);
c4 = (dependency, evaluate results, authenticate client, before,
evaluating the results obtained from the relevant departments);
c5 = (dependency, evaluate results, receive request, before,
receiving an order has to be performed before evaluating results
related to the order)
```

4.2 Checking implicit, redundant and conflicting semantic constraints

A workflow process is designed based upon the specified semantic constraints. However, when defining those constraints, there may occur implicit, redundant or conflicting semantic constraints.

Note that a combination of two or more constraints can constitute some new constraints. This is demonstrated by the *order* parameter in Definition 1. As mentioned above, this parameter indicates the execution order of a source task and a target task. Consider T_1, T_2, T_3, instances of tasks, we identify the following properties which are associative, symmetric, transitive and/or commutative presented in Table 1.

Table 1: Algebra properties of the order parameter

Name		Expression
Association	(1)	$(T_1\ concurrence\ T_2)\ concurrence\ T_3 = T_1\ concurrence\ (T_2\ concurrence\ T_3)$
	(2)	$(T_1\ notSpecified\ T_2)\ notSpecified\ T_3 = T_1\ notSpecified\ (T_2\ notSpecified\ T_3)$
Symmetrization	(1)	$T_1\ before\ T_2 = T_2\ after\ T_1$
Transitivity	(1)	$T_1\ before\ T_2, T_2\ before\ T_3 \rightarrow T_1\ before\ T_3$
	(2)	$T_1\ after\ T_2, T_2\ after\ T_3 \rightarrow T_1\ after\ T_3$
	(3)	$T_1\ concurrence\ T_2, T_2\ concurrence\ T_3 \rightarrow T_1\ concurrence\ T_3$
	(4)	$T_1\ concurrence\ T_2, T_1\ before\ T_3 \rightarrow T_2\ before\ T_3$
	(5)	$T_1\ concurrence\ T_2, T_1\ after\ T_3 \rightarrow T_2\ after\ T_3$
Commutativity	(1)	$T_1\ concurrence\ T_2 = T_2\ concurrence\ T_1$
	(2)	$T_1\ notSpecified\ T_2 = T_2\ notSpecified\ T_1$

These properties are used for inferring implicit semantic constraints. As a result, detecting implicit constraints plays a crucial role in the elimination of redundant constraints. In addition, conflicting constraints will lead to undesirable results. Therefore, it is necessary to resolve them before they can be applied.

Let us consider three constraints presented in Subsection 4.1, including $c1, c4$ and $c5$. According to transitivity property (1) in Table 1, a new constraint, named $c1\_4$, can be inferred from constraints $c1$ and $c4$, where:

```
c1_4={dependency, evaluate results, receive request, before,
receiving an order has to be performed before authenticating
client and evaluating the results obtained from the relevant
departments}
```

Comparing constraint $c1\_4$ to constraint $c5$, their first four values are the same, hence constraint $c5$ is redundant. As a result, constraint $c5$ has to be removed.

Because of the different complexity of each semantic constraint set, we need an algorithm to resolve issues related to redundancy and conflict semantic constraints. The procedure for validating the constraint set will stop as soon as it

detects a couple of conflicting constraints and a message is generated to notify the user. In addition, if there exist any redundant constraints, they will be removed. In our algorithm in Figure 2, the boolean function $conflict$ is used for checking the conflict between two constraints, e.g., it returns $true$ if they are conflicting, otherwise, it returns $false$. The function $infer$ is used for inferring implicit constraints. The time complexity of the algorithm is $O(n^3)$ where n is the number of semantic constraints. To provide a representation of semantic

```
Input: Initial semantic set vector C
Output: Sound semantic constraint set vector C
SCValidation (C: Semantic_Constrait_Set)
1:  { n = C.size;
2:  For (i = 1; i<=n-1; i++)
3:   For (j = i+1; j<=n; j++)
4:     If (conflict(C[i],C[j]))
5:        { Print ‘‘Constraint C[i] conflicts with constraint C[j]’’;
6:          Break ;   }
7:     Else If (!empty(infer(C[i],C[j])))
8:         //If there exists an implicit constraint
9:        { cij = infer(C[i],C[j]);
10:         For (k = j+1; k<=n; k++)
11:           If (conflict(cij,C[k]))
12:         { Print ‘‘The implicit constraint inferred from C[i] and C[j]
13:           conflicts with constraint C[k]’’ ;
14:         Break ;  }
15:       Else If (compare(cij,C[k]))
16:        {  C.Remove(C[k]) ; //Remove redundant constraint C[k]}
18: }
```

Fig. 2: Algorithm for validating the semantic constraint set

constraints related to process elements, in next Subsection, we will describe an approach for constructing a semantic conformance-oriented ontology.

4.3 Development of a Semantic conformance-oriented Ontology

Our work aims at representing processes modelled with CPNs in a knowledge base. Therefore, to provide a representation of semantic constraints related to process elements, we develop an approach for constructing a new ontology. This ontology is oriented to semantic conformity checking in workflow processes. We focus on formalizing the concepts/relations corresponding to the knowledge that is required by model elements.

The following keystones to transform a set of semantic constraints into an OWL DL ontology:

- Each semantic constraint c is mapped to an instance of $owl : Class$.

- *appliedTask* and *relatedTask* are mapped to two instances of $owl : Class$. The $rdfs : subClassOf$ property is used to state that these classes is a subclass of the constraint class.
- Each value of *constraitType* or *order* is defined as an instance of the built-in OWL class $owl : ObjectProperty$.
- *description* is defined as an instance of the built-in OWL class $owl : Datatype\ Property$;
- Each value in the set *Equivalence* is mapped to an instance of $owl : Class$. The built-in property $owl : equivalentClass$ is used to link every class description of these classes to the class description of *appliedTask*.

In the upcoming Section, we will discuss about the integration of a semantic conformance-oriented ontology (domain knowledge) and the CPN Ontology to create workflow processes.

5 Creation of Correspondences Between Ontologies

We rely on ontology mapping techniques for matching semantics between ontologies, i.e., the CPN Ontology and Domain Ontology (a semantic conformance-oriented ontology). In our case, the articulation of two ontologies are used not only for creating semantically workflow processes, but also for verifying their correctness.

We now define our use of the term "mapping": Consider two ontologies, O_1 and O_2. Mapping of one ontology with another is defined as bringing ontologies into mutual agreement in order to make them consistent and coherent. It means that for a concept or a relation in ontology O_1, we try to find the same intended meaning in ontology O_2; For an instance in ontology O_1, we find the same instance in ontology O_2.

Definition 2 (Mapping related to the *before* property).
We define a mapping for all instances I_C of a semantic constraint in which the order between the instance of class appliedTask, named $task_a$, and the instance of class relatedTask, named $task_b$, is indicated by the object property before. The type of instance I_C is either dependency or coexistence. A set of correspondences is determined as follows:

- *Each instance of class appliedTask or relatedTask is mapped into an instance of class Transition (expressing activity node).*
- *There exists a firing sequence $t_1 t_2 \dots t_n$, where t_1, t_n are the instances of class Transition corresponding to instances $task_a$ and t_b respectively, $t_a = t_1$, $t_b = t_n$ $n \geq 2$.*

Definition 3 (Mapping related to the *concurrence* property).
We define a mapping for all instances I_C of a semantic constraint in which the order between the instance of class appliedTask, named $task_a$, and the instance of class relatedTask, named $task_b$, is indicated by the object property concurrence. The type of instance I_C is coexistence. A set of correspondences is determined as follows:

- *Each instance of class appliedTask or relatedTask is mapped into an instance of class Transition (expressing activity node).*
- *Two instances of class transitions which correspond to instance* $task_a$ *and instance* $task_b$ *can be enabled at the same time.*

It is important to underline that object property $before$ is the symmetrical property of object property $after$. Consequently, we do not define a mapping related to the $after$ property.

By continuing the process schema for the order management activity in Section 4, Figure 3 shows the mapping of some instances between two onotologies, CPN Ontology and Semantic Conformance-oriented Ontology.

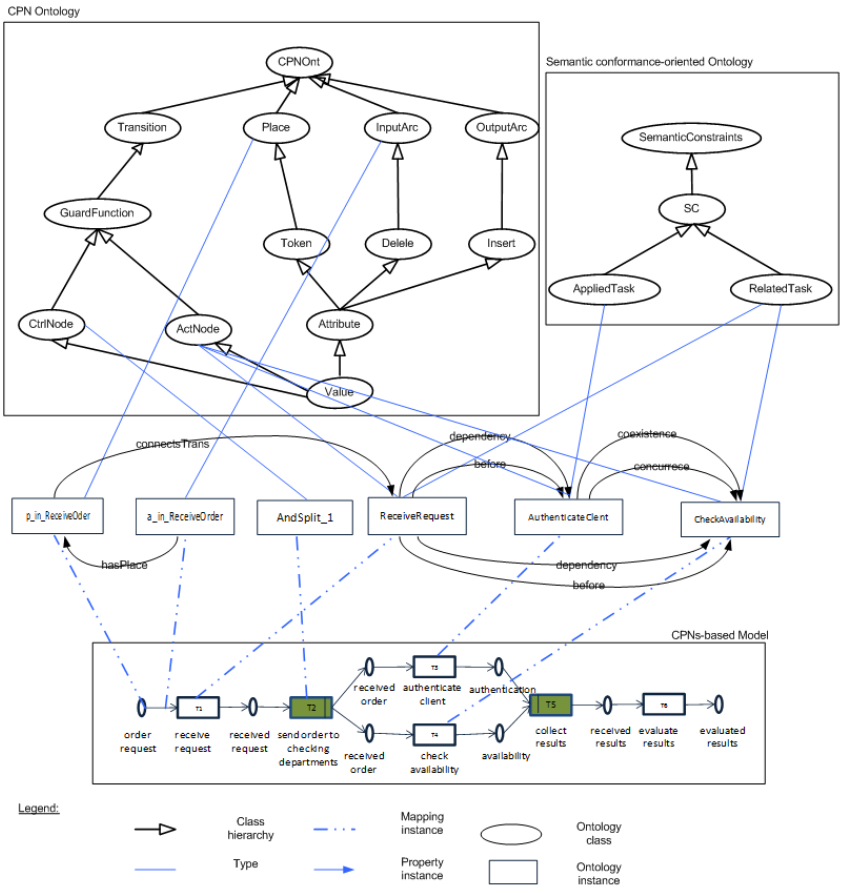


Fig. 3: An example of ontology mapping (excerpt)

We have introduced the formal definition of semantic constraints and illustrated how to model a workflow process with CPNs based on specified semantic

constraints. Note that concrete workflow processes are represented in RDF syntax. Moreover, to develop or modify a workflow process, manipulation operations [20] (e.g., inserting a new element) are required. Therefore, it is necessary to verify workflow processes at design time before they are put into use.

6 Semantic Verification Issues

By using the algorithm presented in Section 4, the sets of specified semantic constraints are checked redundant and conflicting. Hence, we here pay attention to the research question relating to semantic verification: Is the behavior of the individual activities satisfied and conformed with the control flow? To answer this question, we address the following semantic verification issues:

- Are there activities whose occurrences are mutual exclusion, but that may be executed in parallel or in sequence?
- Are there activities whose executions are interdependent, but that may be carried out in choice or in parallel?
- Are there activities whose occurrences are coexistent, but that may be executed in choice?
- Are there any couples of activities whose order executions are defined as one before the other, but that may be executed in the opposite?
- Are there any couples of activities whose order executions are defined as one after the other, but that may be executed in the opposite order?

Because concrete workflows are stored in RDF syntax, we rely on the CORESE [23] semantic search engine for answering SPARQL queries asked against an RDF knowledge base. We initiate SPARQL queries to verify whether workflow processes contain semantic errors or not. SELECT query form is chosen for this work. After a SELECT keyword, the variables are listed that contain the return values. And in the WHERE clause, one or more graph patterns can be specified to describe the desired result.

The following query[1] relating to the third verification issue is used to query if the model contains 'any pairs of activities whose occurrences are coexistence but that may be executed in choice'. The properties $h : coexistence$ and $h : concurrence$ defined in the first ontology indicate the semantic constraint between activities $?t1$ and $?t2$. On the other hand, the other properties defined in the second ontology which represent these activities restricted to the control flow perspective. By applying this query to the workflow example depicted in Figure 3, the result is empty.

The sample query does not only demonstrate that the SPARQL query language is able to check the semantic correctness of workflow processes, but also the usage of terminological background knowledge provided by the semantic conformance-oriented ontology and CPN Ontology.

[1] The prefix is assumed as:
$PREFIX h :< http://www.semanticweb.org/CPNWF\# >$

Moreover, by representing CPNs-based business processes with OWL DL ontology we can also verify the soundness of models. This means that we can check syntactic errors (for example, deadlocks, infinite cycles and missing synchronization, etc.) by the SPARQL query language.

```
SELECT ?t1 ?t2 WHERE
{
?t1 rdf:type h:Transition
?t2 rdf:type h:Transition
?t3 rdf:type h:Xor-split
?t4 rdf:type h:Xor-join
?t1 h:coexistence ?t2
?t2 h:concurrence ?t1
?t3 h:connectsPlace/h:connectsTrans ?t1
?t3 h:connectsPlace/h:connectsTrans ?t2
?t1 h:connectsPlace/h:connectsTrans ?t4
?t2 h:connectsPlace/h:connectsTrans ?t4
FILTER (?t1!=?t2)
}
```

7 Conclusion

This paper presents a formal method for describing semantic constraints that are used to generate workflow processes. First, we propose a formal definition of semantic constraints. Then we describe an algorithm for detecting redundant and conflicting semantic constraints. To integrate the domain knowledge used for annotating the process elements, we develop a semantic conformance-oriented ontology. This ontology is then matched with the CPN Ontology (a representation of CPNs with OWL DL). The mapping is to enable workflow processes which can be verified at the semantic level and also syntactic level. Furthermore, we demonstrate that the SPARQL query language is able to check the correctness of concrete workflow processes represented in RDF syntax. This ensures error-free workflow processes at build-time.

We know that verifying workflow processes at build-time is not enough to guarantee workflows can be executed correctly. The correctness of workflow execution must also be checked. Therefore, in future work, we plan to develop a run-time environment for validating concrete workflows.

References

1. Ellis, C.A., Nutt, G.J.: Modeling and enactment of workflow systems. In: Application and Theory of Petri Nets. (1993) 1–16
2. van der Aalst, W.M.P.: The application of petri nets to workflow management. Journal of Circuits, Systems, and Computers **8**(1) (1998) 21–66

3. Barros, A.P., ter Hofstede, A.H.M., Proper, H.A.: Essential principles for workflow modelling effectiveness. In: PACIS. (1997) 15
4. Koschmider, A., Oberweis, A.: Ontology based business process description. In: EMOI-INTEROP, Springer (2005) 321–333
5. Fellmann, M., Thomas, O., Busch, B.: A query-driven approach for checking the semantic correctness of ontology-based process representations. In: BIS. (2011) 62–73
6. W3C: Sparql 1.1 query language. http://www.w3.org/TR/sparql11-query/ (March 2013) W3C Recommendation.
7. van der Aalst, W.M.P.: Verification of workflow nets. In: ICATPN. (1997) 407–426
8. Verbeek, H., Basten, T., van der Aalst, W.: Diagnosing workflow processes using woflan. The computer journal **44** (1999) 246–279
9. Bi, H.H., Zhao, J.L.: Applying propositional logic to workflow verification. Information Technology and Management **5**(3-4) (2004) 293–318
10. Wainer, J.: Logic representation of processes in work activity coordination. In: Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 1. SAC '00, New York, NY, USA, ACM (2000) 203–209
11. Sadiq, W., Maria, Orlowska, E.: Analyzing process models using graph reduction techniques. Information Systems **25** (2000) 117–134
12. Lu, S., Bernstein, A.J., Lewis, P.M.: Automatic workflow verification and generation. Theor. Comput. Sci. **353**(1-3) (2006) 71–92
13. Ly, L.T., Rinderle, S., Dadam, P.: Integration and verification of semantic constraints in adaptive process management systems. Data Knowl. Eng. **64**(1) (2008) 3–23
14. Kumar, A., Yao, W., Chu, C.H., Li, Z.: Ensuring compliance with semantic constraints in process adaptation with rule-based event processing. In: RuleML. (2010) 50–65
15. Ly, L.T., Rinderle-Ma, S., Göser, K., Dadam, P.: On enabling integrated process compliance with semantic constraints in process management systems - requirements, challenges, solutions. Information Systems Frontiers **14**(2) (2012) 195–219
16. Thomas, O., Fellmann, M.: Semantic process modeling - design and implementation of an ontology-based representation of business processes. Business & Information Systems Engineering **1**(6) (2009) 438–451
17. Weber, I., Hoffmann, J., Mendling, J.: Beyond soundness: on the verification of semantic business process models. Distributed and Parallel Databases **27**(3) (2010) 271–343
18. Gasevic, D., Devedzic, V.: Interoperable petri net models via ontology. Int. J. Web Eng. Technol. **3**(4) (2007) 374–396
19. Sebastian, A., Tudorache, T., Noy, N.F., Musen, M.A.: Customizable workflow support for collaborative ontology development. In: 4th International Workshop on Semantic Web Enabled Software Engineering (SWESE) at ISWC 2008. (2008)
20. Nguyen, T.H.H., Le-Thanh, N.: An ontology-enabled approach for modelling business processes. In: Beyond Databases, Architectures, and Structures. Volume 424 of Communications in Computer and Information Science. Springer International Publishing (2014) 139–147
21. Kristensen, L.M., Christensen, S., Jensen, K.: The practitioner's guide to coloured petri nets. STTT **2**(2) (1998) 98–132
22. W3C: Owl web ontology language reference. http://www.w3.org/TR/owl-ref/ (2004) W3C Recommendation.
23. Corby, O., et al.: Corese/kgram. https://wimmics.inria.fr/corese

Integration of Activity Modeller with Bayesian network based recommender for business processes*

Szymon Bobek , Grzegorz J. Nalepa, Olgierd Grodzki

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

Abstract Formalized process models help to handle, design and store processes in a form understandable for the designers and users. As model repositories often contain similar or related models, they should be used when modelling new processes in a form of automated recommendations. It is important, as designers prefer to receive and use suggestions during the modelling process. Recommendations make modelling faster and less error-prone because a set of good models is automatically used to help the designer. In this paper, we describe and evaluate a method that uses Bayesian Networks and configurable models for recommendation purposes in process modelling. The practical integration of the recommendation module with a Activity Modeller tool is also presented.

1 Introduction

Processes are one of the most popular methods for modelling flow of information and/or control within a sequence of activities, actions or tasks. Among many notations that allow to define and build business process diagrams, the Business Process Modeling Notation (BPMN) [1] is currently considered as a standard. BPMN is a set of graphical elements denoting such constructs as activities, splits and joins, events etc. These elements can be connected using control flow and provide a visual description of process logic [2]. Thus, a visual model is easier to understand than textual description and helps to manage software complexity [3].

Several visual editors were developed to support design of business processes in BPMN, one of which is Activity Modeller [1]. It is a web modeller component that is available as part of the Activiti Explorer web application. The Modeller is a fork of the Signavio Core Components project [2]. The goal of the Activiti Modeller is to support all the BPMN elements and extension supported by the Activiti Engine – a Java process engine that runs BPMN 2 processes natively.

Although visual editors like Activity provide support for building and executing business processes, this support does not include design recommendations. By recommendation we mean suggestions that the system can give to the designer to improve the design process both in terms of quality and time.

* The paper is supported by the *Prosecco* project.

[1] See http://activiti.org/

[2] See http://www.signavio.com/

Three different types of recommendations can be distinguished depending on the subject of recommendation process [4]. These types are:

- structural recommendations – that allows to suggest structural elements of the BP diagram, like tasks, events, etc,
- textual recommendations –that are used to allow suggestions of names of elements, guard conditions, etc.
- attachment recommendations –that allows to recommend attachments to the BP in a form of decision tables, links, service tasks, etc.

In this paper we focus on structural recommendation, that allows for automated generation of suggestions for the next (forward recommendation), previous (backward recommendation) or missing elements (autocompletion) of the designed BP. Such recommendations improves time needed to build new business process and prevents user from making most common mistakes. What is more, such suggestions allow the designer to interactively learn best practices in designing BPMN diagrams as this practices are encoded into the recommendation model.

In this paper we present the implementation and evaluation of the method for structural recommendation of business processes that uses Bayesian networks and configurable processes. The work presented in this paper is part of the Prosecco project[3]. The objective of the project is to provide tools supporting the management of Small and Medium Enterprises (SMEs) by the introduction of methods for declarative specification of business process models and their semantics. The work described in this article is a continuation of our previous research presented in [5].

The rest of the paper is organized as follows. In Section 2 related work is presented and motivation for our research was stated. Section 3 describes briefly the recommendation method developed. A prototype implementation of the recommendation module, and its integration with Activity Modeller in Section 4. This section provides also an evaluation of the method on a real-case scenario. Section 5 provides summary of the research and open issues that are planned to be solved in a future work.

2 Related work and motivation

As empirical studies have proven that users prefer to receive and use suggestions during modelling processes [6], several approaches to recommendations in BP modelling have been developed. They are based on different factors such as labels of elements, current progress of modelling process, or additional pieces of information like process descriptions or annotations.

Among attachment recommendations, support with finding appropriate services was proposed by Born et al. [7] and Nguyen et al. [8]. Such a recommendation mechanism can take advantage of context specified by the process fragment [8] or historical data [9]. Approaches that recommend textual pieces of information, such as names of tasks, were proposed by Leopold et al. [10] and extended in [11].

In the case of structural recommendations, Kopp et al. [12] showed how to autocomplete BPMN fragments in order to enable its verification. Although this approach

[3] See `http://prosecco.agh.edu.pl`

does not require any additional information, it is very limited in the case of recommendations. The more useful existing algorithms are based on graph grammars for process models [13,14], process descriptions [15], automatic tagging mechanism [16,6], annotations of process tasks [17] or context matching [18].

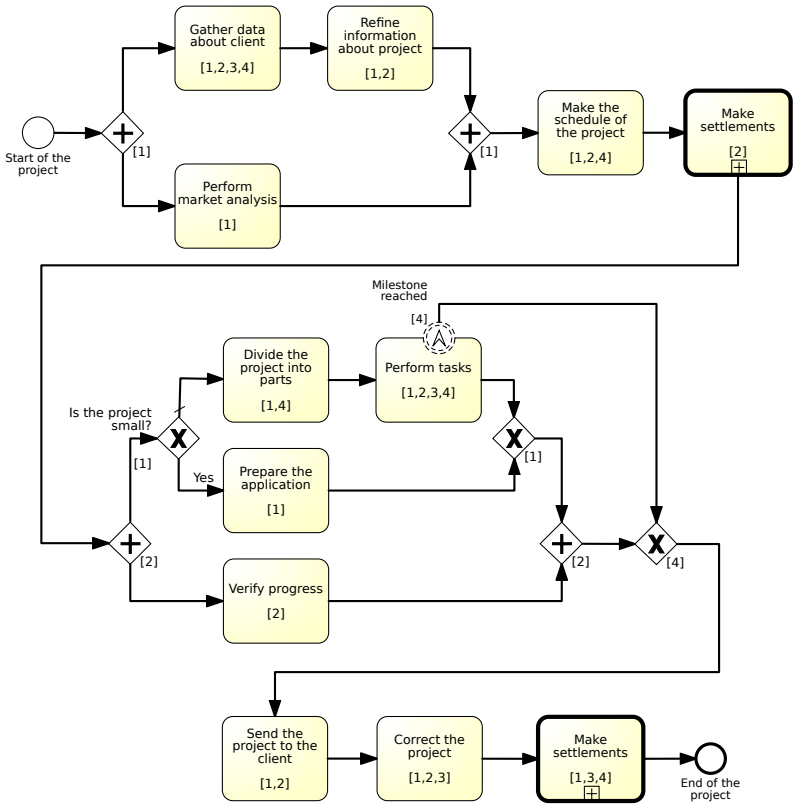


Figure 1. Configurable Business Process [5]

Case-based reasoning for workflow adaptation was discussed in [19]. It allows for structural adaptations of workflow instances at build time or at run time, and supports the designer in performing such adaptations by an automated method based on the adaptation episodes from the past.

The work presented in this paper is a continuation of our previous research presented in [5]. We use Bayesian networks (BN) for recommendation purposes. In this approach a BN is created and learned based on a configurable business process. The motivation for the current work was to evaluate the methods developed in previous research. Therefore this paper focuses on the issues of matching Bayesian network to business processes to allow probabilistic recommendation queries. The BN learning was presented in our previous work and is beyond the scope of this paper. For the eval-

uation environment we decided to use Activity Modeller [4], which is part of one of the most widely used software bundle for designing and executing BPMN models.

In the following section we present a short overview of the recommendation method, that uses BN for structural recommendations. It also describes an algorithm for mapping BPMN process elements to random variables of Bayesian network.

3 Bayesian network based recommendations

Bayesian Network [20] is an acyclic graph that represents dependencies between random variables and provide graphical representation of the probabilistic model. This representation serves as the basis for compactly encoding a complex probability distribution over a high-dimensional space [21].

In the subject of structural recommendations with BN approach, the random variables are BP structural elements (tasks, events, gates, etc). Connections between the random variables are obtained from the configurable process, that captures similarities between two or more BP models and encapsulates them within one meta-model. For the configuration model example, see Figure 1.

The transformation from a configurable model to a BN model is straightforward. Each node in a configurable process has to be modeled as a random variable in BN. Therefore, each node in a configurable process is translated into a node in the network. The flow that is modeled by configurable process represents dependencies between nodes. These dependencies also can be translated directly to the BN model (See Figure 2).

The BN network obtained from the configurable model encodes just the structure of the process. To allow querying the network for the recommendations it is necessary to train it. The comprehensive list and comparison of them can be found in [22]. For the purpose of this paper, we use the Expectation Maximization algorithm to perform Bayesian network training. The software we used to model and train our network is called Samiam[5]. The training data was a configurable process serialized to a CSV file. Each column in the file represents a node in configurable process, whereas each row represents a separate process model that was used to create the former.

3.1 Querying the model for recommendations

We defined three different structural recommendation modes that include forward recommendations, backward recommendation and autocompletion [5]. So far we successfully implemented and evaluated forward recommendations that allows for automated generation of suggestions for the next element of the designed BP. Although the backward recommendations and autocompletion scenarios are not presented in the paper, the overall algorithm remains the same for all three scenarios. The difference between them lays on the implementation rather than conceptual level, and therefore they were skipped for the sake of clarity and simplicity. In this section we describe details of the aforementioned forward recommendation algorithm.

[4] http://activiti.org

[5] See: `http://reasoning.cs.ucla.edu/samiam`.

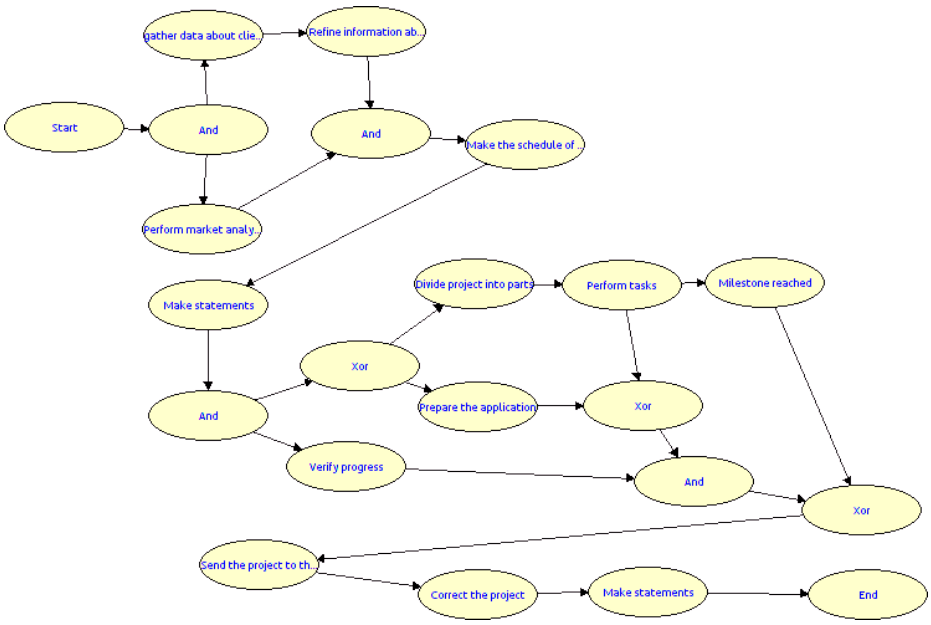


Figure 2. Bayesian Network representing the configurable process from Figure 1

Figure 3 describes a possible query for forward recommendation. The red circle denotes observed states (so called evidence) that represents BPMN elements that were already included by the designed in the model. In the case presented in the Figure 3, the only observed evidence is a *Start* element. The remaining circles denotes possible configuration of BPMN blocks with probabilities assigned to them. For instance probability, that the block *Perform market analysis* will be present in the model is 25%.

The forward recommendation algorithm will scan the Bayesian network starting from the last observed block in a topological order, and return three blocks with the probabilities of presence in the model greater than 50%. The most challenging task in this algorithm was mapping the nodes from BPMN process to nodes in Bayesian network. This was particularly difficult because the BPMN elements are identified by the unique IDs that are different every time a new process is created.

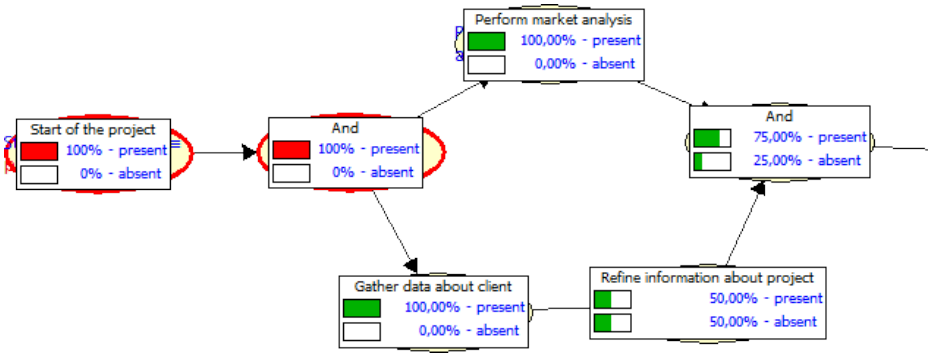


Figure 3. Forward recommendations of the process for the model presented in Fig. 2

Therefore, we distinguished several possible paths for matching the BN model to the process that is designed:

- graph-based metrics, that allows to compare structures of two networks and identify areas that may correspond to the same elements [23],
- text-based metrics, that allows to compare elements based on their labels [24], and
- semantic-based comparisons, that provides more advanced matching based on the elements labels, taking into consideration semantics of the labels [25].

Because the recommendation module should work in a real-time, we decided to use the second approach which is more efficient comparing to graph-based approaches and requires less implementation effort than the third option. This choice was motivated also by the fact that BPMN elements usually have very informative labels and hence, the text comparison should give good results. As the metric for comparing nodes labels, a Levenshtein distance [26] was used. Mathematically, the Levenshtein distance between two strings a, b is given by the equation 1, where where $1_{(a_i \neq b_j)}$ is the indicator function equal to 0 when $a_i = b_j$ and equal to 1 otherwise.

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases} \tag{1}$$

Such text-based matching performs well until two or more nodes have similar or the same labels. For instance in Figure 1 there are four *And* nodes and two *Make settlements* nodes. Hence, when the user puts the block with a label And, the recommendation algorithm has to decide to which of the blocks in the Bayesian network it corresponds. This is performed by the neighborhood scanning algorithm.

The algorithm performs a breadth-first search on the currently designed model, and try to match the neighborhood of the node from BPMN diagram to the neighborhoods of the ambiguous nodes in the Bayesian networks. The node from the Bayesian network which neighborhood matches the most of the nodes from BPMN diagram is chosen. For instance in the example from Figure 3, the user choses to include the *And* gateway in the diagram. The recommendation algorithm has to decide which *And* node from the Bayesian network presented in the Figure 2 should be treated as a reference point for the next recommendation. Because in the BPMN diagram the neighborhood of the *And* node is just one element called *Start*, the neighborhood scanning algorithm will search in the BN for the *And* node with a *Start* element as a neighbor. If the ambiguity cannot be resolved by the first level neighborhood scanning, the algorithm continues the process in a breath-search manner.

The following section presents details of the implementation of the recommendation module and presents brief evaluation of the approach.

4 Activity recommender

In [5] the process of transforming a configurable process into a Bayesian Network is performed manually. In order to automate this process 3 auxiliary modules have been implemented.

The first module is the converter, which creates a Bayesian Network file from a BPMN file. The second module generates a training data file based on the information about each block's occurrence in each of the processes that the configurable process is composed of. The third module takes the untrained Bayesian Network file and the training data file as input and trains the network using the EM algorithm.

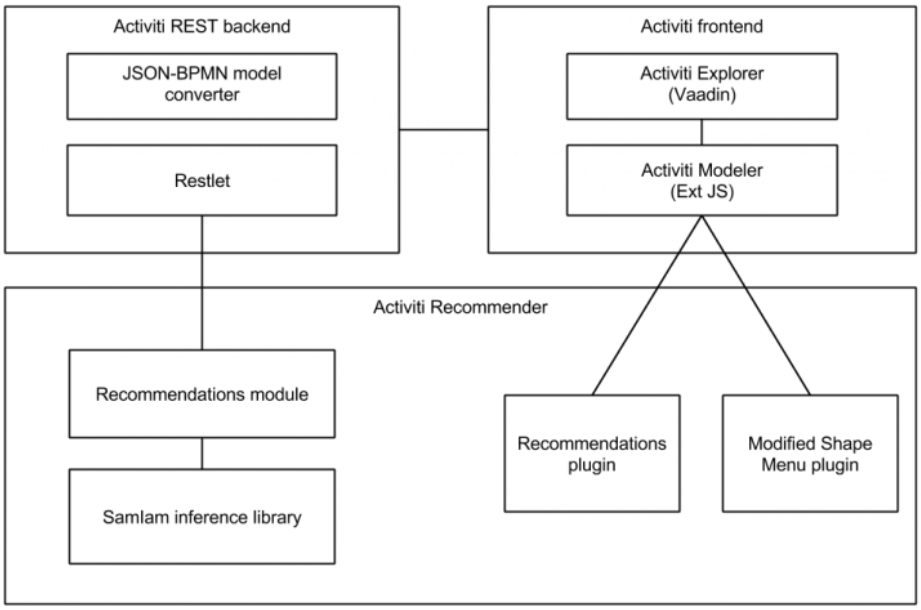


Figure 4. Architecture of the Activity recommender extension.

The output of this three modules is an input for the Activity recommender presented in the following section.

4.1 Implementation

The architecture of the solution consists of four elements as depicted in Figure 4. These elements are:

- Recommendation module – an element responsible for providing recommendations based on the given BN model and evidences. It executes the recommendations queries described in Section 3.
- SamIam inference library [6] – A library that allows for probabilistic reasoning that is used by the recommendation module. The probabilities of occurrence of elements in designed BPMN diagram are calculated by this element.

[6] `http://reasoning.cs.ucla.edu/samiam/`

– Recommendation plugin – A user interface element, that presents the recommendations to the designer and allows to query the recommendation module.
– Shape Menu plugin – A plugin that is a set of icons that surround a selected block providing shortcuts for the most commonly used operations. In this case, a plugin allows to insert the recommended element just after the selected one (see Figure 5).

The Recommendation module and SamIam inference library were encapsulated into a webservice restlet to fit the Activity software architecture. Recommendation plugin and Shape Menu plugin have been implemented as a frontend plugins for Activity modeller. The communication between frontend and backend is based on the JSON exchange format.

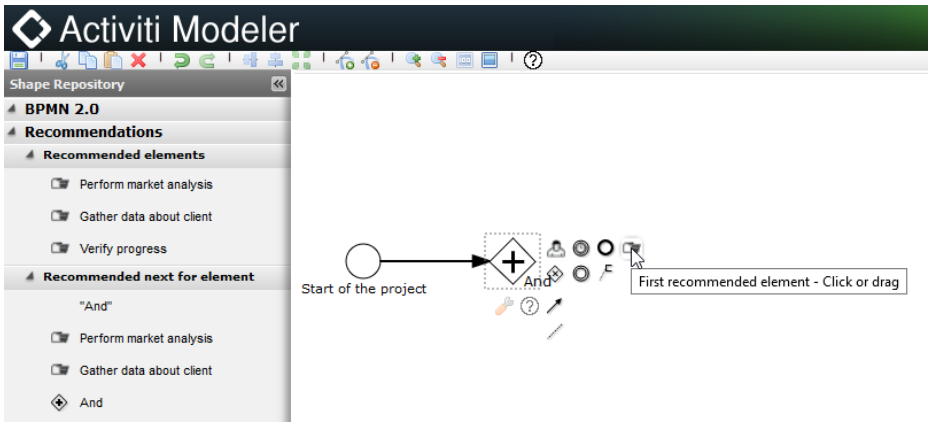


Figure 5. An example of a recommendation process in Activity modeller

To better visualize this process of recommendation performed by the Activity recommender, lets assume that the previously trained Bayesian network was deployed into the Activity recommender system. When a designer queries the system for the recommendation, all the BPMN elements labels that were already placed by the designer into the model are treated as an evidence. The currently selected element is tarted as a reference element for which the forward recommendation should be performed. All the evidence are packaged into the JSON format and sent to the backend, where the recommendation module performs a query to the BN and returns the recommended elements back to the frontend. In the frontend the recommendation plugin and shape menu plugin present the recommendations to the designer and the process continues.

The following section describes a brief evaluation of the descried solution on the simple use-case scenario.

4.2 Evaluation

The evaluation of the Activity recommender was performed on the simple model presented in Figure 2. The model was learnt from the configurable process presented in Figure 1 with an auxiliary modules described briefly at the beginning of this section.

The Figure 5 presents the beginning of the design process, when only two elements are inserted into the diagram: the *Start of the project* element and the *And* gateway. The Bayesian network representing this state was depicted in the Figure 3. If the user selects the node and presses the button depicted with a question mark icon that is located on the top bar of the modeller, the recommendation query will be send to the recommendation module. The module will then perform forward recommendation starting from the element that was selected by the designer (in this case the *And* gateway). The results of the query are presented in the sidebar on the left side of the Activity modeller, and are also accessible through shape menu plugin, which is activated when the user hover the mouse over the element.

As presented in Figure 5, the recommendations are consistent with the probabilities calculated from the Bayesian network presented in Figure 3. It is worth noting, that although in the Bayesian network there exist four *And* gateways, the correct gateway was chosen for the recommendation reference point, thanks to the neighborhood scanning algorithm described in Section 3.

5 Summary and future work

In this paper we presented an implementation and evaluation of the structural recommendation module for BPMN diagrams. We integrated one of the most popular BPMN modeller called Ativity with our recommendation module providing practical tool for structural recommendation of BP models. We also presented an approach that supports matching the similar areas of two graphs that is based on the Levenshtein metric and neighbor scanning algorithm. The evaluation was presented on a simple use-case scenario that was part of the Prosecco project.

The future works assumes implementing remaining two recommendation modes that are: backward recommendation and autocompletion. It is also considered to compare the solution based on the Bayesian networks to the other approach that originates from phrase prediction algorithms [27].

References

1. OMG: Business Process Model and Notation (BPMN): Version 2.0 specification. Technical Report formal/2011-01-03, Object Management Group (2011)
2. Allweyer, T.: BPMN 2.0. Introduction to the Standard for Business Process Modeling. BoD, Norderstedt (2010)
3. Nalepa, G.J., Kluza, K.: UML representation for rule-based application models with XTT2-based business rules. International Journal of Software Engineering and Knowledge Engineering (IJSEKE) **22** (2012) 485–524
4. Kluza, K., Baran, M., Bobek, S., Nalepa, G.J.: Overview of recommendation techniques in business process modeling. In Nalepa, G.J., Baumeister, J., eds.: Proceedings of 9th Workshop on Knowledge Engineering and Software Engineering (KESE9) co-located with the 36th German Conference on Artificial Intelligence (KI2013), Koblenz, Germany, September 17, 2013. (2013)
5. Bobek, S., Baran, M., Kluza, K., Nalepa, G.J.: Application of bayesian networks to recommendations in business process modeling. In Giordano, L., Montani, S., Dupre, D.T., eds.: Proceedings of the Workshop AI Meets Business Processes 2013 co-located with the 13th Conference of the Italian Association for Artificial Intelligence (AI*IA 2013), Turin, Italy, December 6, 2013. (2013)

6. Koschmider, A., Hornung, T., Oberweis, A.: Recommendation-based editor for business process modeling. Data & Knowledge Engineering **70** (2011) 483 – 503
7. Born, M., Brelage, C., Markovic, I., Pfeiffer, D., Weber, I.: Auto-completion for executable business process models. In Ardagna, D., Mecella, M., Yang, J., eds.: Business Process Management Workshops. Volume 17 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2009) 510–515
8. Chan, N., Gaaloul, W., Tata, S.: Context-based service recommendation for assisting business process design. In Huemer, C., Setzer, T., eds.: E-Commerce and Web Technologies. Volume 85 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2011) 39–51
9. Chan, N., Gaaloul, W., Tata, S.: A recommender system based on historical usage data for web service discovery. Service Oriented Computing and Applications **6** (2012) 51–63
10. Leopold, H., Mendling, J., Reijers, H.A.: On the automatic labeling of process models. In Mouratidis, H., Rolland, C., eds.: Advanced Information Systems Engineering. Volume 6741 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 512–520
11. Leopold, H., Smirnov, S., Mendling, J.: On the refactoring of activity labels in business process models. Information Systems **37** (2012) 443–459
12. Kopp, O., Leymann, F., Schumm, D., Unger, T.: On bpmn process fragment auto-completion. In Eichhorn, D., Koschmider, A., Zhang, H., eds.: Services und ihre Komposition. Proceedings of the 3rd Central-European Workshop on Services and their Composition, ZEUS 2011, Karlsruhe, Germany, February 21/22. Volume 705 of CEUR Workshop Proceedings., CEUR (2011) 58–64
13. Mazanek, S., Minas, M.: Business process models as a showcase for syntax-based assistance in diagram editors. In: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems. MODELS '09, Berlin, Heidelberg, Springer-Verlag (2009) 322–336
14. Mazanek, S., Rutetzki, C., Minas, M.: Sketch-based diagram editors with user assistance based on graph transformation and graph drawing techniques. In de Lara, J., Varro, D., eds.: Proceedings of the Fourth International Workshop on Graph-Based Tools (GraBaTs 2010), University of Twente, Enschede, The Netherlands, September 28, 2010. Satellite event of ICGT'10. Volume 32 of Electronic Communications of the EASST. (2010)
15. Hornung, T., Koschmider, A., Lausen, G.: Recommendation based process modeling support: Method and user experience. In: Proceedings of the 27th International Conference on Conceptual Modeling. ER '08, Berlin, Heidelberg, Springer-Verlag (2008) 265–278
16. Koschmider, A., Oberweis, A.: Designing business processes with a recommendation-based editor. In Brocke, J., Rosemann, M., eds.: Handbook on Business Process Management 1. International Handbooks on Information Systems. Springer Berlin Heidelberg (2010) 299–312
17. Wieloch, K., Filipowska, A., Kaczmarek, M.: Autocompletion for business process modelling. In Abramowicz, W., Maciaszek, L., Węcel, K., eds.: Business Information Systems Workshops. Volume 97 of Lecture Notes in Business Information Processing. Springer Berlin Heidelberg (2011) 30–40
18. Chan, N., Gaaloul, W., Tata, S.: Assisting business process design by activity neighborhood context matching. In Liu, C., Ludwig, H., Toumani, F., Yu, Q., eds.: Service-Oriented Computing. Volume 7636 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2012) 541–549
19. Minor, M., Bergmann, R., Görg, S., Walter, K.: Towards case-based adaptation of workflows. In Bichindaritz, I., Montani, S., eds.: ICCBR. Volume 6176 of Lecture Notes in Computer Science., Springer (2010) 421–435
20. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29** (1997) 131–163
21. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
22. Neapolitan, R.E.: Learning Bayesian Networks. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (2003)

23. Dijkman, R., Dumas, M., García-Bañuelos, L.: Graph matching algorithms for business process model similarity search. In Dayal, U., Eder, J., Koehler, J., Reijers, H., eds.: Business Process Management. Volume 5701 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2009) 48–63
24. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., Mendling, J.: Similarity of business process models: Metrics and evaluation. Information Systems **36** (2011) 498 – 516 Special Issue: Semantic Integration of Data, Multimedia, and Services.
25. Sigman, M., Cecchi, G.A.: Global organization of the wordnet lexicon. Proceedings of the National Academy of Sciences **99** (2002) 1742–1747
26. Levenshtein, V.: Binary Codes Capable of Correcting Deletions, Insertions and Reversals. Soviet Physics Doklady **10** (1966) 707
27. Nandi, A., Jagadish, H.V.: Effective phrase prediction. In: Proceedings of the 33rd International Conference on Very Large Data Bases. VLDB '07, VLDB Endowment (2007) 219–230

Towards an ontological analysis of BPMN

Emilio M.Sanfilippo[1,2], Stefano Borgo[2], and Claudio Masolo[2]

[1] Institute of Industrial Technologies and Automation, ITIA-CNR, Italy
emilio.sanfilippo@itia.cnr.it
[2] Laboratory for Applied Ontology, ISTC-CNR, Italy

Abstract. BPMN is a de-facto standard with more than 70 commercial tools that currently support its use. However, its the semantic ambiguities jeopardize its implementation. We perform an ontological analysis of important constructs of BPMN like activities and events to highlight their implicit commitments.

Keywords: Ontological analysis, BPMN, State, Event, Activity

1 Introduction

Business process (BP) modelling concerns the analysis and representation of the activities by which companies coordinate their organisation and work, produce goods, interact with each others and customers. The goal is a common conceptual modelling language that can be easily understood to facilitate business communication. The Business Process Model and Notation (BPMN) [9] is one of such BP languages and is a OMG standard with more than 70 commercial tools that currently supports its use[3]. BPMN focuses on graphical constructs and lacks formal semantics. Thus, it presents conceptual ambiguities regarding the interpretation of its metamodel and the supporting software tools are not guarantee to interoperate. We use ontological analysis to dwell into the backbone elements of BPMN like activity and event. The goal is to investigate whether the standard is (possibly implicitly) committed to some (coherent) ontological perspective. The remainder of the paper is organized as follows: Section 2 describes the state of the art about the analysis of BPMN. Section 3 looks at a BPMN process diagram to highlight some problematic issues. Sections 4 gives the ontological analysis of our target notions.

2 State of the art

The literature about BPMN focuses on three types of analysis: (*i*) the *syntactic* analysis, (*ii*) the *behavioral* analysis, and (*iii*) the *ontological* analysis. The syntactic analysis aims at defining the structural constraints that BPMN-models

[3] This work is part of a larger study [11] that will be presented at the International Conference on Formal Ontology in Information Systems (FOIS 2014). In this version we assume some familiarity with BPMN.

must satisfy. [4] presents the BPMNO meta-ontology implemented in OWL [7]. BPMNO allows reasoning with semantically annotated processes and it enriches BPMN with, e.g., temporal information. Similarly, [1] provides a BPMN ontology encoded in the WSML language [2]. The behavioral analysis looks at what can happen during the execution of a well-formed BPMN model like, e.g., the existence of deadlocks or livelocks [3]. This (static) analysis of a process model considers the semantics underlying the schema only for procedural information and is not relevant for our work in this paper. Finally, the ontological analysis focuses on the characterization of the primitives of the language. [8] discusses the OPAL reference framework and characterizes the specific kinds of activities or events present in a given model, but it lacks a characterization of the general difference between BPMN-activities and BPMN-events. [5] uses the ontology ABDESO to distinguishes actions from other events. The authors find quite a few ambiguous and redundant elements, as well as missing concepts in BPMN. Finally, [10] looks at the mutual relationship between BPMN and the Bunge-Wand-Weber (BWW) foundational ontology. The paper highlights some ontological shortcoming in the first release of the standard with respect to ontological completeness, construct overload, construct excess and construct redundancy.

In this paper we are interested in the ontological analysis of BPMN with the aim of clarifying how one can understand the notions of activity and event. We carry out our study in two ways: first by ontologically analyzing the information provided by the BPMN standard, and then by characterizing our findings on these concepts with the DOLCE ontology [6]. We are not proposing an evaluation of BPMN with respect to an ontology; we rather use the ontology to find (possibly implicit) commitments of the standard, identify business related elements that the standard does not address, and to highlight the variety of interpretations that are consistent with these constraints.

3 Activities and Events in BPMN

The BPMN diagram in Fig. 1 represents a process with four participants: *Purchaser*, *Service provider A*, *Service provider B* and *Service provider C*. The process starts when the event type *None* in the *Purchaser* pool happens. This is followed by the execution of task *Request Quotes* which ends with the sending of a message to each service provider participant in the process. Once a service provider receives the message, it starts its own process consisting in sending a quote to the *Purchaser*. After this, the service provider process ends. When the *Purchaser* receives at least two quotes, it executes the *Assess the Quotes* task after which the process ends for the *Purchaser* provided the condition *Sufficient reserve amount?* is satisfied. Otherwise, the process is back to the *Request Quotes* and flows again.

The reason why some parts of the process are marked as activities and others as events is not immediately clear. In the diagram below, messages are exchanged between the process participants by using tasks of type *Message send*. However, in BPMN one could also use a *Message* as throw event (not showed in Figure

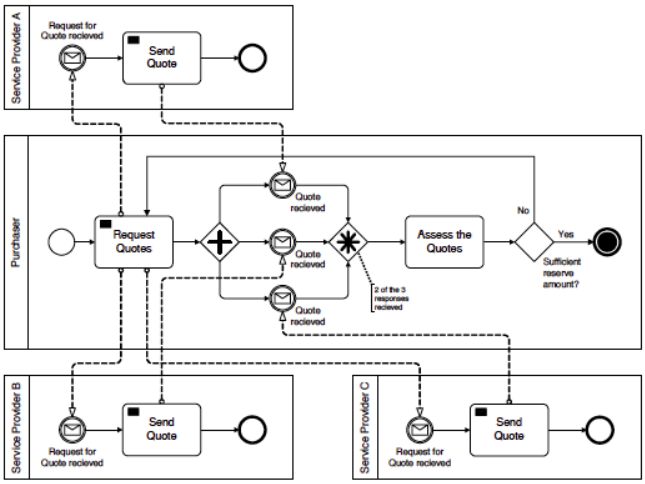


Fig. 1. BPMN process diagram, taken from [9, p.362]

1), which models the sending of a message as well. The meanings of these two different constructs – that seem to model the same thing – is not immediately clear. The BPMN specification [9] provides little help in clarifying the distinction between activity and event. The BPMN Quick Guide[4] states that "[...] an *Event* maps to a time point on a time line [while] a *Task* [activity] maps to a time interval". This seems to mean that events are instantaneous while activities last in time, which implies that temporal atomicity is a discriminating property. Nevertheless, BPMN does not commit to a theory of temporal points or intervals, thus every reference to time remains vague. Another possibility is to understand activities and (at least some) events in terms of endogenous vs exogenous entities: the first are happenings controlled within the poll the latter are out of the control of the pool.

4 Ontological analysis of BPMN events and activities

Events and activities in BPMN are connected to other events and activities in the same (in a different) pool by solid (dashed) arrows; these arrows mark execution precedence and thus temporal dependences. This reveals the temporal nature of events and activities in BPMN. From the ontological viewpoint, we can classify them as (some type of) perdurants or occurrents, in the sense of entities that extend in time possibly by accumulating temporal parts. In the following we use the DOLCE taxonomy of perdurants—mainly the concepts of *Achievement*, *Accomplishment*, *State*, and *Event*—to discuss and ontologically characterize the difference between *Activities* and *Events*, and between *Catch-* and *Throw-events*.

[4] http://www.bpmn.org

While we find helpful to use a foundational ontology like DOLCE, we remark that the analysis could be based on other ontological systems provided they include a taxonomy of perdurants.

Activities and tasks: We have seen that BPMN activities are not instantaneous, thus they take time to execute. In addition, BPMN distinguishes between two types of activities: *tasks*, i.e., atomic activities and *sub-processes*, i.e., non-atomic activities. The relationship between *being instantaneous* and *being atomic* is not trivial given that a task can have a positive temporal extension.

In some ontological theory [12] it is assumed that perdurants extend in time by having different temporal slices at different times. This would rule out BPMN tasks because, by extending in time, they *necessarily* have (temporal) proper parts, where 'necessity' is here used in the ontological sense, namely tasks have temporal parts in all possible worlds. According to this perspective, a task like *Request Quotes* is necessarily *anti-atomic* and *anti-homeomeric*, i.e., all its instances have parts that do not belong to *Request Quotes*. The anti-homeomericity is evident for BPMN sub-processes, whose structure is explicitly included in the BPMN-model.However, it might be suggested that BPMN models the world at some *granularity*, in the sense that tasks are *considered to be atomic* in the context of the model even though they have temporal parts in the actual world. In this case, tasks could be *conceived as* atomic or non-atomic depending on the context, granularity, or perspective on the world. We talk in this case of conceptual modality, because the ontological status of tasks relies not on how they are, but rather on how they are conceived within a certain conceptual framework.

We observe also that the mereological sum of two instances of a task like *Send quote* is not an entity of the same type. This is consistent with the assumption that BPMN activities represent units of work with the purpose of achieving given business goals; they thus *culminate* with the achievement of the goal.

We can then conclude that by considering a strict ontological modality activities are anti-atomic and anti-cumulative, i.e., they can be mapped to DOLCE *accomplishments*. Vice versa, by assuming a conceptual modality, only sub-processes may be mapped to accomplishments. More generally tasks would be mapped to DOLCE *events*, i.e., as anti-cumulative types of perdurants with no commitment on atomicity and homeomericity.

Catch events: We saw that events are instantaneous, consequently they are temporally atomic, that is, they cannot extend over or throughout a temporal stretch. *Catch* events like the reception of a message, are in general exogenous, i.e., their happening is outside of the control of the pool they belong to or, at least, of the branch of the pool process at stake. In this perspective *None* start events could be understood as 'the system is turned-on'. In addition, being culminating perdurants the catch events are anti-cumulative. Anti-cumulativity and atomicity characterize the subcategory of achievements in DOLCE.

The process of *Service Provider A* in Figure 1 cannot proceed unless a trigger is received, i.e., a message is received. Accordingly, if the system of this service provider is 'turned-off', the message will never be received. Thus, behind a catch

event there is the assumption that the process is waiting to be triggered, i.e., the system is on a receiving mode.

Differently from activities, these kinds of perdurants (e.g., *waiting*) are *homeomeric*—i.e., the temporal slices of waiting-instances (if any) are themselves waiting instances—and *cumulative*—i.e., the mereological sum of two (immediately consecutive) waiting-instances is still a waiting-instance. Homeomeric and cumulative perdurants are called *states* in DOLCE. For example, Figure 1 indicates that *Service providers A, B* and *C* are (by default) in a waiting status for receiving messages. Thus, a catch event identifies (perhaps implicitly) a state and it further indicates that the pool is committed towards a specific trigger to occur.

However, on the same line of [10], one could consider catch events as state-transitions. For example, the reception of messages can be understood by referring to two states: the state of 'waiting for the message' and the state of 'message received', where the latter is the pre-condition for executing the successive task. The trigger thus enacts a state transition and, in turn, the starting of the new state enables the process to perform its subsequent tasks[5].In the case of *None* catch (start, intermediate) the trigger that is holding the process is not specified. From our viewpoint, there are at least two possible views regarding the semantics of this modelling construct. It might be a placeholder for the initial temporal boundary of the process that in our example corresponds, as a logical constraint, to 'there are no parts of the process that precede the *Request Quote* task'. In this case, the *None* catch bears no further ontological commitment. On the other side, one can return to the idea of a (hidden) waiting state. The latter case seems to be incompatible with the interpretation of the start event as 'the system is turned-on'.

Throw events: Similarly to catch events, throw events are instantaneous, then temporally atomic, and anti-cumulative, i.e., in DOLCE they are classified as achievements. Differently from *catch* events, throw events tend to be endogenous: actions under the control of the pool they belong to. Note that differently from tasks, which can be conceived as structured perdurants, although atomic under a certain granularity, throw events are punctual, thus intrinsically unstructured.

Throw *None end events* can be understood as the achievement of the whole process, and, in a fashion similar to start events, they can be interpreted either as an ontologically neutral placeholder in the model, as a logical constraint, or as an (ontologically committed) achievement. Note that the throw end events marked with a specific trigger icon, like *Message*, *Terminate*, *Signal* and *Error*, indicate an achievement as well but now the culmination point is qualified: the triggers that are specified in these cases (message, signal, termination and error) are amongst the participating entities of the achievement.

[5]The analysis of the causal dependencies among triggers, events and tasks could be very informative.

5 Conclusions

We focused on the ontological analysis of the BPMN notions of activity (task) and event, and classified them within the DOLCE account of perdurant entities. The results are still preliminary and the hope is that it can help to reach a deeper understanding of the system; and to develop sound BPMN-driven ontologies. In the future, we shall expand this first analysis and develop a formalization capturing our results on BPMN.

Acknowledgements: This research is in part supported by the Gecko and Pro2Evo projects of the "Fabbrica del Futuro" (funding: MIUR).

References

1. W. Abramowicz, A. Filipowska, M. Kaczmarek, T. Kaczmarek Semantically enhanced Business Process Modelling Notation. In M. Hepp et al. (Eds.), *Semantic Business Process and Product Lifecycle Management. Proceedings of the Workshop SBPM 2007*, Innsbruck, April 7, 2007
2. J. de Bruijn, D. Fensel, U. Keller, M. Kifer, H. Lausen, R. Krummenacher, A. Polleres, L. Predoiu Web Service Modeling Language (WSML). W3C Member Submission 3 June 2005 Available at: http://www.w3.org/Submission/WSML. Last access March 2014
3. R.M. Dijkman, M. Dumas, and C.Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software Technology*, 50(12):1281–1294, 2008.
4. C. Ghidini, M. Rospocher, L. Serafini A Formalisation of BPMN in Description Logics. Technical Report TR 2008-06-004, FBK-irst, 2008.
5. G. Guizzardi, G. Wagner Can BPMN Be Used for Making Simulation Models? 7th International Workshop on Enterprise & Organizational Modeling and Simulation (EOMAS 2011), together with the 23rd International Conference on Advanced Information System Engineering (CAiSE'11), London, UK.
6. C. Masolo, S. Borgo, A. Gangemi, N. Guarino, A. Oltramari. WonderWeb Deliverable D18. Ontology Library. Available at: http://wonderweb.man.ac.uk/deliverables.shtml, 2002. Accessed March 2014
7. D.L. McGuinnes, F. van Harmelen OWL Web Ontology Language. Overview. W3C Recommendation 10 February 2004. Available at: http://www.w3.org/TR/owl-features. Last access March 2014
8. M. Missikoff, M. Proietti, and F. Smith. Linking ontologies to business process schemas. Technical Report 10-20, Istituto di Analisi dei Sistemi ed Informatica del CNR, 2010. ISSN: 1128 - 3378.
9. Object Management Group (OMG) Business Process Model and Notation (BPMN). Version 2.0, 2011 Available at: http://www.bpmn.org Last access March 2014
10. J. Recker, M. Indulska, M. Rosemann, P. Green. Do Process Modelling Techniques Get Better? A Comparative Ontological Analysis of BPMN. Australasian Conf. on Information Systems. Sydney, 2005
11. E.M.Sanflippo, S. Borgo and C. Masolo. Events and Activities: Is there an Ontology behind BPMN? In: *Formal Ontology in Information Systems*, IOS Press, 2014
12. T. Sider, *Four-Dimensionalism. An Ontology of Persistence and Time.* Oxford: Clarendon Press, 2001.

CAPJA- A Connector Architecture for PROLOG and JAVA

Ludwig Ostermayer, Frank Flederer, Dietmar Seipel

University of Würzburg, Department of Computer Science
Am Hubland, D – 97074 Würzburg, Germany
{ludwig.ostermayer,dietmar.seipel}@uni-wuerzburg.de

Abstract. Modern software often relies on the combination of several software modules that are developed independently. There are use cases where different software libraries from different programming languages are used, e.g., embedding DLL files in JAVA applications. Even more complex is the case when different programming paradigms are combined like within applications with database connections, for instance PHP and SQL.
Such a diversification of programming languages and modules in just one software application is becoming more and more important, as this leads to a combination of the strengths of different programming paradigms. But not always, the developers are experts in the different programming languages or even in different programming paradigms. So, it is desirable to provide easy to use interfaces that enable the integration of programs from different programming languages and offer access to different programming paradigms.
In this paper we introduce a connector architecture for two programming languages of different paradigms: JAVA as a representative of object oriented programming languages and PROLOG for logic programming. Our approach provides a fast, portable and easy to use communication layer between JAVA and PROLOG. The exchange of information is done via a textual term representation which can be used independently from a deployed PROLOG engine. The proposed connector architecture allows for *Object Unification* on the JAVA side.
We provide an exemplary connector for JAVA and SWI-PROLOG, a well-known PROLOG implementation.

Keywords. Multi-Paradigm Programming, Logic Programming, Prolog, Java.

1 Introduction

Business applications often are implemented with object oriented techniques. JAVA currently is one of the most used object oriented programming languages with rich libraries and a very active community. There are tools based on JAVA for writing complex rules, but these tools still come with flaws [10]. Logic programming languages like PROLOG are particular suitable to write rules more intuitively and declaratively, which helps in building, updating and testing complex structured sets of rules as we have successfully shown in the field of e-commerce in [9]. Because JAVA is the main programming language in most large-scale applications, it is desirable to connect JAVA with PROLOG for certain problem domains.

Many approaches have been proposed to make PROLOG available in JAVA, but in many cases there is no clear distinction between JAVA and PROLOG as they use PROLOG concepts like terms and atoms directly in JAVA. Our efforts are to keep PROLOG structures off from JAVA, but to enable in JAVA the use of existing PROLOG rules and facts. Therefore we propose a fast, portable and intuitive connector architecture between JAVA and PROLOG.

In our approach, objects can directly be used as PROLOG goals, without creating complex structures in JAVA that represent the terms in PROLOG. Member variables that are equal `null` in JAVA are translated into PROLOG variables. Those variables are unified by PROLOG when querying a JAVA object as a goal in PROLOG. The values, the variables are unified with, are set to the corresponding member variables of the JAVA objects. We call this mechanism *Object Unification*. Apart from using existing JAVA classes for Object Unification, we also provide in PROLOG a generator for JAVA classes. The instances of generated classes unify with terms initially passed to the generator.

The remainder of this paper is organized as follows. In Section 2 we look at related work and compare those concepts with our own approach. Section 3 introduces the components of the proposed connector. We show the mechanics of the object term mapping in Section 3.1 and in Section 3.2 the parsing of PROLOG terms in JAVA. An exemplary interface for JAVA and SWI PROLOG completes the connector architecture in Section 3.3. After that, the workflow with our connector architecture is shown in Section 4 from two viewpoints: from JAVA and from PROLOG. In Section 5 we evaluate our approach and finally discuss future work in Section 6.

2 Related Work

Providing a smooth interaction mechanism for JAVA and PROLOG is a challenging problem that has been studied in several research papers of the last decade.

A well known and mature interface between JAVA and PROLOG is JPL [13]. To enable a fast communication JPL provides JAVA classes that represent directly the structures in PROLOG. This leads to much code for complex PROLOG term structures. Also, it requires that either the JAVA developer knows how to program PROLOG or the PROLOG developer knows how to code JAVA in order to build the necessary structures in JAVA via classes like Compound. Furthermore, it is limited to the use with SWI-PROLOG, as it is shipped and created for just this single PROLOG implementation.

An interesting approach is INTERPROLOG [2] that uses the JAVA serialization mechanism in order to send serialized JAVA objects to PROLOG. These strings are analysed in PROLOG with definite clause grammars and a complex term structure is created which describes the serialized object. However, this generated object term structure is complex and contains a lot of class meta information that is not as natural for a PROLOG programmer as the textual term representations of objects in our approach.

The concepts of linguistic symbiosis have been used in [3, 6, 7] to define a suitable mapping. Methods in JAVA are mapped to queries in PROLOG. This differs from our approach, as we use JAVA objects for terms as well as for queries in PROLOG.

A customisable transformations of JAVA objects to PROLOG terms was introduced with JPC [4]. Instead of using annotations, as it is done in our approach to customise the

mapping, in JPC custom converter classes can be defined. These converters implement methods which define the translation between objects and terms. This causes in a lot of extra code and files as the user has to define the converter classes instead of just writing annotations to existing classes.

In [5] tuProlog, a PROLOG engine entirely written in JAVA, was integrated into JAVA programs by using JAVA annotations and generics. But other than in our approach, PROLOG rules and facts are written directly into the JAVA code within annotations. Querying rules and facts is done again by JAVA methods. The mapping of input and return to arguments of a goal in PROLOG is defined with annotations. In contrast to our attempt, this approach is strongly dependent on tuProlog and therefore not compatible to other PROLOG engines.

In [11], we have presented the framework PBR4J (PROLOG Business Rules for JAVA) that allows to request a given set of PROLOG rules from a JAVA application. To overcome the interoperability problems, a JAVA archive has been generated that contains methods to query the set of PROLOG rules. PBR4J uses XML Schema to describe the data exchange format. From the XML Schema description, we have generated JAVA classes for the JAVA archive. In our new approach the mapping information for JAVA objects and PROLOG terms is not saved to an intermediate, external layer. It is part of the JAVA class we want to map and though we can get rid of the XML Schema as used in PBR4J. Either the mapping is given indirectly by the structure of the class or directly by annotations. While PBR4J just provides with every JAR only a single PROLOG query, we are now able to use different goals depending on which variables are bound. PBR4J transmitted along with a request facts in form of a knowledge base. The result of the request was encapsulated in a result set. With our connector architecture we do not need any more wrapper classes for the knowledge base and the result set as it was with PBR4J. That means with our new connector we have to write less code in JAVA. We either assert facts from a file or persist objects with a JAVA method to the database of PROLOG.

3 A Connector for PROLOG and JAVA

The connector for PROLOG and JAVA is based on our work with mappings between objects in JAVA and terms in PROLOG. Before we discuss the individual parts of our connector, we recap briefly the highly customisable Object Term Mapping (`OTM`) which we have introduced in [12]. In addition to a simple, yet powerful default mapping for almost every class in JAVA, different mappings between objects and terms also easily can be defined. We call the mapping of an object to a PROLOG term *Prolog-View* on the given object. Multiple Prolog-Views for a single object can be defined. For this purpose, we only need three annotations in JAVA in a nested way as shown in Figure 1. Because JAVA does not support multiple annotations of the same type within a class until version 7, we use the annotation `@PlViews` to allow multiple `@PlView` annotations in a single given class. A `@PlView` is identified by `viewId` and consists of the following elements to customize the mapping of an object to a term. `functor` is used to change the target term's functor. The list `orderArgs` changes the arguments order and the list `ignoreArgs` prevents member variables to be mapped as arguments of the target

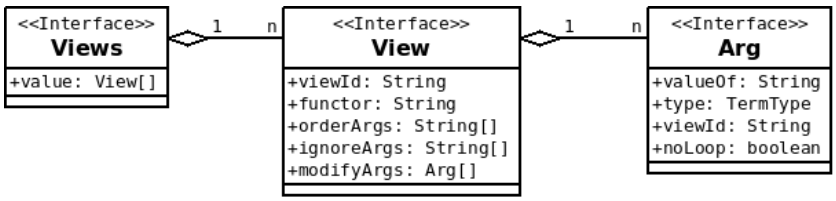


Fig. 1: The Interfaces for `@PlViews`, `@PlView` and `@Arg`

term. The list `modifyArgs` consists of `@Arg` annotations which are used to modify the mapping of a single member variable of the object. The member variable is referenced by `valueOf` and the type in PROLOG can be modified with `type`. If the member variable is a class type that has `@PlView` annotations, a particular Prolog-View can be selected via the appropriate `viewId`. All in all, arbitrary complex nested term structures can be created by the mapping. The following example shows a `Person` class and two different Prolog-Views on `Person`:

```
@PlViews({
  @PlView(viewId="personView1",
    ignoreArgs={"id"},
    modifyArgs=
       {@PlArg(valueOf="children", viewId="personView2")})
  @PlView(viewId="personView2", functor="child",
    orderArgs={"givenName")}
)
class Person {
  private int id;
  private String givenName;
  private String familyName;
  private Person[] children;
  // ... constructor/ getter / setter
}
```

In the listing below instances of `Person` are given followed by the textual term representation under the default mapping and under the Prolog-View `personView1`:

```
Person p1 = new person(1, 'Homer', 'Simpson');
Person p2 = new person(2, 'Bart', 'Simpson');
p1.setChildren(new Person[]{p2});

// default mapping of p1
"person(1,'Homer','Simpson',[person(2,'Bart','Simpson',[])])"
// mapping of p1 under the Prolog-View "personView1"
"person('Homer', 'Simpson', [child('Bart')])"
```

All the information needed for the creation of textual term representations are derived from the classes involved in the mapping. The default mapping uses the information of the classes structure itself. The customised mapping uses the information contained in the annotations `@PlView`.

3.1 Creating Textual Term Representations

We only need two classes in JAVA to request PROLOG as shown in Figure 2. The conversion as well as the parsing is implemented within the wrapper class `OTT` (Object-Term-Transformer). The class `Query` is used to start a call to PROLOG. An example

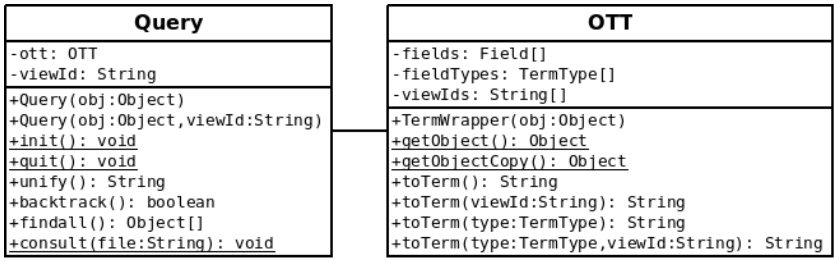


Fig. 2: Classes for CAPJA

for the usage of these classes is shown in Figure 3. The object `o1` is destined to be unified in PROLOG. It has references to two other objects `o2` and `o3` which will lead to a nested term structure in PROLOG. When the instance `query` gets `o1` passed to its constructor, `query` creates an instance of `OTT`, here `ott1`. For all the other references in `o1` instances of `OTT` are created in a nested way, namely `ott2` for `o2` and `ott3` for `o3`.

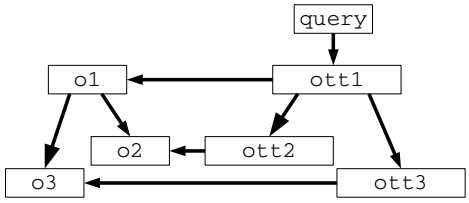


Fig. 3: A Dependency Tree of `OTT` Objects

In order to create the textual term representation of `o1`, the instance `query` causes `ott1` to call its `toTerm()` method that triggers a recursive call of `toTerm()` in all involved instances of `OTT`. In doing so, the first operation is to determine which fields have to be mapped. Dependent on the `viewId` of the requested Prolog-View or on the default mapping, an array of Field references is created that contains all the needed member variables for the particular view in the corresponding order. The information about the Fields is retrieved with help of the Reflection API in JAVA. The same way, additional information like PROLOG types and `viewIds` for particular member variables are saved within such arrays. As the information about a view of a class is solid and does not change with the instances, this field information is just created once and

cached for further use. For the creation of the textual term representation the functor is determined either from a customised `functor` element of an `@PlView` annotation or from the class name in the default case. After that, the Field array is iterated and the string representation for its elements are created. The pattern of those strings depend on the PROLOG type that is defined for a member. If a member is a reference to another object, the `toTerm()` method for the reference is called recursively.

3.2 Parsing Textual Term Representations

After `query` has received a textual representation of the unified term from PROLOG, it is parsed to set the unified values to the member variables of the appropriate objects in JAVA. The parsing uses again the structure of the nested `OTT` objects as shown in Figure 3. The class `OTT` has the method `fromTerm(String term)`. This method splits the passed string into functor and arguments. The string that contains all the arguments is split into single arguments. This is done under consideration of nested term structures. According to the previously generated Field array the arguments are parsed. This parsing happens in dependence on the defined PROLOG type of an argument. For instance, an atom either has single quotes around its value or, if the first character is lowercase, there are no quotes at all. If there is a quote detected, it is removed from the string before assigning it as a value for the appropriate member variable. Assignments for referenced objects in `o1` are derived recursively by calling the `fromTerm(String term)` method of the appropriate instances of `OTT`, in our example `ott2` and `ott3`.

3.3 The Interface for JAVA and SWI-PROLOG

Although the complete mapping process is located in JAVA, we still need an interface to the PROLOG implementation of choice in order to connect both programming languages. The open-source PROLOG implementation SWI-PROLOG [14] comes with the highly specialized, and for SWI optimized, JAVA interface JPL. We have implemented our own JAVA interface for SWI which is optimized for our mapping. Similar to JPL we use SWI's Foreign Language Interface (FLI) and the JAVA Native Interface (JNI). The FLI is bundled with SWI and provides a native interface which can be used to extend SWI by further (meta-)predicates. The FLI also provides an interface for C and is therefore accessible for all other programming languages which have access to C libraries.

We have JAVA on one side and the C interface FLI on the other, so we need the glue to enable the communication between these two worlds. This is done by the JAVA Native Interface (JNI), which enables the usage of in C defined functions in JAVA. With the help of the JNI, we implemented a bridge between JAVA and the SWI-PROLOG system. As mentioned, we focus on the simple transmission of strings that represent terms in PROLOG. This differs from the interface JPL, as our interface does not need complex class structures in JAVA to represent terms in PROLOG. We simply send strings to PROLOG and receives strings from it. The transmitted strings already satisfy PROLOG's syntax and thus can be converted directly into terms on the PROLOG side.

Via the FLI we provide backtracking if there are more solutions. This leads to a return that contains the next unified term in PROLOG. After sending from JAVA a string containing a goal with the logical variable `X`, our interface for SWI-PROLOG returns the

unified term as a string back to our JAVA application. The user on the JAVA side now can call `backtrack()` to send a backtrack command to SWI-PROLOG which returns the next solution.

4 Workflows

We start from two viewpoints: JAVA and PROLOG. Each viewpoint describes the development phase using our connector architecture.

From JAVA The default mapping enables the JAVA developer to use already existing JAVA classes in PROLOG as facts or as goals. If the default mapping of an object in JAVA does not provide a desired term structure in PROLOG, the textual term representation of the object can be altered by using the appropriate `@PlView` annotations. To unify an existing JAVA class the developer just has to wrap it within an instance of `Query` and call its method `unify` in order to call the class' textual term representation as goal in PROLOG:

```
Person p = new Person();
p.setId(1);
Query q = new Query(p, "personView1");
q.unify();
```

The example request to PROLOG above contains an instance `p` of the class `Person` from Section 3. Note, that the only value that is set for `p` is the `id` attribute. The other attributes are not initialized and therefore equal `null`. The class `Query` manages the call to PROLOG. The optional second parameter of the constructor of `Query` defines which Prolog-View is used for the object `p`. It is specified by the `viewId` element of a `@PlView` annotation, here `personView1`. When the method `unify()` is called the textual term representation is created. This is done either according to the default mapping or under the consideration of existing `@PlView` annotations that are defined for the class `Person` or any other referenced classes in `Person`. This string is already a valid term in PROLOG with arguments that represent the attributes of `p` and all referenced objects in `p`. The textual term representation has only arguments for attributes that are mapped as defined by the default mapping or by a referenced `@PlView` annotation. The textual term representation then is used as goal within the PROLOG query.

In the example above, most attributes of `p` are equal to `null` in JAVA. As `null` is a value that can not be transformed into an appropriate type in PROLOG it has to be handled in a particular way. We consider `null` to be in PROLOG a (logical-)variable that is supposed to be unified. After sending a call to PROLOG containing `null` values, the resulting variables are possibly unified in PROLOG. The unified term is sent back as string and parsed in JAVA. Changes to the initial string sent from JAVA to PROLOG will be detected and set to the initial object by JAVA reflections, in our example to the instance `p` of `Person`. This means, those attributes that formerly have been equal to `null` are set to the values of the variables unified in PROLOG. The original object `p` now has been updated and represents the solution of the unification process in PROLOG.

An important feature of PROLOG is unknown to JAVA: the backtracking mechanism. The method `unify` just returns the first solution PROLOG provides. But via backtracking PROLOG is able to provide other unifiable solutions. These solutions can be retrieved with another method of `Query` that is called `backtrack()`. It sends a backtrack request to PROLOG in order to retrieve the next solution, if there is one. The same way a the solution is set to the original object via `unify()`, the solution via `backtrack()` is set to the variables of the original object in JAVA. As it is not sure that there even are other solutions, `backtrack()` returns a boolean in JAVA whether a solution was found by PROLOG or not.

Similar to JPL, we have implemented a third request: get all solutions of a goal in one call. This is called `findall()`, named after the built-in predicate in SWI PROLOG. This method returns an array of the requested objects, e. g. Person. As the method returns multiple objects with different values in their variables, we have to create for each solution a new object. So, when using this method the original object is not touched at all. Creating new objects for every solution is the reason why we need the unifiable objects to have a default constructor in JAVA.

Beside these basic methods for Object Unification there is a method for asserting JAVA objects to the PROLOG database. This method is called `persist()` and just takes the generated string representation of the PROLOG term and asserts it by using the `assertz/1` predicate. After that method call the term representation of the appropriate object is available as fact in PROLOG.

From PROLOG Another viewpoint is the writing of PROLOG terms that are destined for the use in JAVA. In contrast to the previous viewpoint, there are no suitable JAVA classes yet within the current project. So, we show now how is easy it is to write PROLOG libraries that are accessible from JAVA by generated classes.

In [12] we have described a default and a customised mapping between JAVA objects and PROLOG terms. As long as no customisation is defined for a JAVA class via special annotations, a default mapping is applied which links a class to a certain term in PROLOG. With annotations in JAVA the user is able to customise the mapping. These annotations determine which of the member variables will be part of the term representation and which PROLOG type they will be (e. g. ATOM, COMPOUND, LIST). It is possible to define several different views on a class.

We also have introduced in [12] the PVN (*Prolog-View-Notation)* that can be used to define in PROLOG the mapping between JAVA objects and PROLOG terms. Expressions in PVN consist of two predicates: `pl_view` and `pl_arg`. The term `pl_view` describes a textual term representation of a JAVA class. The term `pl_arg` term in a PVN expression is used to define the mapping of the member variables.

A Rule in PROLOG can be made accessible from JAVA using the PVN to describe a rule's head. From this PVN expression we generate JAVA classes with the appropriate `@PlView` annotations. For this purpose we have developed two predicates in PROLOG:

```
create_class_from_pvn(?Pvn, ?Class)
create_annotation_from_pvn(?Pvn, ?Annotation)
```

Typically for PROLOG, both predicates can have the first or the second argument as input. The first predicate generates from a PVN expression source code for JAVA containing all necessary classes. These classes map directly to the terms in PROLOG from which we started from. The second predicate is used to generate the `@PlView` annotations.

5 Evaluation

To evaluate our approach we reimplemented the London Underground example as in [3]. We made two implementations, one with JPL and one with our connector. The structure of the London Underground is defined by `connected/3` facts in PROLOG. Speaking of the undirected graph, representing the London Underground with stations as nodes and lines as edges, a `connected` fact describes in this context adjacent stations. The first and the second argument of a `connected` fact is a station. The third argument is the connecting line. We give some examples for `connected` facts:

```
connected(station(green_park), station(charing_cross),
  line(jubilee)).
connected(station(bond_street), station(green_park),
  line(jubilee)).
...
```

In our first implementation we use JPL in order to retrieve a station connected to a given station:

```
1 public class Line {
2   public String name;
3   public Term asTerm() {
4     return new Compound("line", new Term[]{new Atom(name)});
    }
  }
5 public class Station {
6   public String name;
7   public Station(String name) { this.name = name; }
8   public Term asTerm() {
9     return new Compound( "station", new Term[]{
        new Atom(name)});
    }
10  public static Station create(Term stationTerm) {
11    String name = ((Compound)stationTerm).arg(1).name();
12    return new Station(name);
    }
13  public Station connected(Line line) {
14    String stationVarName = "Station";
15    Term[] arguments = new Term[]{asTerm(),
      new Variable(stationVarName), line.asTerm() };
16    Term goal = new Compound("connected", arguments);
17    Query query = new Query(goal);
18    Hashtable<String, Term> solution = query.oneSolution();
```

```
19      Station connectedStation = null;
20      if(solution != null) {
21        Term connectedStationTerm = solution.get(stationVarName);
22        connectedStation = create(connectedStationTerm);}
23     return connectedStation;}}
```

As one can see, the implementation with JPL leads to a lot of lines of code. In the method `connected` the complex term structure is created in order to query the predicate `connected`. The result handling is tedious again. With our approach we do not have to create any term structures in JAVA. Instead, we need to implement an extra class `Connected` representing the goal in PROLOG with the predicate `connected/3`:

```
1 public class Connected {
2   public Station stat1;
3   public Station stat2;
4   public Line line;
5   public Connected() { };
6   public Connected(
        Station stat1, Station stat2, Line line) {
7     this.stat1 = stat1;
8     this.stat2 = stat2;
9     this.line = line;}
  }
10 public class Line {
11   public String name;
  }
12 public class Station {
13   private String name;
14   public Station connected(Line line) {
15     Connected connected = new Connected(this, null, line);
16     Query query = new Query(connected);
17     query.unify();
18     return connected.stat2;}
  }
```

As the following table shows, our approach needs less lines of code to implement the London Underground example than the implementation with JPL.

	Line	Station (w/o connected())	Connected	connected()	sum
JPL loc	4	8	0	11	23
CAPJA loc	2	2	9	5	18

However, lines of code do not say anything about the code's complexity and information density. Our class `Connected` is very simple. It contains only member variables and two simple constructors whereas in JPL already the method `connected()` of the class `Station` is fairly complex.

With the data of the complete London Underground with 297 stations, 13 lines and 412 connections, we made 50,000 executions[1] with both implementations. The result of the performance test is presented in the following table:

[1] Core i5 2 x 2.4 GHz, 6 GB RAM, Ubuntu 14.04

	ø execution time of 50.000 calls
JPL	$\sim 1.2s$
CAPJA	$\sim 2.6s$

Castro et al. did a similar comparison between JPL and their LOGICOBJECTS [3]. Their implementation with LOGICOBJECTS is slower than the corresponding JPL implementation by a factor of about 7 whereas our connector implementation is just about 2.13 times slower.

Aside from a performance improvement in form of field structure caching, as mentioned in Section 4, we identified that getting and setting the values of the member variables via Reflections is slow. In the future we want to use static calls as often as possible instead of using the Reflection API. In order to make use of direct calls, we need to generate *Specialized* `OTT` classes (`SOTT`) for all classes that we want to map. These generated classes contain highly specialized `toTerm()` and `fromTerm(String)` methods that call their member variables directly if public or with their getter and setter methods. This attempt picks up concepts from our prior work in [11]. But this time, we want to make use of so called *Annotation Processors* that extend the JAVA compiler in order to generate additional code at compile time. Those generated `SOTT` classes are only optional. The `OTT` class as presented in this work, still will be used in the case that no `SOTT` class exists. We have implemented an early prototype in order to test these ideas for feasibility. In an early test using `SOTT` classes, we have measured an average time for 50.000 executions of about 1.5 seconds for the London Underground example. This is a huge performance gain and is just about 25% slower than JPL, the highly optimized interface for SWI-PROLOG.

6 Future Work

The presented interface in Section 3.3 has proven to be well applicable for SWI-PROLOG. However, our approach is not limited to this PROLOG implementation. We currently develop a standard interface based on pipes that is suitable for most PROLOG implementations and completes our generic approach. This way, we want to accomplish a portable solution that is independent from any PROLOG implementation.

In addition, we further want to reduce the necessary lines of code. In our current approach we use a wrapper class called `Query` for calling a PROLOG goal. Instead, we could have used an abstract superclass that is extended by the class of an object that is going to be mapped. This superclass manages the `OTT` objects that contain the logic behind the creation of the textual term representations and the parsing. Even the request control for `unify()`, `backtrack()` and `findall()` then is part of this superclass. Using the abstract superclass the request for PROLOG from JAVA in the Underground example in the lines 16, 17 can be reduced to a single line containing just the method call `connected.unify()` which additionally saves the initialisation of a `Query` object.

However, the approach with a superclass has a big drawback: we want to be able to use almost every class in JAVA for the Object Unification. This will not work for classes that already extend a class because JAVA does not support multiple inheritance

yet. In JAVA 8 there is a new feature called *Default Methods* that allows to implement a method directly within a JAVA interface. Using this new feature we can implement all the needed functions as Default Methods in an interface. Because multiple JAVA interfaces can be implemented by a single class, we achieve with this new interface the same reduction in lines of code as with an abstract superclass. This way, we can avoid the multiple inheritance problem for classes.

References

1. A. Amandi, M. Campo, A. Zunino. *JavaLog: a framework-based integration of Java and Prolog for agent-oriented programming.* Computer Languages, Systems & Structures 31.1, 2005. 17-33.
2. M. Calejo. *InterProlog: Towards a Declarative Embedding of Logic Programming in Java.* Proc. Conference on Logics in Artificial Intelligence, 9th European Conference, JELIA, Lisbon, Portugal, 2004.
3. S. Castro, K. Mens, P. Moura. *LogicObjects: Enabling Logic Programming in Java through Linguistic Symbiosis.* Practical Aspects of Declarative Languages. Springer Berlin Heidelberg, 2013. 26-42.
4. S. Castro, K. Mens, P. Moura. *JPC: A Library for Modularising Inter-Language Conversion Concerns between Java and Prolog.* International Workshop on Advanced Software Development Tools and Techniques (WASDeTT), 2013.
5. M. Cimadamore, M. Viroli. *A Prolog-oriented extension of Java programming based on generics and annotations.* Proc. 5th international symposium on Principles and practice of programming in Java. ACM, 2007. 197-202.
6. K. Gybels. *SOUL and Smalltalk - Just Married: Evolution of the Interaction Between a Logic and an Object-Oriented Language Towards Symbiosis.* Proc. of the Workshop on Declarative Programming in the Context of Object-Oriented Languages, 2003.
7. M. D'Hondt, K. Gybels, J. Viviane *Seamless Integration of Rule-based Knowledge and Object-oriented Functionality with Linguistic Symbiosis.* Proc. of the 2004 ACM symposium on Applied computing. ACM, 2004.
8. T. Majchrzak, H. Kuchen. *Logic java: combining object-oriented and logic programming.* Functional and Constraint Logic Programming. Springer Berlin Heidelberg, 2011. 122-137.
9. L. Ostermayer, D. Seipel. *Knowledge Engineering for Business Rules in Prolog.* Proc. Workshop on Logic Programming (WLP), 2012.
10. L. Ostermayer, D. Seipel. *Simplifying the Development of Rules Using Domain Specific Languages in Drools.* Proc. Intl. Conf. on Applications of Declarative Programming and Knowledge Management (INAP), 2013.
11. L. Ostermayer, D. Seipel. *A Prolog Framework for Integrating Business Rules into Java Applications.* Proc. 9th Workshop on Knowledge Engineering and Software Engineering (KESE), 2013.
12. L. Ostermayer, F. Flederer, D. Seipel. *A Customisable Mapping between Java Objects and Prolog Terms.* `http://www1.informatik.uni-wuerzburg.de/pub/ostermayer/paper/otm_2014.html`
13. P. Singleton, F. Dushin, J. Wielemaker. *JPL 3.0: A Bidirectional Prolog/Java Interface.* `http://www.swi-prolog.org/packages/jpl`
14. J. Wielemaker. *SWI Prolog.* `http://www.swi-prolog.org`

Migration of rule inference engine to mobile platform. Challenges and case study.

Mateusz Ślażyński, Szymon Bobek , Grzegorz J. Nalepa

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

Abstract Mobile devices are valuable sources of information about their user location, physical and social activity, profiles and habits. Such an information can be used to build context-aware applications, that are able to adapt their functionality to user needs and preferences. A lot of research have been done in this field of science, providing multiple context-modelling approaches like rules, ontologies, probabilistic graphical models, etc. There were also several solutions developed that allow for efficient context-based reasoning. However, there is still lack of tools and research done in the area of context-awareness with respect to mobile environments. The solutions that were constructed for the desktop platforms cannot be directly migrated to the mobile systems, as the requirements and characteristics of this two environments are disjoint. In this paper we focus on migrating a lightweight rule-based inference engine to a mobile platform. We define the requirements that have to be fulfilled by the mobile reasoning system in order to be efficient and universal with respect to the variety of mobile operating systems available nowadays.

1 Introduction

Mobile devices such as smart phones or tablets have been becoming better in terms of hardware capabilities, including speed and storage. Moreover, they are equipped with number of sensors gathering a lot of environmental data about device and user context. This triggered an apparent need for complex application running on mobile platforms. In number of situations such platforms have become viable alternatives for classic ones, e.g. PC. Therefore, engineering of software on mobile platforms uses similar tools and techniques to regular software engineering. This also includes methods from the domain of knowledge engineering, since real-time processing of large numbers of data needed to program complex mobile applications requires intelligent techniques.

Rules, are one of the most important techniques in knowledge engineering. Rule-based tools, including rule engines are a first class citizen in number of intelligent software systems developed on PC platforms, server and cloud. Therefore, there has been a growing demand to use rule-based tools on mobile platforms. This turns out to be not a trivial issue. In fact classic rule engines are very hard to be ported, due to either of the source code base (e.g. CLIPS/Jess) or runtime requirements (e.g. Drools). This gives motivation to investigate opportunities to port or develop a mobile rule engine. In this

paper we present the results of such consideration and discuss a specific case study of the HeaRT [1] rule engine developed by us.

The rest of the paper is organized as follows: In Sect. 2 related works is discussed along with the detailed motivation for our research. Then in Sect. 3 we specify requirements for porting an inference engine to a mobile environment. Based on them we discuss possible scenarios in Sect. 4. In Sec. 5 we focus on porting HeaRT to a mobile environment. The paper ends with the summary in Sect. 6.

2 Related works and motivation

In recent years, a lot of development was devoted to build applications that use mobile devices to monitor and analyze various user contexts.

The SocialCircuits platform [2] uses mobile phones to measure social ties between individuals, and uses long- and short-term surveys to measure the shifts in individual habits, opinions, health, and friendships influenced by these ties. Jung [3] focused on discovering social relationships between people. He proposed an interactive approach to build meaningful social networks by interacting with human experts, and applied the proposed system to discover the social networks between mobile users by collecting a dataset from about two millions of users. Given a certain social relation (e.g., isFatherOf), the system can evaluate a set of conditions (which are represented as propositional axioms) asserted from the human experts, and show them a social network resulted from data mining tools. Sociometric badge [4] has been designed to identify human activity patterns, analyze conversational prosody features and wirelessly communicate with radio base-stations and mobile phones. Sensor data from the badges has been used in various organizational contexts to automatically predict employee's self-assessment of job satisfaction and quality of interactions. Eagle and Pentland [5] used mobile phone Bluetooth transceivers, phone communication logs and cellular tower identifiers to identify the social network structure, recognize social patterns in daily user activity, infer relationships, identify socially significant locations and model organizational rhythms.

Besides research projects, there exist also a variety of application that are used for gathering information about context from mobile devices, like SDCF [6], AWARE [1], JCAF [7], SCOUT [8], ContextDriod [9], Gimbal [2]. These are mostly concerned with low-level context data acquisition from sensors, suitable fur further context identification. On the other hand, they do not provide support nor methodology for creating complex and customizable context-aware systems.

Although there is a lot of frameworks and middlewares developed for context-aware systems, they are usually limited to a specific domain and designed without taking into consideration mobile platforms. Examples include CoBrA [10] and SOUPA [11] for building smart meeting rooms, GAIA [12] for active spaces, Context Toolkit [13].

There is still space for research in a field of lightweight context modeling and context reasoning targeted at mobile devices. Some attempts were made to develop such

[1] `http://www.awareframework.com/`
[2] `https://www.gimbal.com/`

frameworks, like SOCAM [14], or Context Torrent [15]. There were also attempts to migrate existing rule engines to the mobile platforms. In our preliminary approach we investigated several candidates for the mobile reasoning engines, including: Jess [3], Context Toolkit [13], ContextDroid [16] and Context Engine [17]. Although some of them were successfully migrated and launched on the Android device, none of these solution fully supported the requirements that we believe are crucial for mobile computing with respect to the context-based reasoning (see Section 3).

Taking all the above into consideration, our primary motivation for the work presented in this paper was to:

- investigate possible scenarios for migrating existing rule-based engines to mobile platforms,
- define requirements that all the inference engines should meet in order to provide efficient, lightweight context-based reasoning on mobile platforms.

In our research we focused on the rule-based reasoners, because they are characterized by the high intelligibility [18] capabilities, which is a crucial factor in designing user-centric applications. Rule-based approach provides also very efficient reasoning, and intuitive modeling languages that help the end-user to understand the system and cooperate with it. The work presented in this paper is a continuation of our previous attempt of migration HeaRT rule-based engine to Android platform, which was successful only in a small fraction [19]. We managed to migrate rule-based reasoning engine to mobile platform, however the efficiency of this migrated version was far worse than the desktop one. Therefore, we decided to perform an extensive case study on the possible migration scenarios, which will allow us to define universal requirements for the mobile reasoning engine. The following section describes in details these, and shows which of them can be fulfilled by the existing solutions.

3 Requirements for mobile inference engine

Mobile devices are now dominated by Android and iOS operating systems. However to design a fully portable reasoning engines other systems should also be taken into consideration. The portability of the software between different mobile platforms is however not a trivial task. Therefore, to support the process of efficient migration of the inference engines to mobile environments, the following requirements were identified [20]:

- **Portability (R1)**. The inference engine should be portable and as independent of the operating system as possible.
- **Responsiveness (R2)**. The inference layer has to work under soft real-time constraints. Mobile environment is highly dynamic, and the inference layer should follow rapid changes of context in such an environment.
- **Resource limitation (R3)**. It should consume as least resources as possible to work transparently in the background.

[3] http://www.jessrules.com

– **Privacy (R4)**. The reasoning service should not send any confidential data to the external servers, but perform all the reasoning locally.
– **Robustness (R5)**. It should work properly when the contextual data is incomplete or uncertain. This requirement is beyond of the scope of this article and is just briefly described in the following paragraphs.
– **Intelligibility (R6)**. It should be able to explain its decision to the user, and thus improve intelligibility of the system.

Due to the development of programming languages which execution is performed by the virtual machines, it became possible to successfully migrate existing rule-based inference engines directly onto different mobile platforms. For example Java bytecode can be executed not only on Android operating system, which supports it natively, but also on iOS and other platforms that provides appropriate Java Virtual Machine implementations. This may give a misleading impression that the requirement (R1) could be easily fulfilled. Our research shown that this is not true, as the quality of the virtual machines is not always satisfactory and hence, although portability is possible it may indirectly affect fulfillment of other requirements. For instance, if the rule-based engine is written in a programming language that is not natively supported by the mobile platform, but depends on the virtual machine that allows executing it, the responsiveness (R2) of such system may be affected by the inefficient implementation of the virtual machine. What is more, depending on the complexity of the inference engine, such migration may not always fully succeed, especially if the inference engine is based on the technology that is not supported by the mobile implementation of the virtual machine (i.e. Hibernate [4] for knowledge management in Drools and Context Toolkit). Some technologies even though possible to migrate to mobile platform, loses their efficiency, due to limited memory or CPU resources on the mobile device. Thus, the (R1) requirement should be achieved by choosing a programming language that is equally and efficiently supported by all the available mobile platforms. Currently to our knowledge there is such solution available.

Responsiveness (R2) of the system is usually affected by the efficiency of its implementation. Hence, this requirement is very tightly bound with (R3), which states that the mobile inference engine should consume as low CPU and memory resources as possible . Complex rule-based solutions like Drools or ContextToolkit although successfully launched on the Android mobile devices, were very inefficient as they depend on the parts of the JVM that is not supported by Dalvik Virtual Machine present in Android. The very important issue that tackles the problem of resource limitation is connected with energy consumption. Most of the sensors which are primary information providers about the environment, when turned on all the time, decrease the mobile device battery level very fast. This reflects on usability of the system and ecological aspects regarding energy saving. Currently most of the solutions (including these that were designed for mobile platforms like ContextEngine) rely on the built in mechanisms for sensor energy consumption.

Having in mind requirement (R3), the reasoning engines that make use of this information should be designed to best manage the tradeoff between high quality, accurate

[4] `http://hibernate.org/orm/`

information form the sensors and energy efficiency [21]. Such a design of a reasoning engine requires on the other hand a mechanism that could cope with uncertain and incomplete knowledge. This can be guaranteed by fulfilling the requirement (R5). It is strictly connected with the nature of the mobile environment. Although the mobile devices are currently equipped with a variety of sensors, the contextual data provided by them is not always complete nor certain. For instance the location provider will not work properly underground, where there is no access to the GPS or the Internet. Currently only a ContextDroid (former SWAN) tackles this issue by introducing expiration time and temporal reasoning.

Another requirement that indirectly is connected with energy efficiency and therefore tackles the requirement (R3) is privacy requirement (R4). To improve energy efficiency of the intelligent mobile system, the most time and energy consuming tasks could be performed externally by the reasoning server. This however requires from the users to agree to share their private contextual information with the service provider, which in some countries like European Union is regulated by the law and not easily accessible. What is more, most of the users do not want to send information about their location, activities, and other private data to external servers. Hence, the context reasoning should be performed locally by the mobile device. This requirement was also one of the first motivations for our attempts to migrating reasoning engines to a mobile devices so they could work as a local inference services for other applications. Although not explicitly stated, all of the existing engines rely on the local knowledge bases. Even Drools and ContextToolkit that allows storing knowledge in remote database can be adapted to use only local storage.

Requirement (R6) on the other hand is the main reason why we decided to use rule-based engines. Besides very efficient reasoning mechanisms, rule-based systems in most cases fulfill the (R6) requirement by the definition. Most of the existing solutions have an explanation mechanism, which allows to trace back the inference chain and therefore explain the user what was the causes, conclusions and actions that was taken based on the previous both. The Context Toolkit s the only solution that provides intelligibility support with a special module designed for this. However, our attempts at migrating it to the mobile environment failed due to the problems in parsing module of Context Toolkit.

4 Possible scenarios

One of the biggest challenges for the mobile software creators is the large variety of technologies used in the mobile operating systems. Due to the more tightly coupled runtime environment than the one met in the traditional desktop operating system, a programmer has a very narrow choice of programming tools and languages. To make things even worse, every one of the leading mobile platforms promotes its own ecosystem incompatible with any other. Therefore it is not surprising that the industry is constantly looking for the cross-platform solutions. This section will concern possible scenarios of the mobile inference engine development, taking into account issues mentioned in this paragraph.

In the rest of the article we will focus on the three most popular platforms, namely: Android, iOS and Windows Phone. Despite this, most of the article applies also to Firefox OS which supports only the web technologies. There exist also a few Linux based operating systems, supporting software written for the desktop platforms — many of them, like Sailfish OS, are trying to reuse the Android ecosystem.

Table 1. Technologies supported by the most popular mobile platforms..

Technology / OS	Android	iOS	Windows Phone
JVM based	**Native.**	One-way interface only through non-standard Java Virtual Machines.	None.
C/C++	Two-way interface using Android NDK.	**Native.**	Two-way interface starting from WP 8.
Objective-C	None.	**Native.**	None.
CLR based	One-way interface through Mono.	One-way interface through Mono.	**Native.**
JavaScript	New devices include V8 JavaScript engine.[5]	JavaScriptCore[6] support starting from iOS 7. Many independent solutions.	Includes proprietary JavaScript engine.
Lua	Many implementations, including LuaJIT.	Supported without JIT.	Supported without JIT.

4.1 Supported technologies

Before presenting the proposed approaches, we will briefly summarize the mobile programming toolset available for the developers. Table 1. contains short summary of supported languages on different platforms. However, it needs some explanation carried out in the next three paragraphs.

Firstly, we should explain our choice of the technologies presented in the table. JVM, CLR and C-family were obvious choices as the native technologies on some platform — their use would simplify implementation at least on the one family of devices. On the other hand, JavaScript and Lua[7] are presented because of their small and easily accessible runtimes, what makes them perfect candidates for the cross-platform embedded solutions. Lack of the JIT support for Lua means that we cannot use the LuaJIT[8] —

[5] `https://code.google.com/p/v8/`
[6] `http://trac.webkit.org/wiki/JavaScriptCore`
[7] `http://www.lua.org`
[8] `http://luajit.org`

a very fast Lua implementation. There are of course other technologies missing from the table, but we were looking only for the tested and production ready implementations.

Secondly, "one-way interface support" means that we can build an application in the selected language, but the code written in it cannot be directly called within a native application. For example: programmer can write an entire application using Mono[9] on iOS, but can not use library written in C# within native application.[10] These inconveniences render Mono as not suitable for our purposes. JVM based languages share the same problem — necessity of running JVM on the system dramatically reduces its possibilities to reuse on the other platforms.

Finally, despite the shared support for the C-family languages, we cannot freely share code written in C/C++ between all these platforms. In fact there are many differences mainly due to different compilation toolchains and standard libraries included in the system — for example Android uses its own Bionic C standard library. Furthermore, while on iOS the C/C++ code can be freely mixed with Objective-C, both Android and Windows Phone need a foreign function interface to call the C functions from Dalvik/CLR virtual machines. Lately it became possible to compile Objective-C code into Android assembler, but it is a very immature project. To conclude, shared C codebase is possible, but we should not expect it will work on the all systems without any platform-dependent fixes.

4.2 Proposed solutions

Leaving aside for the moment technical differences between the mobile platforms we can distinguish different approaches to the cross-platform software development on the basis of the code base architecture, whether it is fully or only partly shared between supported platforms. Particularly there can be listed three border cases:

- **Separated code bases (S1)** — every platform has its own independent code base.
- **Shared code base (S2)** — all platforms share the same code base.
- **Hybrid approach (S3)** — there exist elements of project implemented especially (particularly API interface) for the different platforms, but we can also distinguish shared parts of code.

In this section we will confront all these approaches with technical characteristic of mobile operating systems presented in the previous section.

Foremost, according the the S1 approach, we can simply have separate code bases for every platform. In spite of the obvious disadvantages of this approach, which include larger support and development costs, this is a perfectly sane solution for the projects aiming mainly for the high quality implementation for the one selected platform and less supported ports for the other systems. Moreover, this way we can take advantage of the features characteristic for the different runtimes; in the context of the mobile libraries it has a great impact on the quality of API available for the programmers. Therefore this

[9] `http://www.mono-project.com/Main_Page`

[10] It is not exactly true. Linking Mono code to the native iOS application is possible but troublesome and not production ready.

is a very popular approach used for so called Software Development Kits provided by such companies like Google or Facebook.

On the other hand, when we are interested in supporting possible many (maybe even not known in the beginning of the project) environments, the costs related to the code maintenance can be overwhelming. S2 seems to be an universal solution for these problems, however we must remember that there is always a trade-off between universality and expressiveness of the resulting code — programmer is forced to abstract from the platform dependent features. From the technical point of view there are two distinct methods to achieve the shared code base on mobile devices. In the first scenario we could use a technology support on every platform, we have interest in. Again, in the context of table 1 we could distinguish two possible cases presented below.

Low level approach (S2LL). Most modern operating systems support some kind of the low level interface to the C-family languages. C/C++ code base has many advantages — first of all it has a great performance and, secondly, it is very popular among programmers, so there should not be a problem with the reuse of the existing solutions and libraries. Unfortunately, as it was indicated before, there are also significant differences between the low level characteristics of the operating systems — they differ in used standard libraries and methods of the code calling from the native platform language. These differences eventually lead to the corner cases maintained by platform specific code contained in the conditionally compiled sections of project. Second, the more controversial disadvantage of the low level code is that is harder to maintain than its higher level counterparts. Due to the efficiency reasons, the low level approach is widely adapted by the game development industry. The most popular examples include cocos2d-x[11] and Marmalade[12] libraries.

High level approach (S2HL). Nowadays there can be observed the rapid growth of dynamic languages; particularly web technologies like JavaScript are regarded as a new kind of a portable assembler language. Thanks to the efficient, highly optimized and widely available runtime it recently became the most popular platform for the cross-platform applications. Lua is an other, less known language with similar features — lower popularity of this platform is compensated with more sane language characteristic and even smaller runtime, ported already to the most exotic hardware platforms. Unlike in C-family languages, there should be no need for the platform-dependent sections of code in project written using high level languages. Moreover, distribution of the dynamic code is much easier — it does not need the compilation step and can be distributed as plain text through the Internet. The main concern about high level approach regards its efficiency; dynamic code tends to be slower, however due to the rapid growth of the technology stack, the gap between low level and high level code successfully decreases. Given these facts, it should be not surprising that currently there appear more and more high level frameworks, which could be used to create the mobile applications from scratch (for example PhoneGap[13] or Sencha Touch[14]) including even more de-

[11] `http://www.cocos2d-x.org/`
[12] `https://www.madewithmarmalade.com/`
[13] `http://phonegap.com`
[14] `http://www.sencha.com/products/touch`

manding products like game engines (for example LÖVE[15] and Loom[16], both written in Lua).

The second scenario concerns the more complicated **compiling toolchain (S2CT)** — precisely, before the deployment of the project, we could translate it into the native platform technology. This way we could work on the shared code base without worrying about differences between the platforms. Recently this approach receives a lot of attention — there are many technologies treating C or JavaScript as the portable assembler languages. Especially the latter, given its easy deployment on the web, is currently the compile target of the almost every popular programming language, including even the low level technologies. While making C or JavaScript our compile target does not differ much from using them as the project main language, in the most optimistic scenario we could translate the code base into different language depending on the target platform. This way we could maintain portability of the project and also benefit from native API on every supported platform at the same time. Remarkably, the Haxe[17] language can be compiled into JavaScript, C++, C# and Java, trying to make this optimistic scenario possible.

Given the advantages of both S1 and S2 approaches, the hybrid approach simply combines them through the greater modularization of project. While the core black box part of the project can be written in the portable way, possibly using S2HL or S2LL methods, the interface layer should be interchangeable and written in regard to each supported platform separately. Consequently core of the project can truly abstract from platform dependent code (as in the optimistic cases of S2) and it is possible to make a high quality API for the end-users (as in S1). This approach is widely used by the so called wrappers — libraries written in the high level technology, which only wrap the low level library written already in C or other efficient technology. The proposed architecture of the inference engine implemented using the hybrid approach is presented in the figure 1, where the Platform Independent Runtime can mean low level machine programmable using C-family language or some kind of virtual machine destined to run the high level code, for example Lua interpreter or V8 JavaScript engine. The API part of the architecture should contain an query interface and methods to load rules from their plain representation.

5 HeaRT inference engine on mobile platforms

This section will focus on the case of porting the HeaRT inference engine [1] to the mobile platforms. HeaRT is a lightweight rule-based inference engine that uses XTT2 [22] notation for knowledge representation [23].

The first part of the section will briefly describe our efforts to port existing code to the Android ecosystem, which was believed to be a good platform for the "proof of concept" implementation. The second part will contain plans for the future work, based on the experience gained from the unsuccessful attempts and research results presented earlier in section 4.

[15] `https://love2d.org/`
[16] `https://www.loomsdk.com/`
[17] `http://haxe.org`

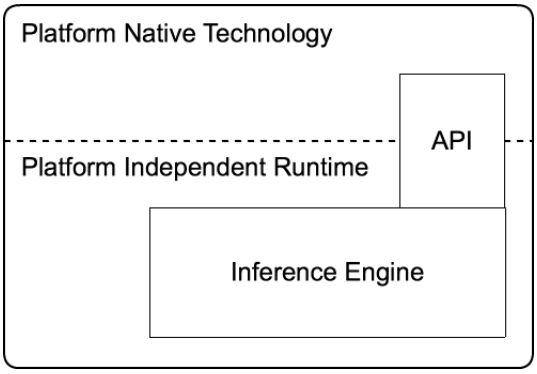


Figure 1. Proposed architecture of the portable interface engine using the hybrid approach.

5.1 Previous attempts

The original implementation of HeaRT was written in the Prolog language and was executed by SWI–Prolog, an open source ISO-compliant Prolog implementation[18]. Thanks to the Prolog metaprogramming and reasoning features it was easy to represent rules as simple logical clauses and querying could be expressed with standard Prolog queries. Therefore the first attempts to port the engine were focused mainly on possible ways of executing Prolog code on the mobile platforms, particularly Android as it was stated in previous paragraph. The detailed description of these efforts can be found in [19]. In the terminology of Section 4.2 all attempts presented below can be assigned to the S2 category.

First and conceptually the simplest attempt can be categorized as S2LL and consisted of porting the SWI–Prolog environment to the Android platform. Due to the source code written almost entirely in C it was hoped to succeed without breaking changes in the environment. Unfortunately, it could not be carried successfully mainly because of the two major issues:

- SWI–Prolog contains large parts of platform dependent code, dealing with the low level facilities like threading support, console interface, etc. Due to the lack of the libraries and different standard library on Android, migration of the SWI–Prolog must involve further refactoring and new conditionally compiled lines of code.
- The interface between Java and SWI–Prolog was found to be not satisfactory, particularly it was not clear, whether it would work with the Dalvik Java virtual machine.

Next attempt was supposed to leave aside issues connected with the low level character of the popular Prolog environments and was oriented on their pure Java counterparts (therefore it can be regarded as a S2HL type). Following the carefully carried research four possible candidates were proposed: tuProlog[19], jekejeke Prolog[20], Gnu

[18] `http://www.swi-prolog.org/`
[19] `http://apice.unibo.it/xwiki/bin/view/Tuprolog/`
[20] `http://www.jekejeke.ch/idatab/doclet/intr/en/docs/package.jsp`

Prolog for Java[21] and jinniprolog[22]. Despite their supposed ISO-compliance, none of them could be really regarded as a fully working Prolog enviroment. Nevertheless, after large refactoring of HeaRT source code we were finally able to run it using the tuProlog environment. Unfortunately, performance of this solution was unacceptable, clearly not satisfying the R2 and R3 requirements presented in Section 3.

The last used technique concerned translation of Prolog code into Java (therefore it was representative of the S2CT approach). Unfortunately, evaluated Prolog Café translator[23], in spite of efficiency, lacked many advanced features used broadly in the HeaRT code, explicitly it dose not support mixing of the dynamic and static predicates. Furthermore, the text representation of HeaRT rules being itself a valid Prolog code, was also needed to be compiled to Java class, making the dynamic loading of the rules very inflexible.

To sum up the foregoing, Prolog proved to be too unpopular language to have adequate running environments on the mobile platforms. Consequently, all approaches of S2 type have to be preceded by the creation (or adaption) of efficient and portable Prolog environment. Due to related development cost, we have currently resigned from this approach.

5.2 Future plans

The current plans for the mobile HeaRT development were created according to the S3 approach. The inference elements of HeaRT will be rewritten in the Lua language — the main reasons of this choice are low development cost and large variety of compact and efficient runtimes on even very exotic hardware. The architecture of so called LuaHeaRT matches the one presented in Figure 1. There were identified three major drawbacks of the selected approach:

1. The source code must be completely rewritten into language with different paradigm and capabilities. The related costs are not negligible, but inevitable.
2. Text representation of the rules in HeaRT is also a Prolog code, therefore in original implementation the task of processing it was performed entirely by SWI–Prolog. In the new architecture this process will be divided into two parts: parsing realized by parser included in the API module and semantic analysis performed entirely by the inference engine. Thanks to the formal grammar of the rule format it will be possible to semi-automatically generate parsers for the different platforms.
3. In the desktop version of HeaRT queries are defined as the standard Prolog queries. For the sake of the new implementation we must specify new, possibly portable method of creating queries. The related code should be automatically generated (similarly to the grammar parser) and included in the API module. On the other hand the inference engine must contain some kind of low level query API, which could be the target of the human friendly representation of queries.

[21] `http://www.gnu.org/software/gnuprologjava/`
[22] `http://code.google.com/p/jinniprolog/`
[23] `https://code.google.com/p/prolog-cafe/`

For the aforementioned reasons, there is no doubt that the portable implementation of HeaRT according to the S2 approach will involve the overall redesign of the desktop version. However, it is not entirely bad phenomenon, the end effects can be highly beneficial for the future development of the engine.

6 Summary

In recent years, a lot of development was devoted to build applications that make use of contextual information to behave in an intelligent way. Due to the evolution of mobile devices which became omnipresent in every human life, the context-aware application became also one of the primary focus of mobile system developers. However, mobile and desktop environments are different, and therefore migrating solutions between them is not a trivial task. In this paper we focused on the problem of migration of a rule engine to mobile platform.

We defined requirements that every mobile reasoning engine should fulfill to provide efficient reasoning solution. We confronted these requirements with an attempt to migrate Prolog-based rule engine HeaRT onto the mobile platform. Finally, we presented a study of different migration scenarios, and propose a solution that provides both efficiency, portability and allows for effective source code maintenance.

References

1. 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
2. Chronis, I., Madan, A., Pentland, A.S.: Socialcircuits: the art of using mobile phones for modeling personal interactions. In: Proceedings of the ICMI-MLMI '09 Workshop on Multimodal Sensor-Based Systems and Mobile Phones for Social Computing. ICMI-MLMI '09, New York, NY, USA, ACM (2009) 1:1–1:4
3. Jung, J.J.: Contextualized mobile recommendation service based on interactive social network discovered from mobile users. Expert Syst. Appl. **36** (2009) 11950–11956
4. Olguin, D., Waber, B.N., Kim, T., Mohan, A., Ara, K., Pentland, A.: Sensible organizations: Technology and methodology for automatically measuring organizational behavior. IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS-PART B: CYBERNETICS (2009) 43–55
5. Eagle, N., (Sandy) Pentland, A.: Reality mining: sensing complex social systems. Personal Ubiquitous Comput. **10** (2006) 255–268
6. Atzmueller, M., Hilgenberg, K.: Towards capturing social interactions with sdcf: An extensible framework for mobile sensing and ubiquitous data collection. In: Proc. 4th International Workshop on Modeling Social Media, ACM Press (2013)
7. Bardram, J.E.: The java context awareness framework (JCAF) – a service infrastructure and programming framework for context-aware applications. In Gellersen, H.W., Want, R., Schmidt, A., eds.: Pervasive Computing. Volume 3468 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2005) 98–115
8. Woensel, W.V., Casteleyn, S., Troyer, O.D.: A Framework for Decentralized, Context-Aware Mobile Applications Using Semantic Web Technology. (2009)
9. van Wissen, B., Palmer, N., Kemp, R., Kielmann, T., Bal, H.: ContextDroid: an expression-based context framework for Android. In: Proceedings of PhoneSense 2010. (2010)
10. Chen, H., Finin, T.W., Joshi, A.: Semantic web in the context broker architecture. In: PerCom, IEEE Computer Society (2004) 277–286

11. Chen, H., Perich, F., Finin, T.W., Joshi, A.: Soupa: Standard ontology for ubiquitous and pervasive applications. In: 1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge, MA, USA, IEEE Computer Society (2004) 258–267
12. Ranganathan, A., McGrath, R.E., Campbell, R.H., Mickunas, M.D.: Use of ontologies in a pervasive computing environment. Knowl. Eng. Rev. **18** (2003) 209–220
13. Dey, A.K.: Understanding and using context. Personal Ubiquitous Comput. **5** (2001) 4–7
14. Gu, T., Pung, H.K., Zhang, D.Q., Wang, X.H.: A middleware for building context-aware mobile services. In: In Proceedings of IEEE Vehicular Technology Conference (VTC. (2004)
15. Hu, H., of Hong Kong, U.: ContextTorrent: A Context Provisioning Framewrok for Pervasive Applications. University of Hong Kong (2011)
16. Palmer, N., Kemp, R., Kielmann, T., Bal, H.: Swan-song: A flexible context expression language for smartphones. In: Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones. PhoneSense '12, New York, NY, USA, ACM (2012) 12:1–12:5
17. Kramer, D., Kocurova, A., Oussena, S., Clark, T., Komisarczuk, P.: An extensible, self contained, layered approach to context acquisition. In: Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing. M-MPAC '11, New York, NY, USA, ACM (2011) 6:1–6:7
18. Dey, A.K.: Modeling and intelligibility in ambient environments. J. Ambient Intell. Smart Environ. **1** (2009) 57–62
19. Szymon Bobek, Grzegorz J. Nalepa, M.S.: Challenges for migration of rule-based reasoning engine to a mobile platform. In Dziech, A., Czyżewski, A., eds.: Multimedia Communications, Services and Security. Volume XX of Communications in Computer and Information Science., Springer Berlin Heidelberg (2014) accepted.
20. Nalepa, G.J., Bobek, S.: Rule-based solution for context-aware reasoning on mobile devices. Computer Science and Information Systems **11** (2014) 171–193
21. Bobek, S., Porzycki, K., Nalepa, G.J.: Learning sensors usage patterns in mobile context-aware systems. In: Proceedings of the FedCSIS 2013 conference, Krakow, IEEE (2013) 993–998
22. Nalepa, G.J., Ligęza, A., Kaczor, K.: Formalization and modeling of rules using the XTT2 method. International Journal on Artificial Intelligence Tools **20** (2011) 1107–1125
23. Ligęza, A., Nalepa, G.J.: A study of methodological issues in design and development of rule-based systems: proposal of a new approach. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery **1** (2011) 117–137

Knowledge Modeling with the Open Source Tool myCBR

Kerstin Bach[1], Christian Sauer[2], Klaus Dieter Althoff[3], and Thomas Roth-Berghofer[2]

[1] Verdande Technology AS
Trondheim, Norway
http://www.verdandetechnology.com
[2] School of Computing and Technology
University of West London, United Kingdom
http://www.uwl.ac.uk
[3] Competence Center Case-Based Reasoning (CC CBR)
German Research Centre for Artificial Intelligence, Kaiserslautern, Germany
http://www.dfki.de/web/competence/cccbr

Abstract. Building knowledge intensive Case-Based Reasoning applications requires tools that support this on-going process between domain experts and knowledge engineers. In this paper we will introduce how the open source tool *myCBR 3* allows for flexible knowledge elicitation and formalisation form CBR and non CBR experts. We detail on *myCBR 3*'s versatile approach to similarity modelling and will give an overview of the Knowledge Engineering workbench, providing the tools for the modelling process. We underline our presentation with three case studies of knowledge modelling for technical diagnosis and recommendation systems using *myCBR 3*.

1 Introduction

Case-Based Reasoning (CBR) is a methodology introduced by Riesbeck and Schank [13] and Kolodner [8] who derived its basic principles from cognitive science. They describe how humans manage and reuse their experience described in episodes. Aamodt and Plaza [2] introduce a basic model for developing CBR applications. It consists of four processes: Retrieve, Reuse, Revise and Retain. The CBR process requires cases that consist of problem and solution description. Problem descriptions are usually attributes values describing a problematic or critical situation while the solution contains information on how to solve the given problem. In the retrieve phase, the attributes describing a problem are matched against cases in a case base. The best n cases are returned. In order to match a given situation these cases can be adapted (Reuse). In the revision phase, reused cases are verified before they are retained.

CBR systems always carry out the retrieve phase which is characterized by a similarity-based comparison of features, while the remaining phases can omitted. Richter [12] introduced to model of four knowledge containers describe the required knowledge within a CBR system:

– *Vocabulary* defining the range of allowed values for attributes. For numeric values this is usually the value range (minimum, maximum) while for symbolic values this can be a list of values.
– *Similarity Measures* defining the relationship between attribute values in form of a similarty assignments. Similarity measures can be formulas like the hamming distance for numeric values or reference tables for symbolic values.
– *Adaptation Knowledge* is knowledge describing how cases can be adapted in the reuse step, often represented as rules.
– *Cases* are instances describing situations that have happened and are worth capturing in order to be reused. They instantiate attributes describing the problematic situation as well as a solution description. Their degree of formalization can vary.

Developing CBR systems requires a systematic development of knowledge models by defining the requirements and building the models itself. *myCBR 3*[4] is an open source tool targeting at developing customized knowledge models with an emphasis on vocabulary and similarity measure development. *myCBR 3* is an open-source similarity-based retrieval tool and software development kit (SDK). With *myCBR 3* Workbench you can model and test highly sophisticated, knowledge-intensive similarity measures in a powerful GUI and easily integrate them into your own applications using the *myCBR 3* SDK[3]. Case-based product recommender systems are just one example of similarity-based retrieval applications.

In the remaining of this paper we will give an overview of other CBR tools and applications (section 2) as well as showcase the functionalities of *myCBR 3* (section 3). In section 4 we will show how *myCBR 3* has been applied in different CBR projects while the final section will sum up the paper and give an outlook on future work on the tool.

2 Related Research

Freely available CBR tools are for instance FreeCBR, jCOLIBRI or eXiT*CBR, which will be briefly discussed in this section. FreeCBR[5] is a rather simple CBR engine, which allows the realization of basic CBR features. However, it does not cover features like case revision or retention and more individualized knowledge models, or comprehensive global and local similarity measures, are not applicable either. Further, it still requires quite some effort to apply it to a high variety of tasks. jCOLIBRI started from a task oriented framework also covering distributed reasoning [10], recently jCOLIBRI Studio [11] for more comprehensive support of building CBR knowledge has been introduced. Up to today jCOLIBRI includes more machine learning and semantic web features while *myCBR 3* focused on the knowledge required in the knowledge containers.

[4] http://www.mycbr-project.net
[5] `http://freecbr.sourceforge.net/`

COLIBRI is another platform for developing Case-Based Reasoning (CBR) CBR software. COLIBRI's main goal, opposed to *myCBR 3*, is to provide the infrastructure required to develop new CBR systems and its associated software components, rather than a CBR knowledge model. COLIBRI is designed to offer a collaborative environment. It is an open platform where users can contribute with different designs or components of CBR systems, which will be reused by other users. Subsequently many of the components available have been developed by third-party research groups and contributed to the platform to be shared with the community.

As a platform, COLIBRI offers a well-defined architecture for designing CBR systems. COLIBRI also provides a reference implementation of that architecture: the jCOLIBRI framework. jCOLIBRI is a white-box tool that permits system designers to have total control of the internal details of the software. The platform also includes graphical development tools to aid users in the development of CBR systems. These tools are enclosed in the COLIBRI Studio IDE and generate applications that use the components provided by jCOLIBRI.

Furthermore, creating individualized case representations and especially flexible similarity measures is the strength of *myCBR 3*. eXiT*CBR has also its roots in machine learning applications and is specialized for medical diagnosis tasks [9]. It has recently been extended in order to cope with more than one case base. In comparison to *myCBR 3*, the ideas behind the methodology also differ, since we are focusing on the knowledge container model rather than the machine-learning-related tasks. The integration of Drools in an existing framework for executing rules on a given corpus has been introduced by Hanft et al. [7]. In this paper Drools has been integrated in an existing OSGi environment. The approach presented here required a more comprehensive customization since *myCBR 3* was not embedded in OSGi and the requirements for the rules differed in terms of usable knowledge and modification of cases.

In industry, most prominent CBR tools or CBR related technologies are used by empolis in the on SMILA[6] based Information Access Suite[7] as well as by Verdande Technology in DrillEdge [8] [6] . The Information Access Suite has been applied in various help-desk scenario applications as well as in document management while DrillEdge focuses on predictive analytics in oil well drilling. Both companies run proprietary implementations based on academic software - CBR-Works [18] and Creek [1] respectively.

3 Knowledge Engineering in myCBR

myCBR 3 is an open-source similarity-based retrieval tool and software development kit (SDK)[19]. With *myCBR 3* Workbench you can model and test highly sophisticated, knowledge-intensive similarity measures in a powerful GUI and easily integrate them into your own applications using the *myCBR 3* SDK.

[6] `https://www.eclipse.org/smila/`
[7] `http://www.empolis.com`
[8] `http://www.verdandetechnology.com`

Case-based product recommender systems are just one example of similarity-based retrieval applications.

The *myCBR 3* Workbench provides powerful GUIs for modelling knowledge-intensive similarity measures. The Workbench also provides task-oriented configurations for modelling your knowledge model, information extraction, and case base handling. Within the Workbench a similarity-based retrieval functionality is available for knowledge model testing. Editing a knowledge model is facilitated by the ability to use structured object-oriented case representations, including helpful taxonomy editors as well as case import via CSV files.

The *myCBR 3* Software development Kit (SDK) offers a simple-to-use data model on which applications can easily be built. The retrieval process as well as the case loading, even from considerably large case bases, are fast and thus allow for seamless use in applications built on top of a *myCBR 3* knowledge model.

Within *myCBR 3* each attribute can have several similarity measures. This feature allows for experimenting and trying out different similarity measures to record variations. As you can select an appropriate similarity measure at runtime via the API, you can easily accommodate for different situations or different types of users.

The *myCBR 3* Workbench is implemented as using the Rich Client Platform (RCP) of Eclipse and offers two different views to edit either knowledge models or case bases. In this section we will focus on the modelling view as shown in 1.

The conceptual idea behind the modelling view is that first a case structure is created, followed by the definition of the vocabulary and the creation of individual local similarity measures for each attribute description (eg. CCM in 1) followed by the global similarity measure for a concept description (Car in 1).

The modelling view of the *myCBR 3* Workbench (see figure 1) is showing the case structure (left), available similarity measures (left bottom) and their definition (center). Modelling the similarity in the Workbench takes place on the attribute level for local similarity measures and the concept level for global similarity measures.

3.1 Building a Vocabulary

The vocabulary in *myCBR 3* consists of concepts and attributes. A concept description can contain one or more attribute descriptions as well as attributes referencing concepts, which allows the user creating object-oriented case representations. In the current version *myCBR 3* also allows for the import of vocabulary items, e.g. concepts and attributes, from CSV files as well as from Linked (Open) Data (LOD) sources.

An attribute description can have one of the following data types: Double, Integer, String, Date and Symbol. When attributes are defined, the data types and value ranges are given with initial default values and can be set to the desired values in the GUI.

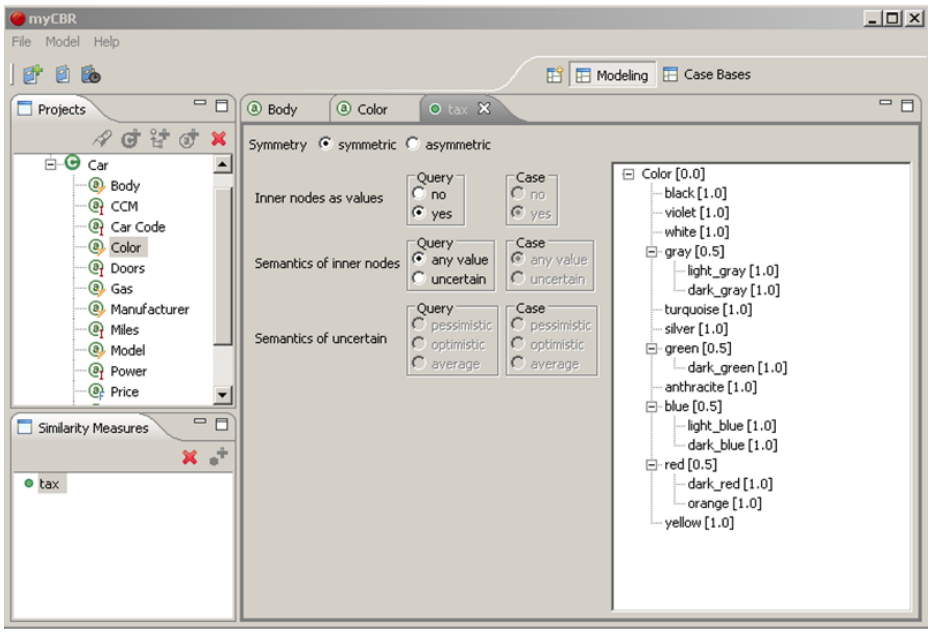


Fig. 1. Example view of the knowledge model view in the *myCBR 3* workbench

3.2 Building Similarity Measures

The Workbench provides graphically supported modelling of similarity functions that support their definition. As an attribute description can have more than one similarity measure experimenting with knowledge modelling approaches is facilitated. For numerical data it is providing predefined distance (or similarity) functions along with predefined similarity behaviour (constant, single step or polynomial similarity decrease). For symbolic values, *myCBR 3* Workbench provides table functions and taxonomy functions. A table function allows defining for each value pair the similarity value, while a taxonomy subsumes similarity values for subsets of values. Depending on the size of a vocabulary, table similarity measures are hard to maintain and taxonomies allow an easier overview. For symbolic values, also set similarities are provided in order to compare multiple value pairs. For each of the similarity measures, as well as for the global similarity measure(s) a specific, versatile editor GUI is provided.

4 Case Study

4.1 Creating Knowledge from Unstructured Documents

This approach has been developed in machine diagnosis based on experiential knowledge from engineers[4]. Most vehicle companies provide service after delivering their machines to customers. During the warranty period they are able

to collect data about how and when problems occurred. They take this data for improving vehicles in different ways: collected data can go back in the product development process, it can be used for improving diagnostic systems to repair them at dealerships or in the factory and also educating service technicians repairing vehicles abroad. This is extremely important if vehicles cannot easily be taken back to factory, e.g. services for aircrafts or trucks.

Such machine manufacturers collect information about machine problems that are submitted by technicians containing machine data, observations and sometimes further correspondence between an engineer or analyst with the technician at the machine. In the end, these discussions usually come up with a solution - however, the solution is normally neither highlighted nor formalized and the topics and details discussed highly depend on the technician and what the Customer Support asks for. That is the reason why cases that are stored for collecting Customer Support information can not directly be used for CBR. Therefore we will differentiate between Customer Support Cases (CS Cases) and CBR Cases. CS Cases contain original information collected during a discussion between Customer Support and the field technician, while a CBR Case contains only relevant information to execute a similarity based search on the cases. The CBR cases content represents each CS Case, but contains machine understandable information.

For building the vocabulary, we extracted all nouns and organized them in attribute values, which were directly imported into *myCBR 3* and from there discussed with the experts. Especially the given taxonomies provided great feedback, because we were discussing both, the terms as well as their relationship. Further, the workbench provided great feedback in explaining CBR because the information the CBR engine uses gets visible. Experts can see local and global similarity measures as well as they can adjust weightings. After 4 sessions with the experts we had a status where the case formats and vocabulary was ready to be deployed in a prototype.

Throughout the project we kept using the workbench when refining case formats as well as similarity measure until the experts themselves started looking into the knowledge models themselves.

On the application's backend, we used the *myCBR 3* SDK to develop a web-based application that searches for similar customer cases after entering all available machine data and observations. Because of the modularity, we were able to deploy updated knowledge models smoothly into the application.

4.2 Knowledge Formalisation for Audio Engineering

A case study on creating a case-based workflow recommendation system for audio engineering support was performed in 2013 [15]. In this study the approach to formalise the special vocabulary used in audio engineering, consisting of vague descriptors for timbres, amounts and directions, was developed. The study introduced CBR as a methodology to amend the problem of formalising the vagueness of terms and the variance of emotions invoked by the same sound in different humans. It was further detailed that the researchers opted for the use of CBR

due to CBR's ability to process fuzzy and incomplete queries and the ability to choose between grades of similarity of retrieved results to emulate the vagueness. The relations between timbres, amounts and effects, were modelled into the local similarity measures of the initial CBR knowledge model as they compose the overall problem description part of what was later used as a case in the resulting CBR engine.

A challenge during this case study was encountered in the form of the task of finding an optimal grade of abstraction for the frequency levels in audio engineering within the CBR knowledge model. This was of importance as in any knowledge formalisation task, one is facing the trade-off between an over engineered, too specific knowledge model and the danger of knowledge loss by employing too much abstraction e.g. choosing the abstraction levels too high. The challenge was met by the researchers by choosing two additional abstraction levels of frequency segments for the timbre descriptors[15].

The next knowledge modelling step consisted of determining the best value ranges for the numerical attributes which were to be integrated into the initial knowledge model. After discussing this approach with the domain experts, the researchers agreed to use two ways to represent *amounts* in the knowledge model. The first way used a percentage approach, ranging from 0 to 100% and the second way used a symbolic approach. The symbolic approach was chosen because the domain experts mentioned that from their experience the use of descriptors for amounts, such as *'a slight bit'* or *'a touch'* were by far more common in audio mixing sessions then a request like *'make it 17% more airy'*. So the researchers integrated, next to the simple and precise numerical approach, a taxonomy of amount descriptors into the initial knowledge model. The taxonomy was ordered based on the amount the symbol described, starting from the root, describing the highest amount down to the leaf symbols describing synonyms of smallest amounts.

The researchers used the *myCBR 3* Workbench to swiftly transfer their initial elicited knowledge model into a structured CBR knowledge model. Figure 2 provides an insight in the modelling of the local similarity measure for timbre descriptors. The first figure shows the taxonomic modelling on the left and a section from the same similarity measure being modelled in a comparative symbolic table on the right.

Within *myCBR 3* the researchers had the choice between a taxonomic and a comparative table approach. Considering the versatile use of taxonomies in structural CBR[5] the researchers initially opted for the use of taxonomies. Yet regarding the complex similarity relationships between the elicited timbre descriptors the researchers also investigated whether a comparative table approach for modelling the similarities of the timbre descriptors. Experiments to establish the performance and accuracy of both approaches yielded no significant difference in the performance of the similarity measures but taxonomies were found to be more easily and intuitively elicited from the audio engineer experts.

After the initial knowledge model was created the researchers performed a number of retrieval experiments using the *myCBR 3* built in retrieval test-

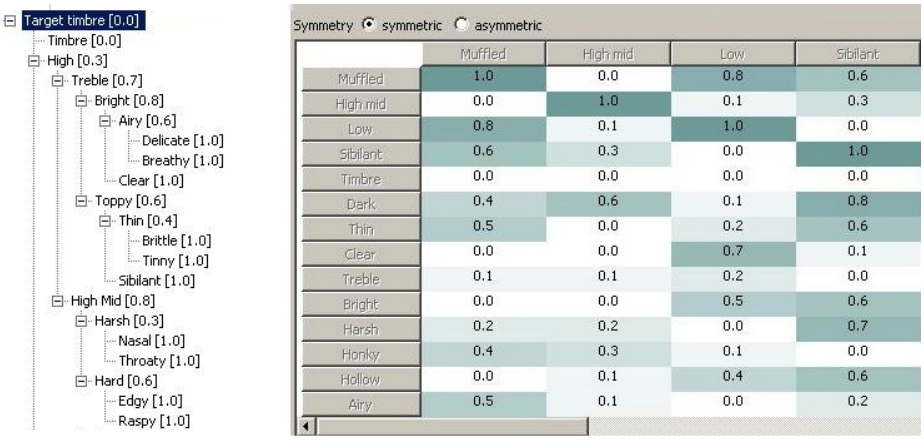


Fig. 2. Timbre descriptor taxonomy and comparative table

ing facilities. The goal of these tests was to refine the initial knowledge model, specifically the similarity measures for the timbre descriptors. Additionally the researchers used the feedback from domain experts to streamline the case structure to the most important attributes. This streamlining was performed within a live system and the researchers were able to directly integrate the streamlined CBR engine into their *Audio Advisor* application thanks to *myCBR 3*'s flexible API.

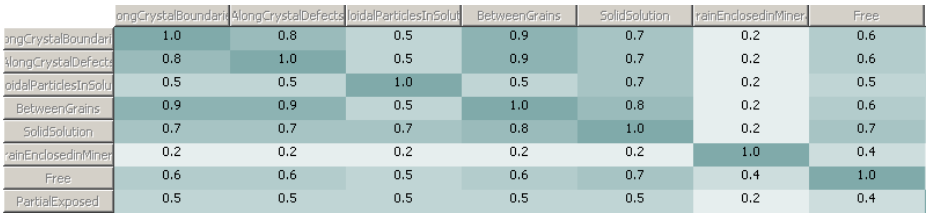
4.3 Knowledge Formalisation for Hydrometalurgy Gold Ore Processing

In this case study a twofold approach to elicitating and formalising knowledge in the domain of hydrometallurgical processing of gold ore was researched. The study demonstrated processes of formalising hydrometallurgy experts knowledge into two different CBR knowledge models. The first knowledge model was than employed in the *Auric Adviser* workflow recommender software [17].

Based on the knowledge gathered from the domain experts the researchers created an initial knowledge model and distributed the knowledge into the 4 knowledge containers of CBR in the following way: The vocabulary consisted of 53 attributes, mainly describing the ore and mineralogical aspects of an ore deposit. With regard to the data types used, the researchers used 16 symbolic, 26 floating point, 6 boolean and 5 integer value attributes. The symbolic attributes described minerals and physical characteristics of minerals and gold particles, such as their distribution in a carrier mineral. Further symbols were elicited to describe the climate and additional contexts a mining operation can be located in, like for example the topography.

The cases were distinctive mainly with regard to the mineralogical context of the mined ore. Thus the researchers created 5 cases describing refractory

arsenopyritic ores, 5 describing free milling gold ores, 2 on silver rich ores, 6 cases on refractory ores containing iron sulphides, 4 on copper rich ores and one each on antimony sulphide rich ores, telluride ore and carbonaceous ore.



	ongCrystalBoundari	AlongCrystalDefects	loidalParticlesInSolut	BetweenGrains	SolidSolution	rainEnclosedinMiner	Free
ongCrystalBoundari	1.0	0.8	0.5	0.9	0.7	0.2	0.6
AlongCrystalDefects	0.8	1.0	0.5	0.9	0.7	0.2	0.6
oidalParticlesInSolu	0.5	0.5	1.0	0.5	0.7	0.2	0.5
BetweenGrains	0.9	0.9	0.5	1.0	0.8	0.2	0.6
SolidSolution	0.7	0.7	0.7	0.8	1.0	0.2	0.7
rainEnclosedinMiner	0.2	0.2	0.2	0.2	0.2	1.0	0.4
Free	0.6	0.6	0.5	0.6	0.7	0.4	1.0
PartialExposed	0.5	0.5	0.5	0.5	0.5	0.2	0.4

Fig. 3. Example of a similarity measure for the gold distribution within an ore

To compute the similarity of a query, composed of prospective data, and a workflow case, the researchers modelled a series of similarity measures for which the researchers had the choice between comparative tables, taxonomies and integer or floating point functions. For their initial knowledge model the researchers mainly relied on comparative tables.

The study's approach included the idea to model as much of the complex knowledge present in the domain of ore refinement into the similarity measures as possible. This was based on the assumption that the similarity based retrieval approach provided by the use of CBR would allow to capture and counter most of the vagueness still associated with the selection of the optimal process in the hydrometallurgical treatment of refractory ores domain. For example, it was possible to model into the similarity measures such facts as that the ore does not need any more treatment if it contains gold grains greater than 15 micro meters in diameter. Such facts are easy to integrate into the similarity measure and thus are operational (having an effect) in the knowledge model. The researchers deemed this capability of the similarity measures to capture and represent such 'odd' behaviours of the knowledge model very important. The study assumes also that these 'odd' facts or bits of knowledge are hard to capture by rules, and thus has ultimately kept another, rule-based approach of modelling the hydrometallurgical domain knowledge, IntelliGold, from succeeding on a broad scale [20].

For the global similarity measure of the cases the researchers used a weighted sum of the attributes local similarities. This allowed for the easy and obvious emphasise of important attributes, such as for example ' Clay Present', as the presence of clay forbids a selection of hydrometallurgical treatments. As the study mainly aiming for case retrieval, the need for adaptation knowledge was minor. Therefore the researchers did not formalised any adaption knowledge. The retrieval results achieved with the first knowledge model was described as satisfying in accuracy and applicability by domain experts.

5 Conclusion and Future Work

In this paper we have presented the approach to knowledge formalisation within *myCBR 3*. *myCBR 3* emphasised the fact that *myCBR 3* is a very versatile tool to create CBR knowledge models with a particular versatile suit of editors for similarity modelling. During the evaluations of the presented projects with the stakeholders, especially the domain experts found that the GUIs offered by *myCBR 3* are intuitive, particularly with regard that they did not have prior knowledge of CBR and the required domain knowledge modeling techniques.

Furthermore, also based on experiences from the case studies, we demonstrated that *myCBR 3* allows for on-going knowledge model improvement, even in a running application. This fact allows also for knowledge maintenance and refinement in live CBR applications and also enables developers to follow the rapid prototyping approach in their projects. As shown in previous a research cooperation with COLIBRI, as well as in a research cooperation on similarity of event sequences, *myCBR 3* is particular versatile for similarity measure based knowledge modelling. Furthermore *myCBR 3* is also easily extendable with regard to its SDK and API to cater for any kind of new similarity measures [14].

For future work we are currently reviewing prototype implementations of additional features for *myCBR 3*. These additional features comprise the ability of automatic extraction of vocabulary items and similarity measures from web community data, the incorporation of drools for the generation of adaption knowledge and the incorporation of case acquisition from databases. Furthermore we are currently finishing the work on the next release of *myCBR 3*, reaching version 3.1. We are also in the process of integrating a mobile version of *myCBR 3*, catering for the needs of android application, such as fast access to assets in a future version of *myCBR 3* [16].

References

1. Aamodt, A.: Knowledge-intensive case-based reasoning in creek. In: Funk, P., Gonzlez-Calero, P.A. (eds.) Proceedings of the ECCBR 2004. LNCS, vol. 3155, pp. 1–15. Springer (2004)
2. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. Artificial Intelligence Communications, 7(1), 39–59 (1994)
3. Bach, K., Althoff, K.D.: Developing case-based reasoning applications using mycbr 3. In: Agudo, B.D., Watson, I. (eds.) Case-Based Reasoning Research and Development, LNCS, vol. 7466, pp. 17–31. Springer Berlin Heidelberg (2012)
4. Bach, K., Althoff, K.D., Newo, R., Stahl, A.: A case-based reasoning approach for providing machine diagnosis from service reports. In: Ram, A., Wiratunga, N. (eds.) Case-Based Reasoning Research and Development (Procs. of the 19th International Conference on Case-Based Reasoning). vol. 6880, pp. 363–377. Springer Verlag, Berlin Heidelberg (2011)
5. Bergmann, R.: Experience Management: Foundations, Development Methodology, and Internet-Based Applications, Lecture Notes in Computer Science, vol. 2432. Springer (2002)

6. Gundersen, O.E., Sørmo, F., Aamodt, A., Skalle, P.: A real-time decision support system for high cost oil-well drilling operations. AAAI Publications, Twenty-Fourth IAAI Conference (2012)
7. Hanft, A., Schäfer, O., Althoff, K.D.: Integration of drools into an osgi-based bpm-platform for cbr. In: Agudo, B.D., Cordier, A. (eds.) ICCBR-2011 Workshop Proceedings: Process-Oriented CBR (2011)
8. Kolodner, J.: Case-based reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (1993)
9. López, B., Pous, C., Gay, P., Pla, A., Sanz, J., Brunet, J.: exit*cbr: A framework for case-based medical diagnosis development and experimentation. Artif. Intell. Med. 51(2), 81–91 (Feb 2011)
10. Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A.: A distributed cbr framework through semantic web services. In: Bramer, M., Coenen, F., Allen, T. (eds.) Research and Development in Intelligent Systems XXII (Proc. of AI 2005). pp. 88–101. Springer (December 2005)
11. Recio-García, J.A., Díaz-Agudo, B., González-Calero, P.A.: Template based design in colibri studio. In: Proceedings of the Process-oriented Case-Based Reasning Workshop at ICCBR'11. pp. 101–110 (2011)
12. Richter, M.M.: Introduction. In: Lenz, M., Bartsch-Spörl, B., Burkhard, H.D., Wess, S. (eds.) Case-Based Reasoning Technology – From Foundations to Applications. LNAI 1400, Springer-Verlag, Berlin (1998)
13. Riesbeck, C.K., Schank, R.C.: Inside case-based reasoning. Lawrence Erlbaum Associates, Pubs., Hillsdale, N.J. (1989)
14. Roth-Berghofer, T., Sauer, C., Garcia, J.A.R., Bach, K., Althoff, K.D., Agudo, B.D.: Building case-based reasoning applications with mycbr and colibri studio. In: Case-Based Reasoning Research and Development. Springer (2012)
15. Sauer, C., Roth-Berghofer, T., Auricchio, N., Proctor, S.: Recommending audio mixing workflows. In: Case-Based Reasoning Research and Development, pp. 299–313. Springer (2013)
16. Sauer, C.S., Hundt, A., Roth-Berghofer, T.: Explanation-aware design of mobile mycbr-based applications. In: Case-Based Reasoning Research and Development, pp. 399–413. Springer (2012)
17. Sauer, C.S., Rintala, L., Roth-Berghofer, T.: Knowledge formalisation for hydrometallurgical gold ore processing. In: Research and Development in Intelligent Systems XXX, pp. 291–304. Springer (2013)
18. Schulz, S.: Cbr-works - a state-of-the-art shell for case-based application building. In: Proceedings of the 7th German Workshop on Case-Based Reasoning, GWCBR'99, Wrzburg. pp. 3–5. Springer-Verlag (1999)
19. Stahl, A., Roth-Berghofer, T.R.: Rapid prototyping of CBR applications with the open source tool myCBR. In: Proceedings of the 9th European conference on Advances in Case-Based Reasoning. pp. 615–629. Springer-Verlag, Heidelberg (2008)
20. Torres, V.M., Chaves, A.P., Meech, J.A.: Intelligold-an expert system for gold plant process design. Cybernetics & Systems 31(5), 591–610 (2000)

SBVRwiki (Tool Presentation)*

Krzysztof Kluza, Krzysztof Kutt and Marta Woźniak

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

Abstract. SBVR is a mature standard for capturing expressive business rules along with their semantics, and is especially useful in the communication with business people. SBVRwiki is a novel tool that allows for an effective use of the SBVR notation. This online collaborative solution allows for distributed and incremental rule authoring for business analytics and users. It supports creation of vocabularies, terms and rules in a transparent, user-friendly fashion. The tool provides visualization and evaluation mechanisms for created rules, and besides basic syntax highlighting and checking, it allows for logical analysis. As it is integrated with the Loki knowledge engineering platform, it allows for on-the-fly conversion of the SBVR rule base and vocabularies to Prolog. The Dokuwiki back-end provides storage and unlimited version control, as well as user authentication.

1 Introduction and Motivation

SBVR (Semantics of Business Vocabulary and Business Rules) [7] is a standard for capturing expressive business rules, commonly perceived as a useful tool in the communication between business analytics and business people. The set of vocabularies and rules described with the use of SBVR can be an important part of requirements specification from the software engineering methodologies.

An effective use of the SBVR notation is non trivial, because it requires certain knowledge engineering skills. Thus, there is a need for software supporting business analytics in the rule acquisition process. Such software should allow for syntax checking, automatic hinting as well as preliminary evaluation of the resulting set of rules.

There are several SBVR-supporting tools like editors that support text-based creation of dictionaries and business rules providing syntax highlighting and suggestions; modelers that allow for generating models based on SBVR compliant documents; or tools that allow a user to import various models and transform them into the SBVR syntax. Considering their limitations of the existing tools supporting SBVR authoring, our motivation is to deliver a lightweight tool allowing for easy creation of the SBVR knowledge bases even for inexperienced users. We opt for a web-based solution that allows business users and analytics to collaborate using a familiar browser-based interface.

* The paper is supported from the *Prosecco* project funded by NCBR.

SBVRwiki uses the Dokuwiki[1] back-end for storage, unlimited version control and user authentication. The tool supports identification and creation of vocabularies, terms and rules in a transparent, user friendly fashion. It also provides visualization and evaluation mechanisms for created rules.

Our tool is integrated with the Loki knowledge engineering platform [5,4], which is based on DokuWiki as well. This allows for on-the-fly conversion of the SBVR rule base and vocabularies to Prolog. Use of the Prolog-based representation also opens up possibilities of formalized analysis of SBVR rules.

The rest of the paper is structured as follows. Section 2 discusses the functional requirements for the wiki-based collaborative platform for knowledge engineering, especially for requirements analysis. Then, in Section 3 the SBVRwiki system is presented. Future work is summarized in the final Section 4.

2 Functional Requirements

Wikis are broadly used in various areas including distributed requirements analysis [2,3]. They are chosen as they are easy to use, provide an efficient way for collaboration within a large group of people and their maintenance is almost costless. The drawbacks of using the Wiki systems in design process are lack of automatic analysis and conflicts detection as well as unstructured text in which documentation is written. Moreover, users can write requirements in the forms that are suitable for them. Thus, others can misinterpret the idea and this can result in useless time-consuming debates. The tool presented in this paper addresses both indicated issues. It combines simplicity of use with the SBVR standard that enforces structured specification.

Main functional requirements for an SBVR wiki-based system are as follows: 1) creation of a new SBVR project composed of vocabularies, facts, and rules using a set of predefined templates, 2) authoring of a project using structured vocabularies, with identified categories, 3) SBVR syntax verification and highlighting in text documents, as well as syntax hinting, 4) visualization of vocabularies and rules as UML class diagrams to boost the transparency of the knowledge base, 5) file export in the form of SBVR XMI, 6) integration with the existing PlWiki and BPWiki [6] platforms, 7) full support for the SBVR syntax, including at least binary facts, 8) constant assistance during the editing of the SBVR statements, including elimination of common errors, the use of undefined concepts, duplicated entries, etc. Using these requirements, a prototype implementation called SBVRwiki was developed [8].

3 System Presentation

The development of the SBVR wiki was based on several implementation requirements, such as: the system should be an extension (plugin) to the DokuWiki platform by extending the capabilities of the native Dokuwiki editor; it should operate in parallel with the PlWiki and BPWiki extensions, and should not interfere with the operation of other add-ons installed on the platform.

[1] A lightweight and open-source wiki engine, see: `www.dokuwiki.org`.

Dokuwiki offers several classes of plugins allowing for fine-grained processing of the wiki text. SBVRwiki implements two main plugin components:

- SBVRwiki Action Plugin – responsible for the file export in the XMI (XML) format. It also handles the user interface events and extends the built-in Dokuwiki editor with shortcuts for common SBVR constructs.
- SBVRwiki Syntax Plugin – used to enter SBVR expressions as wiki text using a special `<sbvr>` wiki markup. Using it a user can enter legal SBVR expressions and the plugin offers rich syntax highlighting. Moreover, vocabularies can be visualized with the dynamic translation to UML class diagrams that are rendered by the wiki using the PlantUML tool[2], see Fig. 1.

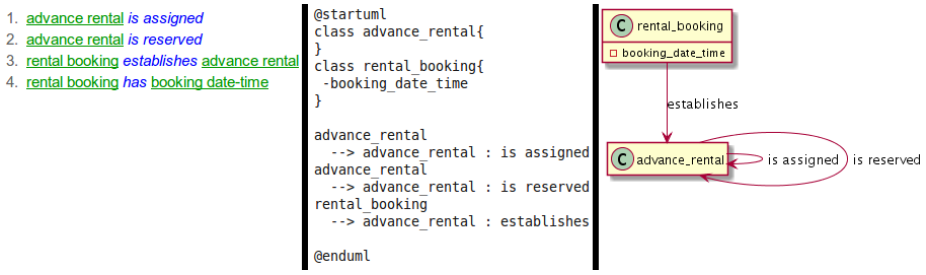


Fig. 1. Class diagram generation with PlantUML (from left: fragment of the fact dictionary; a text file to generate the class diagram; and UML class diagram generated by the PlantUML tool)

There are several advantages of using a Wiki system as the implementation platform. As the SBVR expressions can be stored in separate wiki pages, the content can be simultaneously edited by a number of users. The Loki engine can select only the relevant parts of this knowledge on the fly, so the pages can contain not only the SBVR content, but also additional pieces of information, such as comments, figures, media attachments, and hypertext links to other resources in the wiki or on the Web.

For this tool presentation, we uses a benchmark case of SBVR knowledge base, the classic EU Rent case[3], provided as a part of the SBVR specification [7].

EU-Rent is a fictional international car rental company. Renters may be individuals or accredited members of corporate customers (a company or similar organization). In each country, EU-Rent offers broadly the same kinds of cars, ranging from "economy" to "premium" although the mix of car models varies between countries. Rental prices also vary from country to country. However, different models of offered cars are organized into groups, and all cars in a group are charged at the same rates within a country.

[2] See: `plantuml.sf.net`.

[3] You can browse the EU-Rent case using wiki on: `http://home.agh.edu.pl/~kkutt/sbvrwiki-demo/`. Credentials: `demo/demo`.

A car rental is a contract between EU-Rent and a renter, who is responsible for payment for the rental and any other costs associated with the rental (except those covered by insurance). A rental booking specifies: the car group required; the start and end dates/times of the rental; the EU-Rent branch from which the rental is to start.

During creating a new SBVR project, several steps have to be taken. However, this does not require much knowledge of the application because a user is supported by a set of built-in guiding wizards. Using SBVRwiki, a user should define concepts first, then writes down facts, and finally rules can be authored. Concepts, facts and rules are stored in the separate Wiki namespaces. The Lexer module of the plugin detects the previously defined tokens what allows for proper syntax highlighting as well as for detecting the use of undefined concepts.

In the case of the EU-Rent example, a user can start from creating a dictionary. It is a set of terms grouped in categories. If a user wants to describe concepts such as *additional driver* or *loyalty club*, it is a possibility to write down the information about these terms, such as a source, a type or a definition. A user can also group them in category *Customers*. Finally, dictionary is saved and clearly formatted wiki page can be displayed (see Fig. 2).

The next step consists in creating the base of facts. A user can specify what dependencies exist between previously defined concepts. E.g. it can be specified that *loyalty club* includes *club member*, *renter* is *club member*, and *renter* is *additional driver*. When all such facts are described and saved, simple UML graphs are generated to visualize created database (see Fig. 3). In the last step, a user specifies rules providing constraints for modeled system. When saved, they are also clearly formatted and visualized in simple graphs form (see Fig. 4).

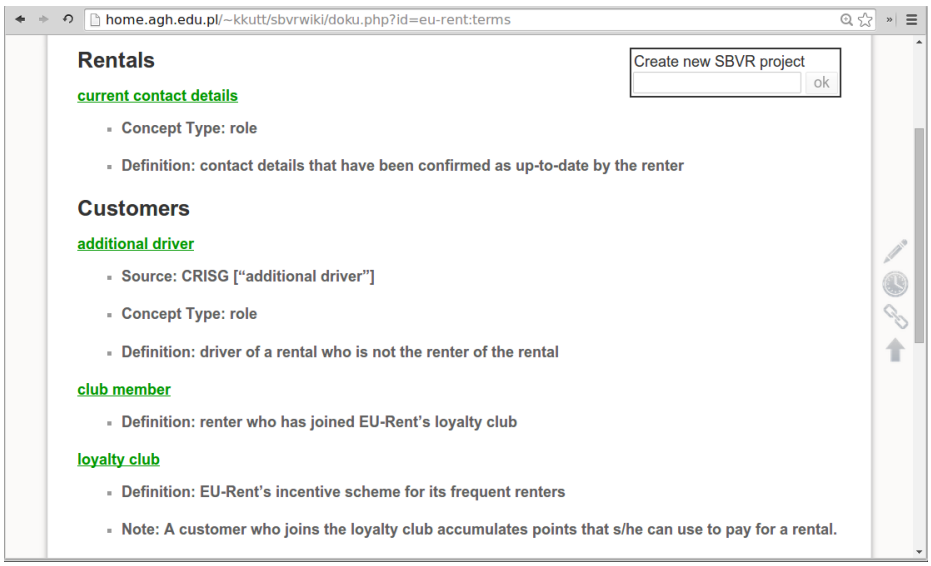


Fig. 2. EU Rent Terms Dictionary

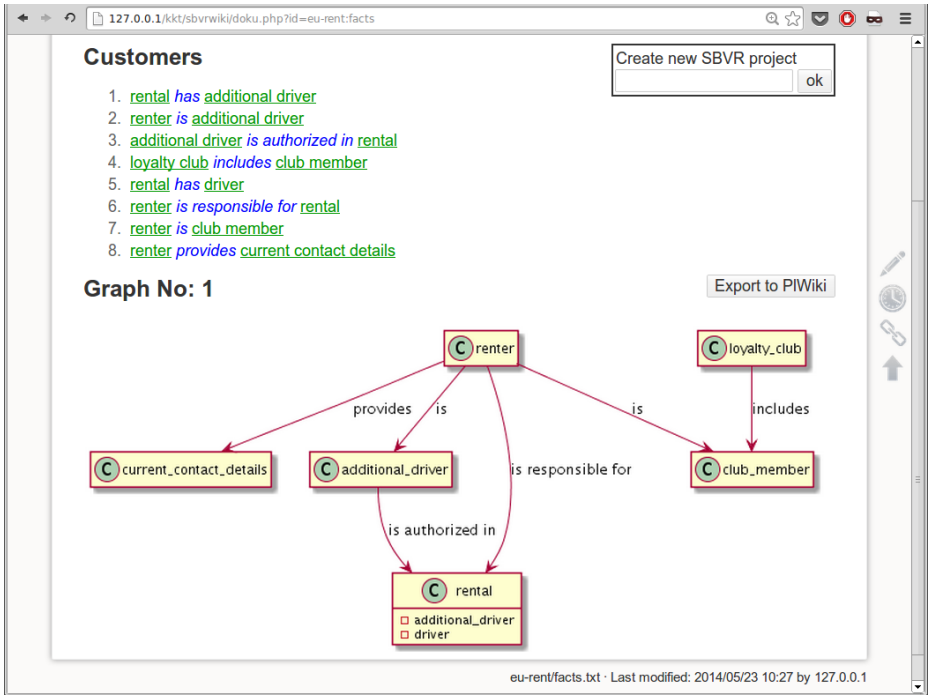


Fig. 3. EU Rent Facts Visualization

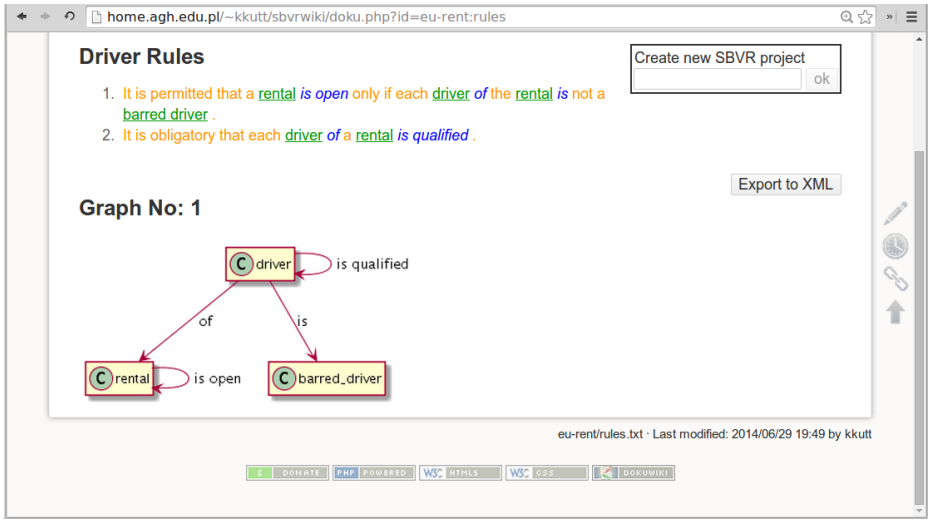


Fig. 4. EU Rent Rules Visualization

4 Future Work

Although the current version of SBVRwiki does not support all aspects of the SBVR standard, it implements enough of its elements to allow users to create complex models of business rules in Structured English. Its limitations are the lack of support for multi-argument facts, polymorphism or additional attributes for the expressions, dictionaries, facts and rules.

As the tool is implemented as a plugin for the DokuWiki system, it can be integrated with the BPWiki plugin [6]. This would allow for specification of systems including both business processes and rules. Moreover, the use of the Prolog-based representation in Loki opens up possibilities of formalized analysis of the SBVR rules. In future, the tool can be also extended to use the rule engine integrated with DokuWiki [1] for reasoning capabilities.

SBVRwiki has been recently used in the Prosecco[4] research project as a tool for authoring the SBVR rules. One of the objectives of the project is the development of a system supporting management of SMEs (Small and Medium Enterprises) using business process and rules with well defined semantics. Thus, the tool was used to model vocabularies and rules identified in the SMEs participating in the project.

References

1. Adrian, W.T., Bobek, S., Nalepa, G.J., Kaczor, K., Kluza, K.: How to reason by HeaRT in a semantic knowledge-based wiki. In: Proceedings of the 23rd IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2011. pp. 438–441. Boca Raton, Florida, USA (November 2011)
2. Decker, B., Ras, E., Rech, J., Jaubert, P., Rieth, M.: Wiki-based stakeholder participation in requirements engineering. Software, IEEE 24(2), 28–35 (March 2007)
3. Liang, P., Avgeriou, P., Clerc, V.: Requirements reasoning for distributed requirements analysis using semantic wiki. In: 2009 Fourth IEEE International Conference on Global Software Engineering (ICGSE 2009), 13-16 July 2009, Limerick, Ireland. pp. 388–393. IEEE (July 2009)
4. Nalepa, G.J.: Collective knowledge engineering with semantic wikis. Journal of Universal Computer Science 16(7), 1006–1023 (2010), http://www.jucs.org/jucs_16_7/collective_knowledge_engineering_with
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), http://www.springerlink.com/content/y91w134g03344376/
6. Nalepa, G.J., Kluza, K., Ciaputa, U.: Proposal of automation of the collaborative modeling and evaluation of business processes using a semantic wiki. In: Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation ETFA 2012, Kraków, Poland, 28 September 2012 (2012)
7. OMG: Semantics of Business Vocabulary and Business Rules (SBVR). Tech. Rep. dtc/06-03-02, Object Management Group (2006)
8. Woźniak, M.: Analysis of applications of rule-based tools in requirements engineering. Master's thesis, AGH UST (2013), Supervisor: G. J. Nalepa

[4] See: http://prosecco.agh.edu.pl.