9th Workshop on

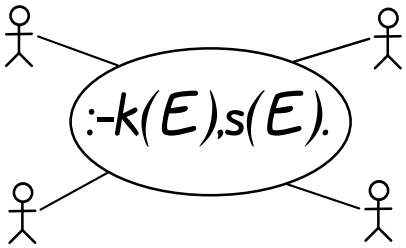
Knowledge Engineering and Software Engineering (KESE)

at the

36th German Conference on Artificial Intelligence (KI2013)

September 17, 2013, Koblenz, Germany

Grzegorz J. Nalepa and Joachim Baumeister (Editors)



Technical Report No. 487, Würzburg University, Würzburg, Germany, 2013

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). Nine years later the KESE9 workshops return to Koblenz, where it is collocated with the 36th Annual Conference on Artificial Intelligence in Koblenz (September 16-20, 2013). This year we solicited contributions having the following topics:

- Knowledge and software engineering for the Semantic Web
- Ontologies in practical knowledge and software engineering
- Business Rules design, management and analysis
- Business Processes modeling in KE and SE
- Practical knowledge representation and discovery techniques in software engineering
- Agent-oriented 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
- 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 five regular, and two short papers were accepted to the workshop. Moreover, one tool presentation was also included.

In their paper "Integrating Semantic Knowledge in Data Stream Processing" the authors Beckstein et al. describe different approaches on integrating stream data and semantic domain knowledge. In particular, as accessing methods the continuous query language CQL is compared with the SPARQL extension C-SPARQL.

Kramer et al. describe new possibilities for explanation generation. Their paper "Towards Explanation Generation using Feature Models in Software Product Lines" investigate how the approach can be applied in dynamic software product lines (DSPL).

In the paper "A Prolog Framework for Integrating Business Rules into Java Applications" the authors Ostermayer and Seipel show an approach to connect the data structures of the logic-based language Prolog with the wide-spread programing language Java.

Baumeister et al. report in their paper "Continuous Knowledge Representations in Episodic and Collaborative Decision Making" on a new type of decision support systems and demonstrate its application in an industrial case study for managing the knowledge about chemical substances.

Pascalau introduces guidelines for designing and engineering advanced software systems to be used by end-users. The paper "Identifying Guidelines for Designing and Engineering Human-Centered Context-Aware Systems" proposes a declarative level to hide the technical level of systems engineering from the end-users.

Kluza et al. tackle business process modeling and give an overview of recommendation possibilities. Their paper "Overview of Recommendation Techniques in Business Process Modeling" describes a categorization of recommendation approaches.

Newo and Althoff report in their paper "Knowledge Acquisition for Life Counseling" on a concrete project that uses case-based techniques and information extraction methods in the life counseling domain.

Kaczor et al. give a tool presentation and show in "HaDEsclipse - Integrated Environment for Rules" an environment for engineering rule-based systems. The tool is based in the well-established software tool Eclipse.

The organizers would like to thank all who contributed to the success of the workshop. We thank all authors for submitting papers to the workshop, and we thank the members of the program committee as well as the external reviewers for reviewing and collaboratively discussing the submissions. For the submission and reviewing process we used the EasyChair system, for which the organizers would like to thank Andrei Voronkov, the developer of the system. Last but not least, we would like to thank the organizers of the KI 2013 conference for hosting the KESE9 workshop.

Grzegorz J. Nalepa
Joachim Baumeister

Workshop Organization

The 9th Workshop on Knowledge Engineering and Software Engineering
(KESE9)
was held as a one-day event at the
36th German Conference on Artificial Intelligence
(KI2013)
on September 17 2013, in Koblenz, Germany

Workshop Chairs and Organizers

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

Programme Committee

Isabel María del Águila, University of Almeria, Spain
Klaus-Dieter Althoff, University Hildesheim, Germany
Kerstin Bach, Verdande Technology AS, Norway
Joachim Baumeister, denkbares GmbH/University Wuerzburg, 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
Thomas-Roth Berghofer, University of West London, UK
José del Sagrado, University of Almeria, Spain
Dietmar Seipel, University Würzburg, Germany

Table of Contents

Integrating Semantic Knowledge in Data Stream Processing 1
Simon Beckstein, Ralf Bruns, Juergen Dunkel and Leonard Renners

Towards Explanation Generation using Feature Models in Software Product Lines 13
Dean Kramer, Christian Sauer and Thomas Roth-Berghofer

Towards Continuous Knowledge Representations in Episodic and Collaborative Decision Making 24
Joachim Baumeister, Albrecht Striffler, Marc Brandt and Michael Neumann

Identifying Guidelines for Designing and Engineering Human-Centered Context-Aware Systems 36
Emilian Pascalau

Overview of Recommendation Techniques in Business Process Modeling .. 46
Krzysztof Kluza, Mateusz Baran, Szymon Bobek and Grzegorz J. Nalepa

A Prolog Framework for Integrating Business Rules into Java Applications 58
Ludwig Ostermayer and Dietmar Seipel

Knowledge Acquisition for Life Counseling 70
Régis Newo and Klaus-Dieter Althoff

HaDEsclipse - Integrated Environment for Rules (Tool Presentation) 77
Krzysztof Kaczor, Grzegorz J. Nalepa and Krzysztof Kutt

Integrating Semantic Knowledge in Data Stream Processing

Simon Beckstein, Ralf Bruns, Jürgen Dunkel, Leonard Renners

University of Applied Sciences and Arts Hannover, Germany
Email: forname.surname@hs-hannover.de

Abstract. Complex Event Processing (CEP) has been established as a well-suited software technology for processing high-frequent data streams. However, intelligent stream based systems must integrate stream data with semantical background knowledge. In this work, we investigate different approaches on integrating stream data and semantic domain knowledge. In particular, we discuss from a software engineering perspective two different architectures: an approach adding an ontology access mechanism to a common Continuous Query Language (CQL) is compared with C-SPARQL, a streaming extension of the RDF query language SPARQL.

1 Introduction

Nowadays, much information is provided in form of data streams: sensors, software components and other sources are continuously producing fine-grained data that can be considered as streams of data. Examples of application fields exploiting data streams are traffic management, smart buildings, health monitoring, or financial trading. Intelligent decision support systems analyze stream data in real-time to diagnose the actual state of a system allowing adequate reactions on critical situations.

In recent years, Complex Event Processing (CEP) [10] has been established as a well-suited software technology for dealing with high frequent data streams. In CEP each data item in a stream is considered as an *event*. CEP uses Continuous Query Languages (CQL) to describe patterns in event streams, which define meaningful situations in the application domain.

However, for *understanding* the business meaning of stream data, the data items must be enriched with semantical background knowledge. For instance in traffic management, velocity measures must be related to specific knowledge about the road network (e.g. road topology and speed limits). In contrast to data streams, this background or domain knowledge is usually rather static and stable, i.e. without frequent changes.

Ontologies defined by Description Logic (DL) [8] provide a well-known formalism for knowledge representation, that can also be used for describing background knowledge. DL distinguishes two different aspects: (1) the TBox contains terminological or domain concepts, and (2) the ABox defines assertional

knowledge or individuals of the concepts that are defined in the TBox. Common languages for describing semantic knowledge are the Resource Description Framework (RDF) for the TBox and the Ontology Language OWL[1] for the ABox. SPARQL [11] provides a standard query language for retrieving knowledge represented in form of RDF data.

Note that SPARQL was originally developed to process static data and is therefore not suitable for the processing of data streams. Otherwise, conventional CEP languages provide no inherent concepts for accessing ontological knowledge.

In this work, we will investigate different approaches on how to integrate data stream processing and background knowledge bases. In particular, we will discuss two different aspects from a software engineering perspective:

- How can CQL languages provided by standard CEP systems make use of ontology models?
- How useful are recently proposed streaming extensions of SPARQL such as C-SPARQL?

The remainder of the paper is organized as follows. The next section discusses related work and other research approaches. Subsequently, section 3 introduces briefly CEP. Then, section 4 discusses the different types of information that can be exploited in stream based systems. The following sections 5 and 6 describe and compare two different approaches of integrating background knowledge into stream processing: The first approach adds an ontology access mechanism to a common CQL-based architecture. The second one uses C-SPARQL, a streaming extension of SPARQL. The final section 7 provides some concluding remarks and proposes an outlook for further lines on research.

2 Related Work

In practice, nearly all stream processing systems are using a proprietary Continuous Query Language (CQL). At present, many mature implementations of event processing engines already exist. Some well-known representatives are ESPER[2], JBoss Drools Fusion[3] or Oracle CEP[4]. As already discussed, none of these engines neither target nor support a built-in way to integrate semantic background knowledge.

Another class of approaches target the integration of RDF ontologies with stream processing. Different SPARQL enhancements have been developed in order to query continuous RDF streams. Basically, they all extend SPARQL by sliding windows for RDF stream processing:

- *C-SPARQL* provides an execution framework using existing data management systems and triple stores. Rules distinguish a dynamic and a static part, which are evaluated by a CQL and a SPARQL engine, respectively [5, 4].

[1] `http://www.w3.org/TR/2012/REC-owl2-primer-20121211/`
[2] `http://esper.codehaus.org/`
[3] `http://jboss.org/drools/drools-fusion.html`
[4] `http://oracle.com/technetwork/middleware/complex-event-processing`

– *Streaming-SPARQL* simply extends a SPARQL engine to support window operators [6].
– *EP-SPARQL* is used with ETALIS, a Prolog based rule engine. The knowledge (in form of RDF) is transformed into logic facts and the rules are translated into Prolog rules [1, 2].
– *CQELS* introduces a so called white-box approach, providing native processing of static data and streams by using window operators and a triple-based data model [9].

Beside SPARQL extensions, various proprietary CEP languages have been proposed for integrating stream processing and ontological knowledge: For instance, Teymourian et. al. present ideas on integrating background knowledge for their existing rule language Prova[5] (with a corresponding event processing engine) [13, 14].

In summary, many proposals for SPARQL dialects or even new languages have been published, but so far not many results of practical experiments have been proposed.

This paper examines two different approaches for integrating RDF and stream data from a software engineering perspective. First, we extend the well-known CQL of ESPER with mechanisms for accessing RDF ontologies. Then, this approach is compared with C-SPARQL, one of the SPARQL extensions that integrates SPARQL queries and stream processing.

3 Complex Event Processing - Introduction

Complex Event Processing (CEP) is a software architectural approach for processing continuous streams of high volumes of events in real-time [10]. Everything that happens can be considered as an *event*. A corresponding *event object* carries general metadata (event ID, timestamp) and event-specific information, e.g. a sensor ID and some measured data. Note that single events have no special meaning, but must be correlated with other events to derive some understanding of what is happening in a system. CEP analyses continuous streams of incoming events in order to identify the presence of complex sequences of events, so called *event patterns*.

A *pattern match* signifies a meaningful state of the environment and causes either creating a new *complex event* or triggering an appropriate action.

Fundamental concepts of CEP are an *event processing language* (EPL), to express *event processing rules* consisting of *event patterns* and *actions*, as well as an *event processing engine* that continuously analyses event streams and executes the matching rules. Complex event processing and event-driven systems generally have the following basic characteristics:

[5] `https://prova.ws/`

- *Continuous in-memory processing*: CEP is designed to handle a consecutive input stream of events and in-memory processing enables real-time operations.
- *Correlating Data*: It enables the combination of different event types from heterogenous sources. Event processing rules transform fine-grained simple events into complex (business) events that represent a significant meaning for the application domain.
- *Temporal Operators*: Within event stream processing, timer functionalities as well as sliding time windows can be used to define event patterns representing temporal relationships.

4 Knowledge Base

In most application domains, different kinds of knowledge and information can be distinguished. In the following, the different types of knowledge are introduced by means of a smart building scenario:[6] An energy management system that uses simple sensors and exploits the background knowledge about the building, environment and sensor placement.

The main concepts used in the knowledge base are *rooms* and *equipment*, such as *doors* and *windows* of the rooms. Rooms and equipment can be attached with certain *sensors* measuring the *temperature*, *motion* in a room or the *state* of a door or a window, respectively. By making use of this background information, the raw sensor data can be enriched and interpreted in a meaningful manner. For instance, room occupancies due to rescheduled *lectures* or ad-hoc meetings can be identified for achieving a situation-aware energy management. In this sample scenario, we can identify three types of knowledge classified according to their different change frequencies:

1. ***Static knowledge:*** We define static knowledge as the knowledge about the static characteristics of a domain, that almost never or very infrequently changes. A typical example in our scenario is the structure of a building and the sensor installation.
   Static knowledge can be modeled by common knowledge representation formalisms such as ontologies. Because this information does usually not change, appropriate reasoners can derive implicit knowledge before the start of the stream processing. OWL can serve as a suitable knowledge representation language that is supported by various reasoners, for example KAON2[7] or FaCT++[8].
2. ***Semi-dynamic knowledge:*** We consider semi-dynamic knowledge as the knowledge about the expected dynamic behavior of a system. It can be represented by static knowledge models, e.g. ontologies, as well. In our scenario, a class schedule predicts the dynamic behavior of the building: though the

[6] More details about the smart building scenario can be found in [12].
[7] `http://kaon2.semanticweb.org/`
[8] http://owl.man.ac.uk/factplusplus/

class schedule can be defined by static data (e.g facts in an ontology), it causes dynamic events, e.g. each monday at 8:00 a 'lecture start' event. Of course, real-time data produced by sensor could outperform the predicted behavior, e.g. if a reserved class room is not used.

3. ***High-dynamic knowledge:*** The third type of knowledge is caused by unforseeable incidents in the real world. It expresses the current state of the real world and cannot be represented by a static ontology. Instead the current state has to be derived from continuous stream of incoming data. This type of knowledge can be described by an event model specifying the types of valid events.[9] Examples in our scenario are sensor events representing observations in the physical world, e.g. motion, temperature, or the state of a window or door, respectively.

The three knowledge types introduced above provide only a basic classification scheme. As already discussed in the introduction (section 1), various types of information must be integrated and correlated in order to derive complex events that provide insight to the current state of a system.

5 Using Semantic Knowledge in Event Processing

In this section, we will investigate how the different types of knowledge introduced above can be integrated in stream processing – in particular, how ontological knowledge can be exploited in stream processing.

We start our discussion with a small part of an ontology for our energy management scenario (see Figure 1). This sample ontology is used in the following paragraphs for discussing the different knowledge integration approaches. The model defines the three concepts 'room', 'sensor' and 'equipment' and their relationships. It shows that each room can contain sensors and equipment. Furthermore, it specifies that a certain sensor is either directly located in a certain room or attached to an equipment located in a room.

Note that the location of a sensor can be inferred from the location of the equipment it is attached to. The dashed line describes this implicit property, which can be expressed as role composition in Description Logic:

$isAttacedTo \circ IsEquippedIn \sqsubseteq hasLocation$. A DL role composition can be considered as a rule: If a sensor is attached to an equipment and the equipment is equipped in a certain room, then the sensor is assumed to be located in the same room.

Listing 1.1 defines two individuals (*Window362* and an attached contact sensor *C362W*) using the RDF turtle notation[10] . Using the above presented DL rule, it can be inferred that the contact sensor is located in room 362 and the triple (`:C362W :hasLocation :Room362`) can be added to the knowledge base.

In the same way, further role and concept characteristics of the ontology can be used for reasoning purposes.

[9] Note that such an event model can also be formally defined by an OWL ontology.
[10] `http://www.w3.org/TR/turtle/`

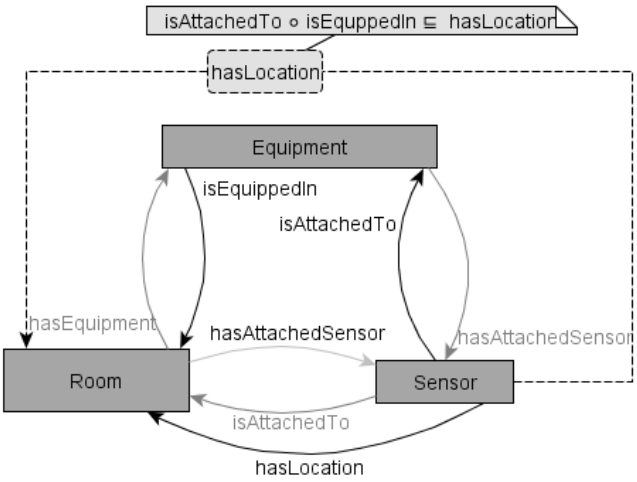


Fig. 1. OWL ontology relationship

```
:Window362
    rdf:type :Window ,
    :isEquippedIn :Room362 .

:C362W
    rdf:type :ContactSensor ,
    :isAttachedTo :Window362 .
```

Listing 1.1. Some sample entries of the domain knowledge

5.1 ESPER

As a first approach of integrating stream data and background knowledge we have chosen the established event processing engine ESPER. Since it is a regular CQL based engine it does not natively support the access of additional knowledge bases. Figure 2 depicts the conceptional architecture of the approach. Different event sources send streams of events via message channels to the ESPER CEP engine. The event sources provide all events in a format that is processable by ESPER, for instance simple Java objects (POJOS). The cycle within the engine should denote that the events are processed in several stages. Each stage transforms relatively simple incoming events into more complex and meaningful events.

Knowledge Access: As already mentioned, ESPER does not inherently support a specific access to a knowledge base such as an OWL ontology, but it provides a very general extension mechanism that allows invoking static Java methods within an ESPER rule. Such methods can be used for querying a Java domain model, a database or any other data source. To make our OWL domain

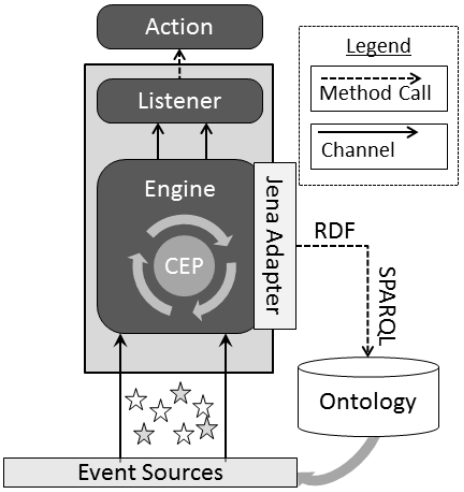


Fig. 2. Architecture using ESPER as CEP component

model accessible from ESPER rules, we implemented an adapter class that uses the Jena Framework[11] to query the ontology via SPARQL.

Events: Because ESPER can only process Java objects, the adapter has to map RDF triples to Java objects. For instance, the mapping transforms an RDF-URI identifying a sensor to an ID in the Java object. Each Java class corresponds with a certain concept of the ontology TBox.

Queries: ESPER provides its own event processing language that is called *ESPER Event Query Language (EQL)*. EQL extends SQL with temporal operators and sliding windows. A simple example is given in Listing 1.2 that shows how motion in a certain room is detected by an ESPER query.

```
SELECT room
FROM pattern[every mse=MotionSensorEvent],
     method:Adapter.getObject(mse.sensorID)
     AS room
```

Listing 1.2. A sample ESPER query

Actions triggered by a pattern match are implemented in a listener class that must be registered for an ESPER rule. A listener can call any event handling Java method or create a new complex event. The example rule looks rather simple, because the access to the knowledge base is hidden behind the method call (here: `Adapter.getObject(mse.sensorID)`). In our case, the adapter executes a SPARQL query using the Jena framework as shown in Listing 1.3.

[11] `http://jena.apache.org`

```
PREFIX    : <http://eda.inform.fh-hannover.de/sesame.owl>
PREFIX  rdf: <http://www.w3.org/1999/02↙
    ↳/22-rdf-syntax-ns#>
SELECT ?room ?object
WHERE { :"+sensorID+" :isAttachedTo ?object ;
                      :hasLocation  ?room .
}
```

Listing 1.3. SPARQL query in the Jena-Adapter method *Adapter.getObject(sensorID)*

5.2 C-SPARQL

As an alternative approach, we investigate a software architecture using C-SPARQL[12], a streaming extension of SPARQL. Figure 3 illustrates the main building blocks of the architecture. The main difference to the previous approach is, that all event sources produce a continuous stream of RDF data. This means that the entire knowledge base of the system uses RDF as uniform description formalism.

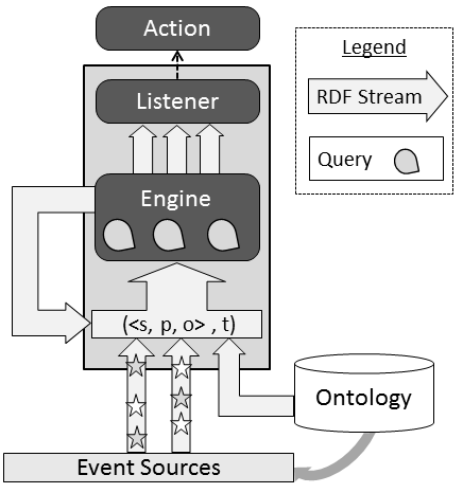


Fig. 3. Architecture using C-SPARQL as CEP component

Knowledge access: In this approach, C-SPARQL queries are used for accessing the homogeneous RDF knowledge base. A single C-SPARQL query can combine incoming RDF streams with static background knowledge (also represented in RDF).

[12] We used the 'ReadyToGoPack', an experimental implementation of the concept in [4, 5], available on `http://streamreasoning.org`

Events: The events themselves arrive as continuous streams of RDF triples. To allow stream processing with RDF triples, they must be extended with a timestamp. Thus, each event can be described by a quadruple of the following form:

$$(\langle subj_i, pred_i, obj_i \rangle, t_i)$$

The *subject* is a unique event identifier, the *predicate* and *object* describe event properties. The *timestamp* is added by the engine and describes the point of time the event arrived. Listing 1.4 shows a set of RDF triples describing a simplified temperature sensor event.

```
:event123 rdf:type                                    :↙
    ↳TemperatureSensorEvent
:event123    :hasSensorId    :3432
:event123    :hasValue       24.7^^xsd:double
```

Listing 1.4. A sample temperature event

Queries: C-SPARQL queries are syntactically similar to SPARQL. Listing 1.5 shows a C-SPARQL query expressing the same pattern as the ESPER query in Listing 1.2. In contrast to SPARQL, it provides language extensions for temporal language constructs like (sliding) time and batch windows as shown at the end of the `FROM STREAM`-clause. The `FROM`-clause selects the various data streams

```
SELECT ?room
FROM STREAM <http://eda.inform.fh-hannover.de/↙
    ↳MotionSensorEvent.trdf>↙
    ↳[RANGE 10s STEP 1s]
FROM <http://eda.inform.fh-hannover.de↙
    ↳/sesame.owl>
WHERE {
 ?mEvent rdf:type         :MotionSensorEvent ;
             :hasSensorID  ?sid .
 ?sid        :hasLocation  ?room .
}
```

Listing 1.5. A sample C-SPARQL query

that are processed in the query. Each C-SPARQL query can either generate new triples that can be processed by another query or call a (Java) listener class to trigger an action.

An interesting point to mention is that the C-SPARQL engine internally transforms the query into a dynamic part dealing with the event stream processing and a static part accessing the background knowledge. These parts are each individually executed by a suitable engine or query processor. This behav-

ior is transparent for the user as the entire rule is written in C-SPARQL and the rule result contains the combined execution outcome.

6 Comparison

In this section, we will investigate the capabilities of two introduced approaches of integrating stream processing and background knowledge. Based on our practical experiences, we discuss the two architectures from a software engineering perspective. Table 1 summarizes the results of the comparison. The criteria will be discussed in more details in the following paragraphs.

Table 1. Comparison of CQL (ESPER) and C-SPARQL

	ESPER	C-SPARQL
Maturity	+	–
Event Pattern Expressiveness	+	o
Conceptual Coherence	–	+
Dynamic Rules	o	+
Heterogeneous knowledge sources	o	–
Stream Reasoning Support	–	o

Maturity: ESPER is a widely used event processing engine, which is under development by an active open source community for many years and, consequently, has reached a stable and market-ready state. It provides a comprehensive documentation and several guides, as well as tutorials. In contrast, C-SPARQL, and the ready-to-go-pack in particular, is a conceptual prototype. This means that the implementation is not as mature and, furthermore, it is not as good documented as ESPER. So far, there are no published experiences about real-world projects using C-SPARQL.

Event Pattern Expressiveness: According to its maturity, ESPER provides a rich set of operators for specifying event patterns, e.g. for defining different types of sliding windows or various even aggregations operators. The event algebra of C-SPARQL is less expressive compared to ESPER, but, nevertheless, it supports all important features for general event processing tasks.

Conceptual Coherence: C-SPARQL allows the processing of stream data and the integration of static background knowledge by using only one paradigm (or language). Listing 1.5 shows a C-SPARQL query that combines event stream processing and SPARQL queries. In this sense, a C-SPARQL query is self-contained and coherent: only C-SPARQL skills are necessary for understanding it.

In contrast, ESPER does not support inherent access to knowledge bases. Consequently, specialized Java/Jena code must be written to integrate background data. The ESPER-based architecture combines the ESPER query language (EQL) for stream processing and Jena/SPARQL code implemented in a Java adapter class to query knowledge bases. The ESPER rules are not self-contained and delegate program logic to the adapter classes. Note that this can

also be viewed as an advantage: hiding a (perhaps) big part of the logic in method calls results in simpler and easier understandable rules.

Dynamic Rules: Changing a rule at runtime is difficult in ESPER, because modifying an ESPER rule can cause a change of the EQL pattern *and* of the SPARQL query in the listener class of the corresponding rule. In this case, the code must be recompiled. C-SPARQL makes changes much easier, because only the C-SPARQL query must be adjusted. Such queries are usually stored as strings in a separate file, which can be reloaded at runtime - even for rules including completely new queries of the knowledge base.

Heterogeneous knowledge sources: C-SPARQL is limited to ontological background knowledge stored in RDF format. In contrast, ESPER can be extended by arbitrary adapters allowing the usage of different knowledge sources. For instance, beside RDF triple stores also relational databases or NoSQL data sources can be used. However, the access methods have to be implemented and maintained by hand, as mentioned in the previous paragraph.

Stream Reasoning Support: Both approaches do not support stream reasoning, i.e. implicit knowledge is not automatically deduced when new events arrive. Conventional reasoners can only deal with static data, but not with high-frequent RDF streams. But, because (static) background knowledge changes infrequently, a conventional reasoning step can be processed, if a new fact in the static knowledge base appears.

Considering the two approaches from a conceptional point of view, C-SPARQL is better suited for inherent reasoning. For instance, SPARQL with RDFS entailment can be achieved by using materialization or query rewriting [7]. These approaches must be extended to stream processing. First discussions about this issue can be found in [15] and [3].

7 Conclusion

In this paper, we have discussed two different architectural approaches of integrating event stream processing and background knowledge.

The first architecture uses a CQL processing engine such as ESPER with an adapter class that performs SPARQL queries on a knowledge base. In this approach stream processing and knowledge engineering is conceptually and physically separated.

The second architecture is based on an extension of SPARQL to process RDF data streams. C-SPARQL allows integrated rules that process stream data and query RDF triple stores containing static background knowledge. Thus, C-SPARQL provides a more homogeneous approach, where query logic, event patterns and knowledge base access are combined in one rule and is, therefore, superior from a conceptional point of view.

Otherwise, CQL engines are well-established in real-world projects and at this time, they offer higher maturity and better performance. Therefore, CQL-based systems are (still) superior from a practical point of view.

Generally, the integration of semantic reasoning into stream processing is still an open issue that is not fully supported by any approach yet. Stream reasoning is therefore an important and promising research field to put effort in and has several work in progress, for example the appproaches in [3].

Acknowledgment

This work was supported in part by the European Community (Europäischer Fonds für regionale Entwicklung) under Research Grant EFRE Nr.W2-80115112.

References

[1] Anicic, D., Fodor, P., Rudolph, S., Stojanovic, N.: Ep-sparql: A unified language for event processing and stream reasoning. In: Proceedings of the 20th International Conference on World Wide Web. pp. 635–644. ACM (2011)
[2] Anicic, D., Rudolph, S., Fodor, P., Stojanovic, N.: Stream reasoning and complex event processing in etalis. Semantic Web pp. 397–407 (2012)
[3] Barbieri, D.F., Braga, D., Ceri, S., Della Valle, E., Grossniklaus, M.: Incremental reasoning on streams and rich background knowledge. ESWC pp. 1–15 (2010)
[4] Barbieri, D.F., Braga, D., Ceri, S., Grossniklaus, M.: An execution environment for c-sparql queries. In: Proceedings of the 13th International Conference on Extending Database Technology. pp. 441–452. EDBT (2010)
[5] Barbieri, D.F., Braga, D., Ceri, S., Valle, E.D., Grossniklaus, M.: Querying rdf streams with c-sparql. SIGMOD Rec. pp. 20–26 (2010)
[6] Bolles, A., Grawunder, M., Jacobi, J.: Streaming sparql - extending sparql to process data streams. In: The Semantic Web: Research and Applications, pp. 448–462 (2008)
[7] Glimm, B.: Using sparql with rdfs and owl entailment. In: Reasoning Web, pp. 137–201. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2011)
[8] Krötzsch, M., Simancik, F., Horrocks, I.: A description logic primer. CoRR (2012)
[9] Le-Phuoc, D., Dao-Tran, M., Xavier Parreira, J., Hauswirth, M.: A native and adaptive approach for unified processing of linked streams and linked data. In: The Semantic Web – ISWC 2011, pp. 370–388 (2011)
[10] Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley (2002)
[11] Prud'hommeaux, E., Seaborne, A.: Sparql query language for rdf, http://www.w3.org/TR/rdf-sparql-query/
[12] Renners, L., Bruns, R., Dunkel, J.: Situation-aware energy control by combining simple sensors and complex event processing. In: Workshop on AI Problems and Approaches for Intelligent Environments. pp. 29–34 (2012)
[13] Teymourian, K., Paschke, A.: Enabling knowledge-based complex event processing. In: Proceedings of the 2010 EDBT/ICDT Workshops. pp. 37:1–37:7. ACM (2010)
[14] Teymourian, K., Rohde, M., Paschke, A.: Fusion of background knowledge and streams of events. In: Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems. pp. 302–313. ACM (2012)
[15] Volz, R., Staab, S., Motik, B.: Incrementally maintaining materializations of ontologies stored in logic databases. In: Journal on Data Semantics II, pp. 1–34. Lecture Notes in Computer Science (2005)

Towards Explanation Generation using Feature Models in Software Product Lines

Dean Kramer, Christian Sauer, and Thomas Roth-Berghofer

School of Computing and Technology, University of West London,
St Mary's Road, London W5 5RF, United Kingdom
{first.lastname}@uwl.ac.uk

Abstract. Dynamic Software Product Line (DSPL) Engineering has gained interest through its promise of being able to unify software adaptation whereby software can be configured at compile time and runtime. Just like conventional adaptive software, software dynamism can confuse the user, and lower user trust. Variability knowledge expressed in a feature model though may not be understandable to the end user. Explanations have been shown to improve intelligibility of the software, and improve user trust. In this work, we consider how explanations can be used in DSPLs, by adding explanatory knowledge to feature models that can be used to generate explanations at runtime.

Keywords: Explanation Generation, Dynamic Software Product Lines, Feature Models

1 Introduction

Smart phones in recent years have seen high proliferation, allowing more users to stay productive while away from the desktop. It has become common for these devices to have an array of sensors including GPS, accelerometers, digital compass, proximity sensors, sound etc. Using these sensors with other equipment already found in phones, a wide set of contextual information can be acquired.

This contextual information can be used in Context-Aware Self Adaptive (CASA) software. This software can monitor different contextual parameters and dynamically adapt at runtime to satisfy the user's current needs [8]. These behavioural variations can be seen to share similarities with features in Software Product Lines (SPL), where product commonality and variability is handled, providing higher asset reuse. Within SPLs, Feature Oriented Software Development (FOSD) has emerged as a method for modularising the features of a system [3]. The one fundamental difference between these two concepts is that while SPLs conventionally manage static variability which is handled at compile time, adaptive software requires dynamic variability to be handled at runtime.

Dynamic Software Product Lines (DSPL) enables the SPL to be reconfigurable at runtime [9]. By using DSPLs, variability can be static, adapted at compile time, or dynamic and adapted at runtime. This allows for greater reuse

as variability can be implemented for both static and dynamic adaptation, as different products may require the adaptation to be applied at different times [14].

Feature Modelling has become the *de facto* method of variability representation, used in software product lines. In feature models, the adaptation of the product, be it static, or dynamic, are modelled, enabling a wide variety of products and product behaviours. While feature modelling is of great use in the development, the dynamics within feature modelling can be confusing to end-users. To amend the seemingly unpredictable and thus confusing nature of the behaviour of a dynamic system and the results it produces, it is desirable to enable the system to explain its behaviour as well as the results it produces to the end-user. As we will detail further on in this paper explanations are very useful to justify results a system produces and thus help to rebuild the trust an end-user has in the systems behaviour and results. So explanations are useful to the end-user as they can counter the mentioned non-transparency of DSPL end-products and their dynamic behaviours.

In our previous work [18], on enabling a system we developed to generate explanations, we investigated the integration of explanations into a context acquisition engine, used for developing context-aware applications. We did this with regard to mobile applications were one has to adhere to many constraints. We developed a *ContextEngine* to easier deal with such limitations and situation-specific information across applications [12], thus easing the creation of context-aware, mobile systems. We noticed that with the increased adaptability and dynamics of context-aware applications came an increase in complexity of the application, which in turn made it harder to understand the behaviour of such applications. In our initial research on this topic we then described how we enhanced the *ContextEngine* platform with explanation capabilities. As we describe in this paper and as it was proven in a variety of other work on explanations, explaining can be seen as complex reasoning task on its own. In our initial work we focused on the use of *canned explanations*. *Canned explanations* are information artefacts, pre-formulated by the software engineer, that serve as explanatory artefacts stored in the system and delivered to the user on demand. We integrated storage facilities for such information artefacts, or explanatory knowledge artefacts within the code structure of the *ContextEngine* and thus were able to provide these stored canned explanations on demand to a software engineer working with the *ContextEngine*. After this early steps and relatively simple approach, based also on a further study into the matter of explanation provision in the feature model and especially in the automated analysis feature models (AAFM) domain, we decided to elaborate on our initial work.

The rest of the paper is structured as follows: We introduce the feature modelling background of our work in the following section and based on the technological possibilities described there motivate our approach to use an extended feature model for explanation generation in Section 3. We then interlink our approach with related work on feature modelling, explanation generation and the use of explanations itself in the following section. We then introduce our approach to explanation generation from explanatory knowledge stored in an

extended feature model and demonstrate our concept of explanation generation in Section 5 . After discussing the advantages and possible disadvantages of our approach in Section 6 a summary and outlook on future aspects of our work concludes the paper.

2 Feature Models

The purpose of a feature model is to represent all possible products from a SPL in terms of features, and the relationships between them. An example feature model for a content store application is shown in Figure 1. A *feature* of a system

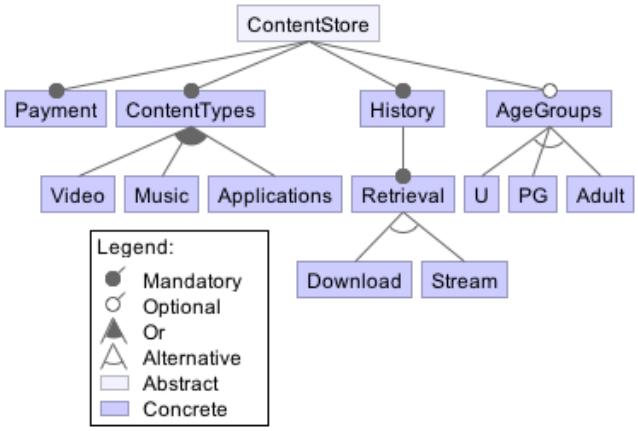


Fig. 1. Feature Model of a content store

has been described in a number of variations [2]. For this paper, we use the definition by Kang et al. [10] in that a feature is "*a prominent or distinctive user-visible aspect, quality, or characteristic of a software system or systems*". Feature models are modelled using hierarchical trees of features, with each node representing commonality and variability of its parent node. Relationships between each feature can be modelled using:

- Tree feature relationships between parent (compound) features and their child features (subfeatures) .
- Cross-tree constraints which typically apply feature inclusion or exclusion statements, normally using propositional formula. An example of this includes "if ABC is included, then feature XYZ must also be included."

Within feature models, different feature relationships can be applied including:

- **Mandatory**. A child feature is defined as mandatory in all products where its parent is also contained.

– **Optional**. A child feature is defined as optional when it optionally can be included or excluded when its parent is contained in a product.
– **Or**. A set of child features exhibit an or-relationship when one or more children are selected along with the parent of that set.
– **Alternative (XOR)**. A set of child features exhibit an xor-relationship when only a single child can be selected when the parent is included.

Feature models have been applied not only to modelling system features, but also context [1]. As DSPLs can be driven by context, modelling both contexts and the features that they affect in feature models allows for a single modelling language. The feature models introduced above represent what is known as *basic feature models*. There have been different additions to feature modelling, including *cardinality feature models* [7], and *extended feature models* [6].

Extended feature models extend basic feature models by the ability to attach additional information about features to the model. This additional information is included by the use of *feature attributes*, which are attached to features within the feature model. Feature attributes normally consist of a *name, domain,* and *value*. Feature attributes have been used in previous work for specifying extra-functional information [4]. We intend to also use feature attributes in our approach. We will employ additional feature attributes to store explanatory knowledge artefacts, see section 5.1 for details.

3 Motivation of our work

The GUI of an application, or even more intriguing, the behaviour of an application generated by the use of SPL can be rather dynamic. This dynamic behaviour can be confusing if not daunting to the end-user of the application. The end-user might not be aware of why the GUI has adapted and the factors influencing how it changes. Furthermore, the dynamic behaviour of the application, producing different results while receiving identical inputs just under different circumstances (for example a network being available or not), is a challenge to the trust the user develops towards the applications results. As the feature model, being the component responsible for the dynamic behaviour of the application, is a black box system to the end-user the need for explanations of this black box systems behaviour arises.

The benefits of being able to explain the behaviour of an application and subsequently its GUI are plenty. According to [17] there are a number of benefits explanations can provide to the end-user. The main benefits of interest with regard to our problem at hand are the generation of trust into the results the application generates and justification of and guidance on the changes in the GUI of the application.

As [6] have shown it can be a complicated process to apply abductive reasoning to generate minimal explanations from the logical representation of a feature model within an AAFM. To circumvent the effort involved in using abductive reasoning to generate a minimal explanations from the logical representation of the feature model our approach aims at integrating canned 'micro' or 'atomic'

explanations within the logical representation of the feature model. By doing so we aim to re-use the feature model itself in the same way it is used in the product configuration to also 'configure' or synthesise more complex explanations of the actual product generated from the 'atomic' building blocks given by the canned 'micro' explanations embedded in the feature descriptions themselves as well as in the representation of the relationships between these features described in the feature models logical representation.

3.1 Scenario Application

To illustrate our motivation, consider a DSPL example of a content store application for a mobile device. This application may provide different content for the user including applications, movies, music etc. Different content is organised into different categories. A simplified feature model of the DSPL can be seen in Figure 1. This application provides content for different age groups, and also the application can be tailored to suit these different groups.

In the feature model, we can see that the features *Payment, ContentTypes, History,* and *Retrieval* are required in every configuration of this DSPL. The *Payment* feature handles all payment transactions when content is bought or rented. The *ContentTypes* feature contains the different components for browsing, and buying different types of content. Because different regions in the world may require different content distribution licenses, it may not be possible to sell content in every region, so depending on the location of the user, different content type features including *Video, Music,* and *Applications* will be bound or unbound. In the *History* feature, all bought content is found, which can be retrieved in *Retrieval*. There are two primary methods in which content can be retrieved, downloaded or streamed. Certain content including video maybe downloaded or streamed. Depending on how much storage is available on the device, it may be not be possible to download the movie, so only the *Streaming* feature is bound. In Figure 2, we can see the variability of the screen according to content type features. If you consider the video feature, there is a button that takes the user to a set of screens for video content, and also a containership of widgets for advertising popular movies.

4 Related Work

Explanations and feature models have been used before, but more to aid the analysis process and error analysis of feature models [20] as well as in automated feature model analysis in general as Benavides et al. describe in [5]. As we already mentioned there are a number of goals that can be reached by providing explanations to the user of a, then, explanation aware system. An explanation aware system is a system that is able to provide explanations of the results it produces as well as of the means it employs to produce these results [11,13].

The goals pursued by enabling a system to provide explanations [19] are the following: Increase the transparency of a systems reasoning process to increase

Fig. 2. Variability of the main screen

the users trust into the system. Justifying results the system produces. This goal aims at explaining the quality and applicability of results the system produced to the end-user. Another goal of explanation provision is to provide relevance explanations of either question asked by the system or on information provided by the system. Conceptualisation, thus explanations of the concepts the system is working on, directly aids the last goal of providing explanations, learning. By explaining the concepts the system works on to the end-user the end-user is enabled to learn about the domain in which the system works.

Our approach to providing applications needs, next to the knowledge used by the feature model system, additional explanatory knowledge to create the explanations we want the system to be able to provide to the end-user. It is always necessary to provide explanatory knowledge in any system that is intended to provide explanations on its reasoning [16]. This additional explanatory knowledge is provided to and used by the explainer component of an explanation aware system, enabling it to provide explanations of its reasoning and results. The need for additional explanatory knowledge is also shown in [20] as the abduction process described there is relying also on additional explanatory knowledge.

In our approach the explanatory knowledge needed to generate explanations in our system will be broken down into 'atomic' canned explanatory knowledge artefacts that will be paired with each feature of the feature model as well as additional 'atomic' canned explanatory knowledge artefacts that will be attached to the relationship descriptors within our feature model. The aim of this 'enrichment' or 'dotation' of the feature model with 'micro' or 'atomic' explanatory

knowledge artefacts is it to reuse the artefacts 'bound' to the features and their relationship descriptors in the final product to synthesise complex explanations based on the available 'bound' atomic explanatory knowledge artefacts. We focus our approach especially on the issue of explaining a dynamic GUI to the end-user. As for example [15] described in their work the problems that can result from a dynamic and automatically assembled GUI, we aim to amend these problems by providing the end-user of an application with an insight into the GUI's changes by explaining them to her.

5 Our Approach

In our approach, we attempt to enable explanations in DSPLs. By adding explanations to DSPLs, we see two benefits. Firstly, explanations have been shown in other work to improve user understanding of a system which can be applied to DSPL systems [13]. Secondly, because a SPL enables many products to be produced using reusable common assets, we can then easily produce many explanation-ware applications, because we can leverage the reuse properties of the SPL with the explanations. The first part of our approach regards the modelling of the system.

5.1 Modelling

Just as the rest of the system is modelled using feature models, so too are the explanations. To add explanations to the feature model, we use extended feature models. As introduced earlier in the paper, with extended feature models, extra explanatory information can be attached to features using feature attributes. These feature attributes can be used for storing the specific explanation snippets for each feature. For each feature attribute there is a name, domain, and value. The domain of the attribute holds what type of explanation it is, with the value holding the explanation fragment.

Mapping explanatory knowledge fragments to features is not just enough; we also need to map explanatory knowledge fragments to feature model relationships. Examples of explanatory knowledge fragments mapped to relationships include:

- Mandatory - "in all configurations".
- Optional - "which is optional".
- Or - "which can be".
- Alternative - "which is either".

5.2 Composing Explanations

Once we have the explanations added to the feature model, we can compose complex explanations. These complex explanations are made up of the concatenation of explanations added to the features and the relationship explanations.

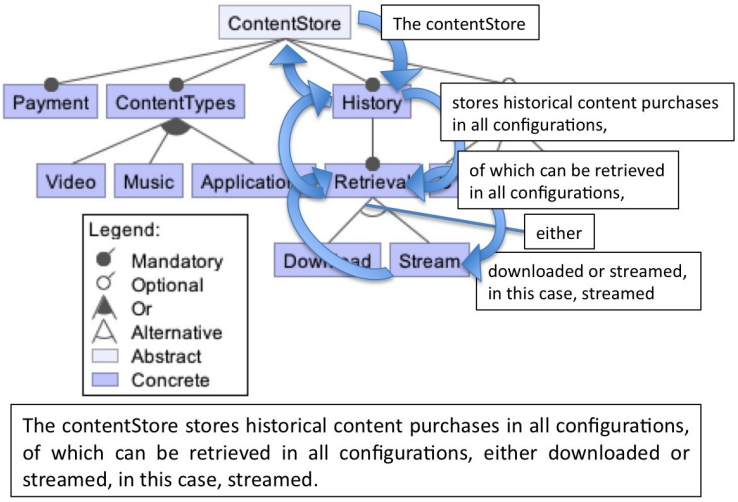


Fig. 3. Composing Explanation of why streaming is active

Lets take the example of considering a configuration where content is streamed to the user instead of downloaded as shown in Figure 3. If the user wanted to know *why* streaming is active, to generate an explanation we firstly follow the tree from the streaming feature to the root. We then get the conceptual explanation of the root, in this case "The content store". Next we add the conceptual explanation for the "History" feature, in this case "Stores historical content purchases", and because it is a mandatory feature, we add "in all configurations". Following this, we add the conceptual explanation for the "Retrieval" feature, in this case "of which can be retrieved", and add "in all configurations" because the feature is mandatory. Then because the sub features of "Retrieval" are alternatives, we add "either", and the two conceptual explanations joined with an "or". Lastly, because in the configuration, the "Stream" feature is active, we add "in this case, streamed". We therefore can 'reuse' the structural information encoded in the feature model representation directly for the composition of complex explanations by simply 're-tracing' the explanatory knowledge artefacts, stored in feature and relationship nodes, along a path in the feature model.

6 Discussion

The implementation effort is expected to be minor, given the fact that our approach just adds three additional feature attributes to the feature and relationship descriptions. The intention of 'piggyback' riding the inherent logical structure encoded in the feature model graph to also derive complex explanations from it is still open to be tested for its actual quality of generated explanations as well as for its scalability. With regard to the scalability of our approach we intend it to be limited. Once a feature model exceeds a certain complexity the

coherent concatenation of explanatory knowledge artefacts described in the feature and relation nodes along a path in such a model will fail or become too much of a computational effort. However we assume that for small to medium scale feature models our relatively 'practical' approach of concatenating explanatory knowledge artefacts stored in the models nodes is relatively efficient compared to existing more complex approaches of explanation generation explained, for example, in [6].

7 Summary and Outlook

In this paper we presented how the variability of SPL based products and their behaviours could be explained to their end-users using explanations composed from explanatory knowledge added to enhanced feature model nodes. Based on the feature modelling background of our work we motivated our approach to use an extended feature model for swift explanation generation. We reviewed and compared our approach with related work on feature modelling, explanation generation and the use of explanations itself, especially inspecting approaches employing relatively complex abductive reasoning. We then introduced our approach to explanation generation from explanatory knowledge stored in extended feature model nodes explaining features themselves and their relationships. By tracing an example graph in a feature model we showed the working of our approach. Given the fact that we are in the early stage of further examining our approach we then discussed its possible limitations but also its possible advantages given by the fact that our approach promises to be easily implemented base on existing work on enhanced feature models and is very easy to use for explanation composition in small to medium sized feature models, compared to more complex approaches like abductive reasoning.

As we cannot yet predict the effectiveness of our seemingly 'pragmatic' approach, we have to implement the enhanced node generation in our existing enhanced feature model and then perform a series of experiments on this enhanced feature model. The aim of these experiments will be to establish our approaches boundaries with regard to parameters such as quality and usability of generated explanations as well as the scalability of our approach. We also have to measure the computational effort necessary for explanation composition and then measure it against the gain in usability to establish the worthiness of further researching our approach of reusing extended feature model structures for explanation generation from explanatory knowledge artefacts stored in the nodes of the feature model.

An additional feature of the reuse of an enhanced feature models structure for explanation generation not yet investigated further is the reuse of propositional logic formulae derived from the feature model. We plan to investigate the possibilities of this reuse in our immediate follow up research.

References

1. Acher, M., Collet, P., Fleurey, F., Lahire, P., Moisan, S., Rigault, J.P.: Modeling Context and Dynamic Adaptations with Feature Models. In: 4th International Workshop Models@run.time at Models 2009 (MRT'09). p. 10 (Oct 2009)
2. Apel, S., Kästner, C.: An overview of feature-oriented software development. Journal of Object Technology 8(5), 49–84 (2009)
3. Batory, D., Sarvela, J., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Softw. Eng. 30, 355–371 (June 2004)
4. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Inf. Syst. 35(6), 615–636 (Sep 2010)
5. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. Information Systems 35(6), 615–636 (2010)
6. Benavides, D., Trinidad, P., Ruiz-Cortés, A.: Automated reasoning on feature models. In: Proceedings of the 17th international conference on Advanced Information Systems Engineering. pp. 491–503. CAiSE'05, Springer-Verlag, Berlin, Heidelberg (2005)
7. Czarnecki, K., Helsen, S., Ulrich, E.: Formalizing cardinality-based feature models and their specialization. Software Process: Improvement and Practice 10, 7 – 29 (01/2005 2005)
8. Daniele, L.M., Silva, E., Pires, L.F., Sinderen, M.: A soa-based platform-specific framework for context-aware mobile applications. In: Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M.J., Szyperski, C., Poler, R., Sinderen, M., Sanchis, R. (eds.) Enterprise Interoperability, Lecture Notes in Business Information Processing, vol. 38, pp. 25–37. Springer Berlin Heidelberg (2009)
9. Hallsteinsen, S., Hinchey, M., Park, S., Schmid, K.: Dynamic software product lines. Computer 41, 93–95 (April 2008)
10. Kang, K., Cohen, S., Hess, J., Nowak, W., Peterson, S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21 (1990)
11. Kofod-Petersen, A., Cassens, J.: Explanations and context in ambient intelligent systems. In: Modeling and Using Context, pp. 303–316. Springer (2007)
12. 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. pp. 6:1–6:7. M-MPAC '11, ACM, New York, NY, USA (2011), http://doi.acm.org/10.1145/2090316.2090322
13. Lim, B.Y., Dey, A.K., Avrahami, D.: Why and why not explanations improve the intelligibility of context-aware intelligent systems. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 2119–2128. ACM (2009)
14. Parra, C.: Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations. Ph.D. thesis, INRIA Lille Nord Europe Laboratory (March 2011)
15. Pleuss, A., Hauptmann, B., Dhungana, D., Botterweck, G.: User interface engineering for software product lines: the dilemma between automation and usability. In: Proceedings of the 4th ACM SIGCHI symposium on Engineering interactive computing systems. pp. 25–34. ACM (2012)
16. Roth-Berghofer, T.R.: Explanations and case-based reasoning: Foundational issues. In: Funk, P., Calero, P.A.G. (eds.) Advances in Case-Based Reasoning. pp. 389–403. Springer-Verlag, Berlin, Heidelberg, Paris (September 2004)
17. Roth-Berghofer, T.R., Cassens, J.: Mapping goals and kinds of explanations to the knowledge containers of case-based reasoning systems. In: Case-Based Reasoning Research and Development, pp. 451–464. Springer (2005)

18. Sauer, C., Kocurova, A., Kramer, D., Roth-Berghofer, T.: Using canned explanations within a mobile context engine. Explanation-aware Computing ExaCt 2012 p. 26 (2012)
19. Sørmo, F., Cassens, J., Aamodt, A.: Explanation in case-based reasoning–perspectives and goals. Artificial Intelligence Review 24(2), 109–143 (2005)
20. Trinidad, P., Benavides, D., Durán, A., Ruiz-Cortés, A., Toro, M.: Automated error analysis for the agilization of feature modeling. J. Syst. Softw. 81(6), 883–896 (Jun 2008)

Towards Continuous Knowledge Representations in Episodic and Collaborative Decision Making

Joachim Baumeister[1], Albrecht Striffler[1], Marc Brandt[2] and Michael Neumann[2]

[1] denkbares GmbH, Friedrich-Bergius-Ring 15, 97076 Würzburg, Germany
{firstname.lastname}@denkbares.com
[2] The Federal Environment Agency (Umweltbundesamt), Section IV 2.3 Chemicals
Wörlitzer Platz 1, 06844 Dessau-Roßlau, Germany

Abstract. With the success of knowledge-based approaches in decision support systems new requirements arise in practice. That way, users demand not only for the collaborative development of such systems, but also for the collaborative and episodic use in decision processes. Moreover, in complex decision domains multiple knowledge representations are available that need to be jointly processed. In this paper we introduce a novel approach and a system implementation that aims to meet these requirements.

1 Introduction

In the past, decision support systems based on knowledge bases emphasized the explicit representation of decision knowledge for its automated application in the target scenario. Typically, those systems are used monolithically by one user or automated by a machine. Examples are for instance the medical consultation system SonoConsult [12], the medical therapeutic system SmartCare [6], and TIGER [8] for the monitoring of gas turbines. With the success of those systems new requirements arise to adapt into new environments. Advanced requirements are as follows:

- *Collaborative use:* More than one person is working on the same decision process at the same time.
- *Episodic use:* The actual decision process is not a one-step question-answer interview, but needs (sometimes sporadically) input over time, i.e., a *decision episode*.
- *Mixed representation:* Systems are build from knowledge bases that do not use a single knowledge representation (e.g., rules) but a combination, for instance rules with models and ontologies.

The requirements stated above call for extensions of todays systems in the following manner:

- A systematic extension of systems that support the collaborative and the episodic decision making. Here, especially an approach of representing the provenance of decisions is required.

– A continuous knowledge representation to support heterogenous representations for decision making and its episodic application. Here, the already introduced *knowledge formalization continuum* [2] needs to be reconsidered in the light of its use in decision making.

In this paper, we try to shed more light into fulfilling the requirements mentioned above. The formalization and use of the knowledge formalization continuum is introduced in Section 2. In Section 3 we discuss a systematic approach for episodic decision making in collaborative use. A case study in Section 4 exemplifies the successful application of the described approach. The overall ideas are summarized and concluded in Section 5.

2 Continuous Knowledge Representation and Application

One main challenge in complex decision making is finding the appropriate scope of the knowledge base: Complex domains require a large number of aspects to be considered. Thus, a 'complete' knowledge base needs to include many aspects, to be later useful in practice. Most of the times however, not all aspects can be included in the knowledge base:

– *Uncertain domain knowledge:* Parts of the domain are not well-understood in a technical sense. Here, decisions in practice are often based more on past experience, evidence, and intuition than on strict domain laws and rules.
– *Bloated domain knowledge:* For some parts of the domain, the explicit representation of the knowledge would be too time-consuming and complex. For instance, much background knowledge needs to be included, that is required for proper decision making. Here, the expected cost-benefit ratio is low, e.g., because many parts will be rarely used in real-world decisions[1].
– *Restless domain knowledge:* Especially in technical domains, some parts of the domain knowledge are frequently changing due to technological changes. The explicit representation of these parts would require frequent maintenance. Here, also the cost-benefit of the maintenance vs. the utility of the knowledge needs to evaluated.

In this section we introduce an approach that allows for the combined representation and use of knowledge at a varying formalization granularity, i.e., the *knowledge formalization continuum*. The main idea of the knowledge formalization continuum is to use varying knowledge representations for one knowledge base and to select the best-fitting representation for each partition. Besides the representation of different knowledge representations, the approach also considers the mixed application of and reasoning with knowledge at different formalization levels.

[1] Costs for developing/maintaining the knowledge vs. the benefit/ frequency of using the single parts in practice

2.1 The Knowledge Formalization Continuum

In general, the *knowledge formalization continuum* is a conceptual metaphor extending the knowledge engineering model for a domain specialist. The metaphor emphasizes that entities of a knowledge base can have different facets ranging from very informal representations (such as text and images) to very explicit representations (such as logic formulae), see Figure 1. Here, it is not necessary to commit to a specific knowledge rep-

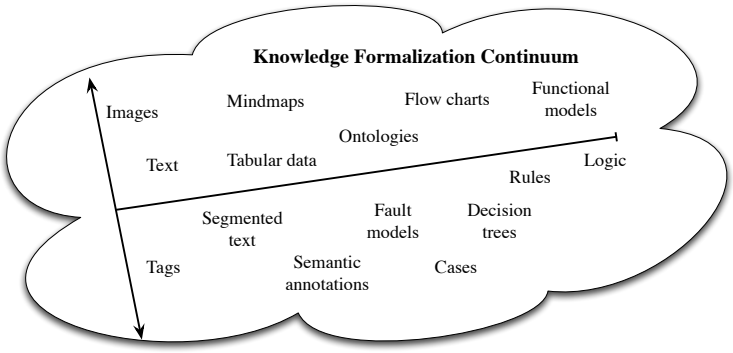


Fig. 1. An idealistic view of the knowledge formalization continuum.

resentation at the beginning of a development project. Rather, it supports concentrating on the actual knowledge by providing a flexible understanding of the knowledge formalization process. Specific domain knowledge can be represented in different ways, where adjacent representations are similar to each other, e.g., tabular data and cases. More extreme representations are much more distinct, e.g., text vs. logic rules. It is important to note that the knowledge formalization continuum is neither a physical model nor a methodology for developing knowledge bases. Rather, the concept should help domain specialists to see even plain data, such as text and multimedia, as first-class knowledge that can be transformed by gradual transitions to more formal representations when required. On the one hand, data given by textual documents denote one of the lowest instances of formalization. On the other hand, functional models store knowledge at a very formal level.

When working with different representations of knowledge one has to keep in mind, that every granularity of formalization has its advantages and disadvantages. On the informal side, textual knowledge can be easily acquired and it is often already available. No prior knowledge with respect to tools or knowledge representation is necessary. However, (automated) reasoning using textual knowledge is hardly possible. The knowledge can only be used/retrieved through string-based searching methods. The formal side proposes rules or models as knowledge representation; here automated reasoning is effective but the acquisition of such knowledge is typically complex and

time-consuming. Further, the knowledge engineer needs to "model" the knowledge in a much more precise manner.

The knowledge formalization continuum embraces the fact that knowledge is usually represented at varying levels of formality. A system supporting the knowledge formalization continuum should be able to store and work with different representations, and it should support transitions between the representations where its cost-benefit ratio is (in the best case) optimal.

In typical projects, prior knowledge of the domain is already at hand, often in the form of text documents, spreadsheets, flow charts, and databases. These documents build the foundational reference of the classic knowledge engineering process, where a knowledge engineer models domain knowledge based on these documents. The actual utility and applicability of knowledge usually depends on a particular instance. The knowledge formalization continuum does not postulate the transformation of the entire collection into a knowledge base at a specific degree but the performance of transitions on *parts* of the collection when it is possible *and* appropriate. This takes into account the fact that sometimes not all parts of a domain can be formalized at a specific level or that the formalization of the whole domain knowledge would be too complex, considering costs and risks.

2.2 Reasoning in the Knowlegde Formalization Continuum

When using different types of knowledge representations the most important question is how to connect these elements when used during the reasoning process.

Pragmatic Reasoning As a pragmatic approach to be used in decision support systems, we propose to define a taxonomy of decisions and connect entities of knowledge (*knowledge elements*) with decisions of the decision taxonomy. See Figure 2.2 for an exampled depiction. Here, the knowledge base contains rules, workflow models, and

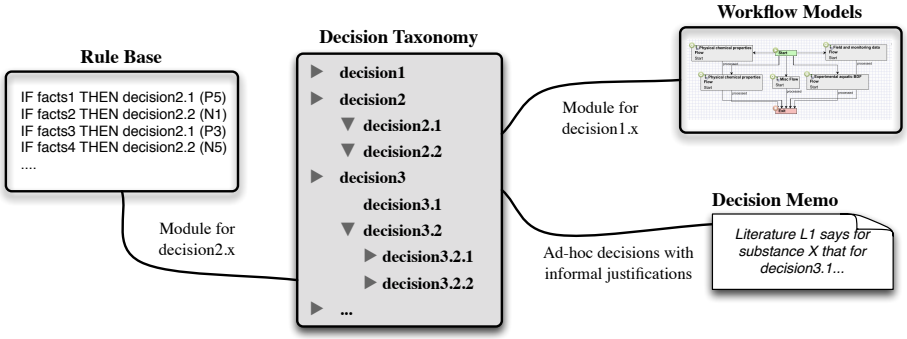


Fig. 2. An example for connecting different knowledge elements with a decision taxonomy.

textual decision memos. All elements reference the same collection of decisions and

thus can jointly infer decisions. When a knowledge element is activated during decision making–the knowledge element fires–then the corresponding decision element is established and presented as derived decision.

Please note, that more formal approaches like RIF [16] do use a comparable connection, i.e., decisions are formalized as concepts/instances and rules are defined to derive the existence of the concept/instance.

Decision Making using Scoring Weights With the simple approach sketched above, decisions can be only taken categorically. For a more leveled approach, we propose to introduce scores as weights for decisions. Scores are a well-understood weighting scheme in knowledge engineering [11, 7] and has a simple reasoning semantics: Each decision has an account which stores the scoring weights given to the decision by knowledge elements during the reasoning process. When a knowledge element "fires", then the corresponding score is added to the account of the particular decision. Scoring weights included in the account are aggregated in a predefined manner. A decision element is established and shown as derived decision, when the aggregated scoring weight exceeds a given threshold.

Example: Often a reduced set of score weights S = {N3, N2, N1, 0, P1, P2, P3} is sufficient for developing large knowledge bases. Given the weight categories a developer can select from seven weights N1 (weakly negative) to N3 (excluded) for negative scoring and seven weights P1 (weakly positive) to P3 (clearly established) for positive scoring. The weight 0 represents an unclear state. The score weights of a decision account are aggregated as follows: The sum of two equal weights results in the next higher category, e.g., P2 + P2 = P3. Positive and negative weights are aggregated, so that two equal score weights nullify each other, e.g., P2 + N2 = 0. A decision is established (confirmed), if the aggregation of the collected scoring weights exceeds the category P3.

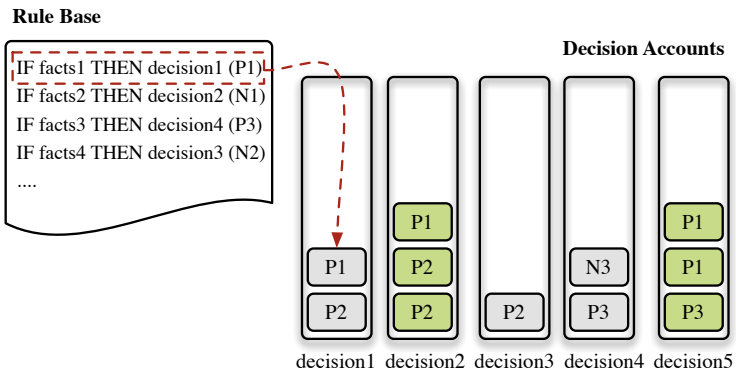


Fig. 3. Exemplary score accounts for five decisions.

In Figure 3 the accounts of five decisions and an excerpt of a rule base are shown. One rule fires and adds the weight P1 to the account of `decision1`. We see that `decision2` and `decision5` are established, since the aggregation of their collected scoring weights exceeds the weight P3. In contrast, `decision4` is not established because of the negative weight N3.

3 Episodic and Collaborative Decision Making

Complex decisions often are not made by taking one step, but are typically divided into a number of sub-decisions. Each of them may need further research and collaborative interaction for clarifying details. Collaboration is necessary when a complex decision can only be made by joining experts from different domains into the decision process. These requirements can be fulfilled by specific extensions of a decision support system:

1. Contemporary access to the data and decisions.
2. Episodic collaboration during decision making.
3. Provenance of data and decisions.

3.1 Contemporary Access

Authorized persons need to be able to access the system at the same time. They should be able to work with the system in order to make decisions or to retrieve already taken decisions. Contemporary access can be provided by a web-based implementation of the system, as for example implemented by semantic wiki systems [13]. Further examples are collaborative ontology development environments such as WebProtégé [9, 14].

In such a distributed setting we need to consider concepts like rights management for access control, revision management of old versions of the knowledge, and conflict management of simultaneous edits.

3.2 Episodic Collaboration

Authorized persons should be able to enter data for making a particular decision. The data entry needs not to be made at one time but can be partitioned over multiple sessions, i.e., decision episodes. Also, different users can enter data used for the same decision.

3.3 Provenance of Data and Decisions

When more than one person contributes to a complex decision making process and when the process is partitioned into episodes, then the process and reasoning should be traceable and understandable by the users. This implies the documentation of the decisions including their history but also the provenance of the data used for making the decision (see below). Therefore, the system needs to provide versioning of the decisions made including a documentation by the respective users. When representing the history and documentation of decisions by an ontology, then known approaches can be applied, for instance [10, 4].

Provenance of data and decisions is needed in collaborative and episodic environments. Here, the following questions need to be clearly answered:

– At which time was a particular data element entered?
– Who entered the data?
– Which knowledge elements are responsible for a particular decision?
– What is the history of a particular data and decision?
– Which persons contributed to the process of a particular decision?

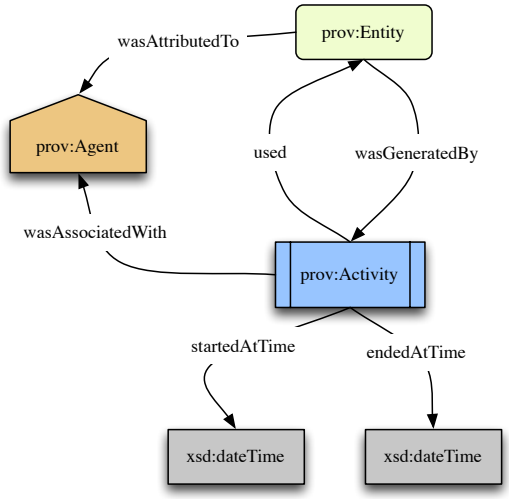


Fig. 4. Simple version of the PROV ontology.

We propose the application of the PROV ontology [15] to knowledge elements and the entities of the decision process. That way, an extensible and standardized ontology is used to represent the use and origin of decisions. In Figure 4 the three Starting Point classes and properties of the PROV ontology are depicted. Here, an `prov:Agent` is responsible for taking an `prov:Activity`. The `prov:Activity` generates an `prov:Entity`, but instances of `prov:Entity` can be also used in (other) instances of `prov:Activity`. An `prov:Entity` is a general representation for a thing, being physical, digital, conceptual, or any other kind of interpretation. We can see that answers to the questions stated above can easily be represented using the simple version of the PROV ontology, when people involved in the decision making process are represented as `prov:Agent` instances, entered data and the decisions themselves are represented as `prov:Entity` instances, and the data entry and decision episodes are represented as `prov:Activities`.

4 Case Study: KnowSEC – A System for Managing Chemical Substances of Ecological Concern

In this section we describe the KnowSEC project and its corresponding tool. KnowSEC stands for "Managing Knowledge of Substances of Ecological Concern" and it is used to support substance-related work and workflows within a unit of the Federal Environment Agency (Umweltbundesamt). More precisely, the tool supports decisions by a number of knowledge-based modules. In the context of the KnowSEC project only substances under REACH[2] are considered. For the implementation of the KnowSEC project the semantic wiki KnowWE [3] was extended by KnowSEC-specific plugins. KnowWE is a full-featured tool environment for the development of diagnostic knowledge bases and RDF(S) ontologies. It provides plugins for automatically testing and debugging knowledge bases including continuous integration. For a recent overview we refer to [1].

Many of the requirements stated in the introduction of this paper apply to the KnowSEC project: The group is divided into sub-groups; each sub-group collaboratively works on a number of substances. For each substance under consideration a number of complex decisions need to be made concerning the safety and regulation of the substance. Decision making on a substance sometimes can take a couple of months or even years, therefore support for episodic decision making is required.

4.1 Substances as Wiki Instances

Since the single substances are the primary target of decision making, every substance under consideration is represented by a distinct (semantic) wiki article. The article stores relevant information of the substance such as chemical end-points, relevant literature, and comments of group members. The information is entered by group members using (user-friendly) editors. In the background the information is silently translated into an ontology representation for automated reuse and processing. That way, any information (e.g., alternative identifiers, end-points, paragraphs, comments) is represented as an RDF triple. Consequently, the visualization of the latest changes and specific overviews are easily defined by SPARQL queries [17]. The article of the imaginary substance "Kryptonite" is depicted in Figure 5. The article is maintained for demonstration purposes and reflects by no means any real work of the agency.

At the right of the article all decision work on the substance "Kryptonite" is displayed giving a summary of the currently taken (sub-)decisions, the comments by group members, and a fact sheet showing the identifiers of the substance.

At the time of writing, KnowSEC stores more than 11,000 substances as separate wiki articles including a number of critical chemical characteristics. A small part of these substances are currently under decision making.

[2] REACH stands for the European Community Regulation on chemicals and their safe use (EC 1907/2006). The regulation handles the registration, evaluation, authorization, and restriction of chemical substances.

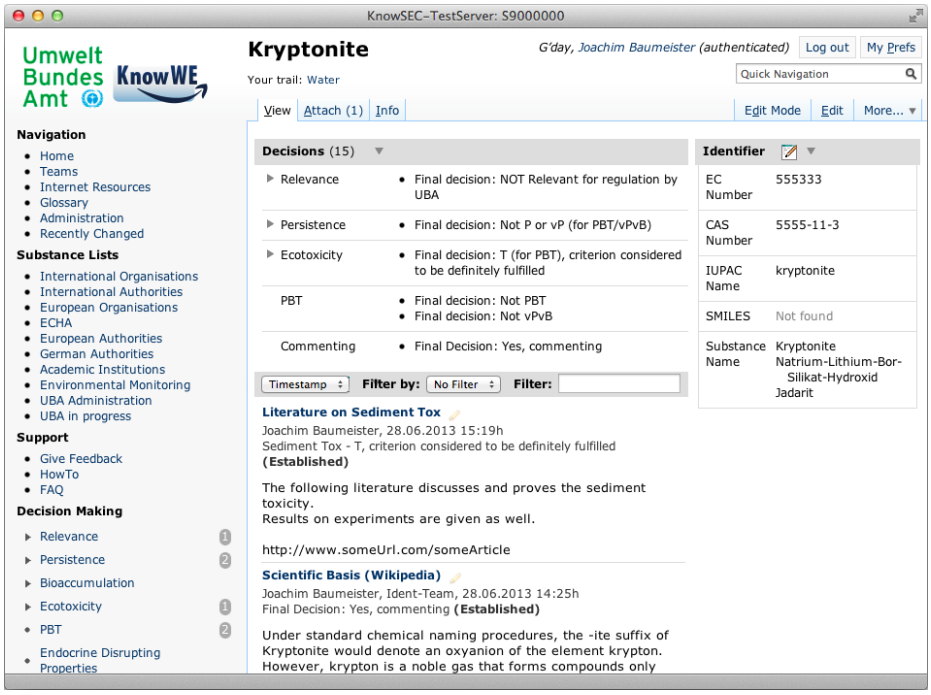


Fig. 5. The article describing the imaginary substance "Kryptonite".

4.2 Continuous and Collaborative Decision Making

When displaying a substance article in KnowSEC, the left menu of the wiki is extended by a decision making bar; see Figure 5. Here, all decision aspects are listed that are relevant for the working group. When clicking on a decision aspect, the sub-aspects of the selected aspect are shown. By selecting one of these (sub-)aspects the user can initiate a decision form, where specific questions of the aspect are asked and decisions are proposed automatically. These interactive decision forms are generated by explicit knowledge represented by scoring rules or DiaFlux models [5]. Any data entry and taken decision is recorded by KnowSEC including the time and user. An explanation component shows the justifications of taken decisions by visualizing the supporting data and the acting users of the data. For the explanation the PROV ontology—as described in Section 3.3—is applied. Users, i.e., team members, are instances of `prov:Agent` and entered data and (decision) memos are instances of `prov:Entity`. The creation or edit of a (decision) memo and an interactive decision form are represented as `prov:Activity` instances including the corresponding edit times. A simplified depiction of this application is shown in Figure 6; the prefix `dss` (decision support system) stands for the KnowSEC ontology namespace.

Explicit Knowledge and the Taxonomy of Decisions From the technical point of view, the explicit part of the knowledge base is partitioned into modules, that are con-

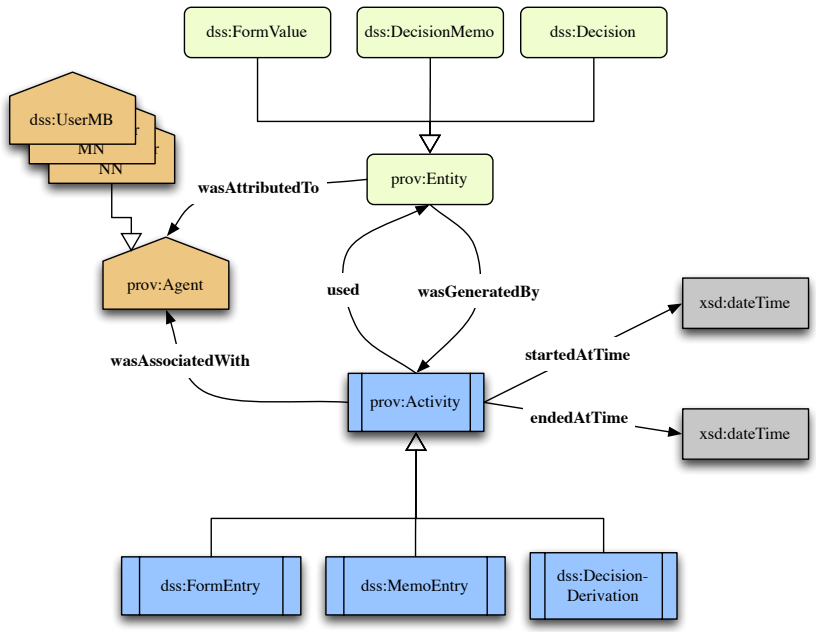


Fig. 6. Simplified version of the applied PROV interpretation for tracing the provenance of decisions.

nected by a taxonomy of decision instances. Since the taxonomy is represented as an RDF ontology, it is strongly connected with the ontology of the article information (see paragraph above). The formal versions of the aspects are implemented by knowledge base modules and connected by the taxonomy of decision instances. Some modules are using decision trees, other modules use scoring rules, also RDF ontologies are used.

Decision Memos For some aspects and decisions, respectively, no explicit knowledge base is available. When the user wants to document a decision without using an explicit knowledge base, he/she is able to create a *decision memo*. A decision memo is, entered by an authorized user and consists of free text, some meta-data (e.g., tags, time, etc.), and an explicit decision with a scoring weight. The decision memos are attached to the article of the corresponding substance. The included decision is used in the overall reasoning process. A decision memo is an implementation of an implicit reasoning element of the knowledge formalization continuum. An example of a decision memo can be the note of a group member that a particular aspect was proven by a specific experiment giving the reason for deriving a specific (sub-)decision. For instance, see the decision memos about the persistence of the substance "Kryptonite" being created in Figure 7. Decision memos are automatically attached to the articles of the corresponding substances.

Fig. 7. A decision memo created for the exemplary substance Kryptonite.

Size and Current Status Currently, KnowSEC provides explicit decision modules for supporting the assessment of the relevance, the persistence in the environment, the bioaccumulation potential, and the toxicity of a given substance. The taxonomy of decisions however, contains 15 different main decisions on substances having a larger number of sub-decisions.

The static part of the knowledge base currently consists of 282 questions (user inputs to characterize the investigated substance) grouped by 92 questionnaires, 558 decisions (assessments of the investigated substance), and about 1,000 rules to derive the decisions. The rules are automatically generated from entered decision tables that allow for an intuitive and maintainable knowledge development process. Two knowledge engineers are supporting a team of domain specialists, that partly define the knowledge base themselves, partly giving domain knowledge to the knowledge engineers.

At the beginning of the project a couple of internal data bases were integrated into KnowSEC as (decision) memos. Currently, the system contains more than 27,000 (decision) memos for the 11,000 substances. In the form dialog more than 51,000 questions were answered; partially automatically by imports of internal data bases. Both, decision memos and the explicit rule base derived more than 42,000 module decisions.

5 Conclusions

Advanced decision support systems allow for the distributed and episodic handling of complex decision problems. They implement large knowledge spaces by mixing different knowledge representations with informal decision justifications. In this paper, we introduced a novel approach for building decision making systems, that support collaborative and episodic decision making. Furthermore, we motivated how the application of the knowledge formalization continuum helps to create knowledge in complex domains. The practical applicability and relevance of the presented approach was demonstrated by the discussion of an installed decision support system for the assessment of chemical substances. When decisions are derived in a collaborative and episodic setting, the transparency of found decisions is of prime importance. Thus, we are currently working on an elaborated explanation approach based on the provenance ontology PROV, that is capable to provide intuitive and effective ad-hoc explanations even for end users.

References

1. Baumeister, J., Reutelshoefer, J., Belli, V., Striffler, A., Hatko, R., Friedrich, M.: KnowWE - a wiki for knowledge base development. In: The 8th Workshop on Knowledge Engineering and Software Engineering (KESE2012). http://ceur-ws.org/Vol-949/kese8-05_04.pdf (2012)
2. Baumeister, J., Reutelshoefer, J., Puppe, F.: Engineering intelligent systems on the knowledge formalization continuum. International Journal of Applied Mathematics and Computer Science (AMCS) 21(1) (2011), http://ki.informatik.uni-wuerzburg.de/papers/baumeister/2011/2011-Baumeister-KFC-AMCS.pdf
3. Baumeister, J., Reutelshoefer, J., Puppe, F.: KnowWE: A semantic wiki for knowledge engineering. Applied Intelligence 35(3), 323–344 (2011), http://dx.doi.org/10.1007/s10489-010-0224-5
4. Franconi, E., Meyer, T., Varzinczak, I.: Semantic diff as the basis for knowledge base versioning. In: 13th International Workshop on Non-Monotonic Reasoning (NMR). pp. 7–14 (2010)
5. 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)
6. Mersmann, S., Dojat, M.: SmartCare^{tm} - automated clinical guidelines in critical care. In: ECAI'04/PAIS'04: Proceedings of the 16th European Conference on Artificial Intelligence, including Prestigious Applications of Intelligent Systems. pp. 745–749. IOS Press, Valencia, Spain (2004)
7. Miller, R.A., Pople, H.E., Myers, J.: INTERNIST-1, an Experimental Computer-Based Diagnostic Consultant for General Internal Medicine. New England Journal of Medicine 307, 468–476 (1982)
8. Milne, R., Nicol, C.: TIGER: Continuous diagnosis of gas turbines. In: ECAI'00: Proceedings of the 14th European Conference on Artificial Intelligence. Berlin, Germany (2000)
9. Noy, N.F., Chugh, A., Liu, W., Musen, M.A.: A framework for ontology evolution in collaborative environments. In: ISWC'06: Proceedings of the 5th International Semantic Web Conference, LNAI 4273. pp. 544–558 (2006), http://dx.doi.org/10.1007/11926078_39
10. Noy, N.F., Musen, M.A.: PromptDiff: a fixed-point algorithm for comparing ontology versions. In: In 18th National Conference On Artificial Intelligence (AAAI-2002. pp. 744–750 (2002)
11. Puppe, F.: Knowledge Reuse among Diagnostic Problem-Solving Methods in the Shell-Kit D3. International Journal of Human-Computer Studies 49, 627–649 (1998)
12. Puppe, F., Atzmueller, M., Buscher, G., Hüttig, M., Luehrs, H., Buscher, H.P.: Application and evaluation of a medical knowledge system in sonography (SONOCONSULT). In: ECAI'08/PAIS'08: Proceedings of the 18th European Conference on Artificial Intelligence, including Prestigious Applications of Intelligent Systems. pp. 683–687. IOS Press, Amsterdam, The Netherlands, The Netherlands (2008)
13. Schaffert, S., Bry, F., Baumeister, J., Kiesel, M.: Semantic wikis. IEEE Software 25(4), 8–11 (2008)
14. Tudorache, T., Nyulas, C., Noy, N.F., Musen, M.A.: WebProtégé: A collaborative ontology editor and knowledge acquisition tool for the web. Semantic Web (2012)
15. W3C: PROV-O: The PROV Ontology: http://www.w3.org/tr/prov-o/ (April 2013)
16. W3C: RIF-Core Recommendation: http://www.w3.org/tr/rif-core/ (February 2013)
17. W3C: SPARQL 1.1 recommendation: http://www.w3.org/tr/sparql11-query/ (March 2013)

Identifying Guidelines for Designing and Engineering Human-Centered Context-Aware Systems *(Position paper)*

Emilian Pascalau

Conservatoire National des Arts et Métiers,
2 rue Conté,75003 Paris, France
emilian.pascalau@cnam.fr

Abstract. In the "future internet" environment that is generative and fosters innovation, applications are simplifying, are becoming mobile, are getting more social and user oriented. Software design capable of coping with such a generative environment that drives and supports innovation is still an issue. Some of the challenges of such systems include: empowering end-users with the necessary tools to model and develop applications by themselves, while in the same time hiding the technical layer from them. This paper introduces a set of guidelines for designing and engineering human-centered context-aware systems from a human computer interaction and meta-design perspective.

1 Introduction

Recent years have brought rapid technological advances both hardware and software: increasing pervasive computing paradigm, embedded sensor technologies and wide range of wireless and wired protocols. Applications are simplifying, are becoming mobile, are moving to the cloud, are getting more social and user focused [14]. "Future Internet" is emerging as a new environment, an environment that is generative, that fosters innovation, through the advances of technologies and a shift in people's perception about it and a paradigm shift in how people act and react in this environment [22].

In this context two directions get highlighted: context together with context -awareness and human-centered computing. Studied for more than 30 years in the field of artificial intelligence, computer and cognitive science, context has still been identified by Gartner, alongside cloud computing, business impact of social computing, and pattern based strategy, as being one of the broad trends that will change IT and the economy in the next 10 years [21].

We observe a paradigm shift in terms of users of context-aware systems. For example a user is no longer a specific individual or organization. It is often a community of collaborating, or performing similar tasks groups of users. Therefore, there is a growing need for systems meeting expectations of massive distributed

user base of pervasive ubiquitous devices as well as distributed cloud-based web services.

Design and deployment of such software capable of coping with a generative environment that drives and supports innovation through direct interaction and empowerment of the end-user is still an issue. Not only the development of such systems should be agile - the boundary is pushed even further - we need systems that are designed to be agile on run-time. It has been already argued (see for instance [15]) that designers and programmers can not foresee and anticipate what end-users will need. Users know better what they need and future internet environment clearly underlines this fact.

In consequence some of the challenges of such systems that arise in the future internet environment include: empowering end-users with the necessary tools to model and develop applications by themselves, while in the same time hiding the technical layer from them. This paper is part of a work in progress, and identifies and introduces a set of guidelines for designing and engineering human-centered context-aware systems.

The rest of the paper is organized as follows: Section 2 discusses a use case that is based on a concrete end-user problem (managing and tracking online purchases) arising from the future internet environment; the use case will support us in identifying a set of guidelines, for designing and engineering human-centered context-aware systems, in Section 3; Section 4 is reserved for related work. We conclude in section 5.

2 Application Scenario - Slice.com

In the future internet environment email communication is part of daily activities. Many of the business processes that take place in a company and not only are started and / or integrate the actions of receiving / sending emails. In some situations entire processes are comprised in email communications. This type of email based use cases are important both for academia as well as for industry. For academia from a perspective oriented towards methodology, for industry from a practical perspective, addressing very specific problems. Emails contain knowledge that is bundled in an unstructured way but which has meaning to end-users, and which could be used to address end-user specific problems.

A human-centered context-aware system dealing with such a use case would be required to provide at least the following capabilities:

- provide the means and allow the end-user to model and represent context;
- allow the modeling of relationships between context elements;
- allow the modeling of interactions between different contexts, this implies both in the form of conditions and sequences of events and actions (more precise business rules and business processes)
- based on the provided models have the capabilities to discover in the environment the modeled context(s)
- sense events and actions that are performed in the system
- perform actions according to models defined.

We support our previous assertions by discussing the Slice application scenario, in the next paragraphs.

Commius[1] is a European research project that tackles systems for small and medium enterprise systems (SMEs). The final goal of Commius, as argued in [5], is to turn existing email-systems into a management framework for structured processes. Each incoming email is autonomously matched to a process that is enhanced with proactive annotations.

Slice.com is an industry project. Similar to Commius uses emails to tackle a very specific end-user related problem, keeping track of online purchases, that emerged from the future internet dynamic and generative environment. This project it is even more specific from the perspective of an end-user.

Slice is an online purchase management tool that gets hooked into your email account. Whenever a new email is received Slice automatically analyzes the content of the email. If the email contains order information from one of your online shops, then Slice via pattern-based recognition techniques extracts order related contextual information and organizes this information for you. Hence all your purchases will be gathered in one place, you will be able to keep track of your shopping history, amount of money that you spent, type of products, time related information i.e. when a shipment is about to arrive and so forth.

We analyze from an end-user perspective what this use case is about.

- **Problem faced by users:** keeping track of the purchases made online.
- **Applications involved, Services**: Email as a legacy system; Services: online shops (Amazon, EBay), shipment services (FedEx, UPS); Geolocation services (Google Maps); other type of services i.e. topic extraction
- **Concepts**: shop, service, user, invoice, email, time and date, amount of money, product, type of product, location, address, tracking number. The list of concepts is not exhaustive, and is up to the each user; however these are the most obvious ones. Concepts are all those entities that are used in taking decisions and / or involved in any way in the process of resolving the end-user's problem.
- **Context**: For example one context, from the perspective of an end-user in the Slice use case, could comprise: a specific service such as FedEx; concepts associated with it, i.e. shipment, location, address. Further more interaction related to this specific context could be provided, as what to do with this information and so forth.

Figure 1 depicts a general interaction process with respect to this use case.

3 Identifying Guidelines

Fisher and Giaccardi argue in [11] that in a world that is not predictable, improvisation, evolution and innovation are a necessity. There is a shift from processes towards users. Users want their problems and their requirements to be taken into

[1] `http://www.commius.eu/`

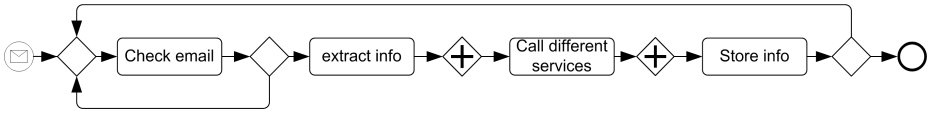


Fig. 1. General interaction process to solve the problem of keeping track of online purchases

account; they want to be part of the conversation. Continuously changing business models, do not fit anymore the old and stiff approaches. Processes must be in accordance with the reality. For instance process mining techniques [20] that look at event logs emphasize the fact that processes which actually get executed are different compared to the original blueprints. Companies need to change to what customers/users actually do.

Context greatly influences the way humans or machines act, the way they report themselves to situations and things; furthermore any change in context, causes a transformation in the experience that is going to be lived, sensed [4]. Many psychological studies have shown that when humans act, and especially when humans interact, they consciously and unconsciously attend to context of many types as stated in [12].

Traditionally context has been perceived in computer science community as a matter of location and identity, see for instance [9]. However interaction and problems concerning interaction require more than just the environmental context (location, identity) used traditionally in context-aware systems [12]. Lately the notion of context has been considered not simply as state but as part of a process in which users are to be involved [7].

Nardi underlines this aspect clearly in [15] stating that *"we have only scratched the surface of what would be possible if end users could freely program their own applications... As has been shown time and again, no matter how much designers and programmers try to anticipate and provide for what users will need, the effort always falls short because it is impossible to know in advance what may be needed... End users should have the ability to create customizations, extensions and applications..."*.

From a human-centered computing perspective this type of system is what Donald Norman calls in [16] the type of system, where the system itself disappears from sight, and humans (end-users) can once again concentrate upon their activities and their goals.

Grundin [12] continues and argues that aggregation or interpretation done by software systems are different than aggregation and interpretation done by biological, psychological and social processes.

Meta-design is a conceptual framework defining and creating social and technical infrastructures in which new forms of collaborative design can take place [11]. Meta-design originates in human computer interaction field and tackles end-user development.

Table 1, introduced in [11] compares side by side traditional design vs. meta-design. However, in our perspective a human-centered context-aware system, in order to provide a high degree of generality and to avoid re-implementation of common things related to infrastructure, should be a layered system as discussed in [18]. A low level that is very technical and should be hidden from the end-user. This low level would follow to great extent traditional design. The high level on the other hand should follow mainly meta-design. A translation mechanism has to be put into place to assure translation between these two layers.

Table 1. Traditional Design vs. Meta-Design [11]

Traditional Design	**Meta-Design**
guidelines and rules	exceptions and negations
representation	construction
content	context
object	process
perspective	immersion
certainty	contingency
planning	emergence
top-down	bottom-up
complete system	seeding
autonomous creation	co-creation
autonomous mind	distributed mind
specific solutions	solutions spaces
design-as-instrumental	design-as-adaptive
accountability, know-what (rational decisioning)	affective model, know-how (embodied interactionism)

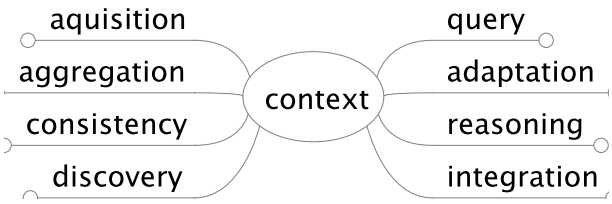


Fig. 2. Context requirements

Several definitions for the concept context have been enunciated; Fischer, however gives a definition that takes into account the human-centered computational environments. He defines context in [10] as being *the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way to the 'right' person.*

Figure 2 depicts aspects that have been identified in [1] as requirements for dealing with context.

Table 2, introduced in [10] depicts adaptive and adaptable systems. Context aware systems traditionally position themselves, according to Table 2 in the category of adaptive systems. These systems employ users' profiles information and other type of contextual information, like location to improve users' experience. These approaches, although they provide a degree of flexibility, are however still stiff approaches because they are still based on predefined designs and very specific.

Table 2. Adaptive vs. Adaptable Systems [10]

	Adaptive	**Adaptable**
definition	dynamic adaptation by the system itself to current task and current user	users change the functionality of the system
knowledge	contained in the system; projected in different ways	knowledge is extended by users
strengths	little (or no) effort by users; no special knowledge of users is required	users are in control; users know their tasks best
weaknesses	users often have difficulties developing a coherent model of the system; loss of control	users must do substantial work; complexity is increased (users need to learn adaptation components); systems may become incompatible
mechanisms required	models of users, tasks, dialogs; incremental update of models	support for end-user modifiability and development
application domains	active help, critiquing systems, recommender systems	end-user modifibiality, tailorability, design in use, meta-design

In our vision a human-centered context-aware system is a system where adaptivity and adaptability are blended together. By such a method users will be able to directly model how the context should look like for a particular problem, and afterwards the system would be required only to verify that the specified context really exists in a given environment. Moreover while for adaptive systems as stated in [3] the architecture of such systems comprises a user model (user perspective on the problem) and a domain model (system perspective as it has been designed), for a human-centered context-aware system there should be only one conceptual model of the problem, that should be shared and understood in the same way both by the end-user and the system.

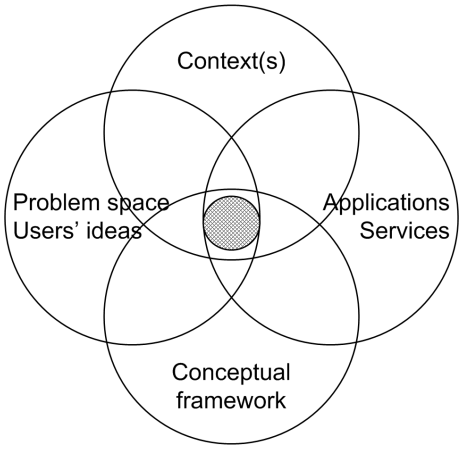


Fig. 3. Interrelated topics

Figure 3 depicts the main perspectives that need to be blended together in order to design a human-centered context-aware system. This particular view has its roots in the meta-design conceptual framework [11].

Aspect Oriented Programming [19] is a relatively new methodology for software development that aims at providing software modularity by means of separation of cross cutting concerns. This is an approach for requirements engineering that focuses on customers concerns to be made consistent with aspect oriented software development. In terms of software engineering throughout the code there are specifically designed points that support adaptation based on defined aspects.

The authors in [8] discuss how aspect orientation and can be used in context aware systems design. Furthermore because in aspect oriented programming direct user input is taken into account this is an example of human-centered context aware systems. However this approach although it goes into the right direction it is restricted by the predefined points that support aspect orientation in the implemented code.

Based on the analysis made the design and engineering process of human-centered context-aware systems should follow the following guidelines:

1. such a system should be as general as possible and should be able to tackle as many problems as possible [11];
2. such a system should should provide the right level of representation such that a problem representation could be automatically translated into the core constructs of the underlying programming language in which the overall system is implemented;
3. such systems should not be domain specific and therefor closed system, but an open system that could be used to address a wide range of problems and applications;

4. in such a system the focus should be on the needs of the end-user and not on the system itself; the system should be hidden from the end-user;
5. such a system should be designed for evolution and should provide the means to evolve through the input of users, as well as by itself;
6. such a system should support both skilled domain workers as well as novice users;
7. such a system should be a co-adaptive environment where users change because they learn and systems change because users become co-developers;
8. such a system should allow active participation and empowerment of end-users;
9. in such a system the end-user should be the coordinator of how the system works.

Some of the guidelines refer to the relationship between system and the end-user, and some concern just the system. The system that we envision is similar to an operating system in terms of being general and not domain specific. The system will be an intelligent one and will accept problem descriptions given by the end-users. These descriptions will act as application models. Such a description together with the intelligent system will make the application. We have already made initial steps towards implementing such a system in [18], [17].

4 Related Work

Context awareness has been studied for several decades from many perspectives. Development of context-aware application, however, is still very difficult and time consuming and aside location-based services, not too many applications have been put into real use.

A series of surveys addressing different aspects (i.e. context modeling and context-based reasoning, context-aware frameworks and applications, context-aware web services) of development of context-aware systems and their applications have been developed. We observe that the present approaches to context reasoning and context-aware system design are only partial, in most of the cases being too specific, or too generic.

Types of context used and models of context information systems that support collecting and disseminating context, and applications that adapt to the changing context have been addressed by Chen and Kotz in [6].

Bolchini et al. discuss in [4] general analysis framework for context models and a comparison of some of the data-oriented approaches available in the literature. The framework addresses: modeled aspects of context (space, time, absolute/relative, history, subject, user profile), representation features (type of formalism, level of formality, flexibility, granularity, constraints) and context management and usage (construction, reasoning, information quality monitoring, ambiguity, automatic learning features, multi-context modeling).

A unified architectural model and a new taxonomy for context data distribution has been discussed in [2]. Authors identify 3 major aspects: (1) context

data distribution should take into account node requests and quality of context requirements to reduce management overhead; (2) context data distribution requires adaptive and crosscutting solutions able to orchestrate the principal internal facilities according to specific management goals; (3) informed context data distribution can benefit from their increased context-awareness to further enhance system scalability and reliability.

The impact of context-awareness on service engineering has also been noticed. A classic and relatively recent survey [13] by Kapitsaki et al. considers context as constituting an essential part of service behavior, especially with the interaction of users. They observe that "at the heart of every context-aware service, relevant business logic has to be executed and (. . .) adapted to context changes".

Related work concerning this human-centered context-aware perspective, as it was analyzed in this paper is to the best of our knowledge only in an early stage.

5 Conclusions

In this paper we have started an initial discussion about the design and engineering of human-centered context-aware systems. Aspects discussed in this paper are part of a work in progress. Our previous experience [18], [17] with developing human-centered context-aware systems proved to be not trivial. This discussion comprised aspects from human computer interaction, meta-design, context and context-awareness. We have emphasized the fact that systems predesigned can not foresee all aspects and facets of a problem. Therefore end-user should be given the necessary tools to design and develop their own solutions based on existing services. We provide a set of guidelines and properties that should characterize such human-centered context-aware systems.

Next steps include formalizing a conceptual framework, methodology and implementation guidelines for developing such a system that is capable of tackling in a unified way the problem of development of human-centered context-aware applications.

References

1. Christos B. Anagnostopoulos, Athanasios Tsounis, and Stathes Hadjiefthymiades. Context awareness in mobile computing environments. *Wireless Personal Communications*, 42(3):445–464, 2007.
2. Paolo Bellavista, Antonio Corradi, Mario Fanelli, and Luca Foschini. A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys (CSUR)*, 44(4):50 pages, 2012.
3. David Benyon and Dianne Murray. Applying user modeling to human-computer interaction design. *Artificial Intelligence Review*, 7(3-4):199–225, 1993.
4. Cristiana Bolchini, Carlo A. Curino, Elisa Quintarelli, Fabio A. Schreiber, and Letizia Tanca. A data-oriented survey of context models. *ACM SIGMOD Record*, 36(4):19–26, 2007.

5. Thomas Burkhart, Dirk Werth, and Peter Loos. Context-sensitive business process support based on emails. In *WWW 2012 – EMAIL'12 Workshop*, April 2012.
6. Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Dartmouth College Hanover, NH, USA, 2000.
7. Joëlle Coutaz, James L. Crowley, Simon Dobson, and David Garlan. Context is key. *Communications of the ACM - The disappearing computer*, 48(3):49–53, 2005.
8. Abhay Daftari, Nehal Mehta, and Shubhanan Bakre Xian-He Sun. On design framework of context aware embedded systems. In *Monterey Workshop on Software Engineering for Embedded Systems: From Requirements to Implementation*, 2003.
9. Anind K. Dey and Gregory D. Abowd. Towards a better understanding of context and context-awareness. In *In HUC '99: Proceedings of the 1st international symposium on Handheld and Ubiquitous Computing*, pages 304–307. Springer-Verlag, 1999.
10. Gerhard Fischer. Context-aware systems - the 'right' information, at the 'right' time, in the 'right' place, in the 'right' way, to the 'right' person. In *AVI'12*. ACM, 2012.
11. Gerhard Fischer and Elisa Giaccardi. *End User Development - Empowering People to Flexibly Employ Advanced Information and Communication Technology*, chapter Meta-Design: A Framework fo the Future of the End-User Development. Kluwer Academic Publishers, 2004.
12. Jonathan Grudin. Desituating action: digital representation of context. *Human-Computer Interaction*, 16(2):269–286, 2001.
13. Georgia M. Kapitsaki, George N. Prezerakos, Nikolaos D. Tselikas, and Iakovos S. Venieris. Context-aware service engineering: A survey. *The Journal of Systems and Software*, 82(8):1285–1297, 2009.
14. Charles McLellan, Teena Hammond, Larry Dignan, Jason Hiner, Jody Gilbert, Steve Ranger, Patrick Gray, Kevin Kwang, and Spandas Lui. *The Evolution of Enterprise Software*. ZDNet and TechRepublic, 2013.
15. Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, 1993.
16. Donald A. Norman. *The Invisible Computer*. MIT Press, 1999.
17. Emilian Pascalau. Mashups: Behavior in context(s). In *Proceedings of 7th Workshop on Knowledge Engineering and Software Engineering (KESE7) at the 14th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2011)*, volume 805, pages 29–39. CEUR-WS, 2011.
18. Emilian Pascalau. Towards TomTom like systems for the web: a novel architecture for browser-based mashups. In *Proceedings of the 2nd International Workshop on Business intelligencE and the WEB (BEWEB11)*, pages 44–47. ACM New York, NY, USA, 2011.
19. Ian Sommerville. *Software Engineering 8*. Addison Wesley, 2007.
20. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A. J. M. M. Weijters. Workflow mining: a survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.
21. Min Wang. Context-aware analytics: from applications to a system framework. http://e-research.csm.vu.edu.au/files/apweb2012/download/APWeb-Keynote-Min.pdf, 2012.
22. Jonathan Zittrain. *The Future of the Internet And How to Stop It*. Yale University Press New Haven and London, 2008.

Overview of Recommendation Techniques in Business Process Modeling*

Krzysztof Kluza, Mateusz Baran, Szymon Bobek, Grzegorz J. Nalepa

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

Abstract Modeling business processes is an important issue in Business Process Management. As model repositories often contain similar or related models, they should be used when modeling new processes. The goal of this paper is to provide an overview of recommendation possibilities for business process models. We introduce a categorization and give examples of recommendation approaches. For these approaches, we present several machine learning methods which can be used for recommending features of business process models.

1 Introduction

Business Process (BP) models are visual representations of processes in an organization. Such models can help to manage process complexity and are also easy to understand for non-business user. Although there are many new tools and methodologies which support process modeling, especially using Business Process Model and Notation (BPMN) [1], they do not support recommendation mechanisms for BP modelers.

As BPMN specifies only a notation, there can be several ways of using it. There are style directions how to model BPs [2], or guidelines for analysts based on BPs understandability (e.g. [3]). However, a proper business process modeling is still a challenging task, especially for inexperienced users.

Recommendation methods in BP modeling can address this problem. Based on current progress or additional pieces of information, various features can be recommended to a modeler, and he/she can be assisted during designing models. Such assistance can provide autocompleting mechanisms with capabilities of choosing next process fragments from suggested ones. Names of model elements or attachments can be recommended as well. Such approaches can reduce number of errors during process design as well as speed up modeling process. It also supports reusing of existing process models, especially when a process repository is provided.

The rest of this paper is organized as follows: In Section 2, we provide a categorization of recommendation methods used in business process modeling. Section 3 describes the current state of the art in this research area. Selected machine learning methods that can be used for recommending features of process models are presented in Section 4. Section 5 presents an example which can be considered as a suitable case study for recommendation purposes. The paper is summarized in Section 6.

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

2 Types of recommendations

Basically, recommendation methods in BPs modeling can be classified as one of two types: *subject-based* and *position-based* classification. The first one concentrates on what is actually suggested, while the second one focuses on the place where the suggestion is to be placed. However, they are suited for different purposes and therefore are complementary. A hierarchy of the identified types of recommendation methods is presented in Figure 1.

2.1 Subject-based classification

In subject-based classification we focus on what is actually suggested. The suggestion itself is not directly dependent on the context it is placed in. The recommendation algorithms may actually inspect the context to be able to deliver more accurate results but it is not an inherent feature of recommended item.

1. **Attachment recommendations** – as the name suggests, these recommendations suggest how to link a business process (or, more precisely, a selected element of it) with an external entity like a decision table or another process. Attachment recommendations appear naturally where user should link two already existing items.
   (a) **Decision tables** – recommendations for a decision table describing conditions in a gate. See an example in Figure 2.

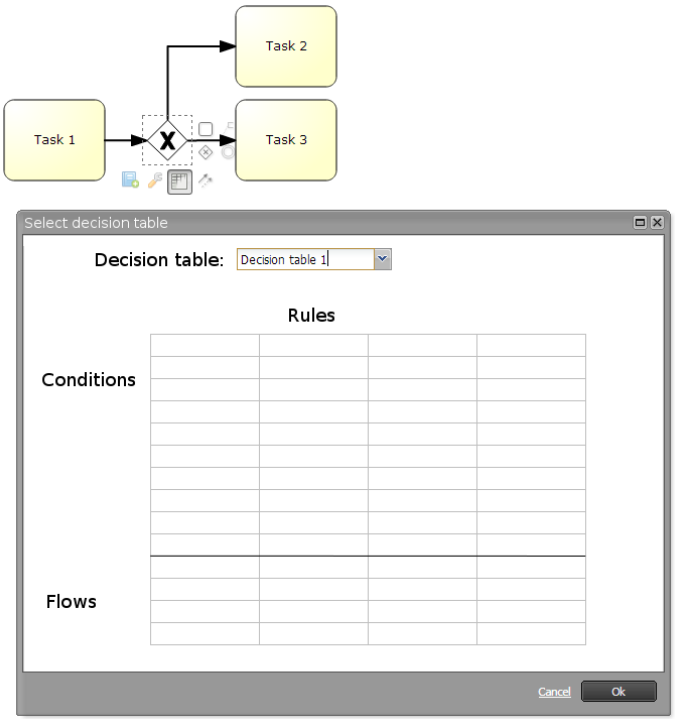


Figure 2. Decision table suggestion

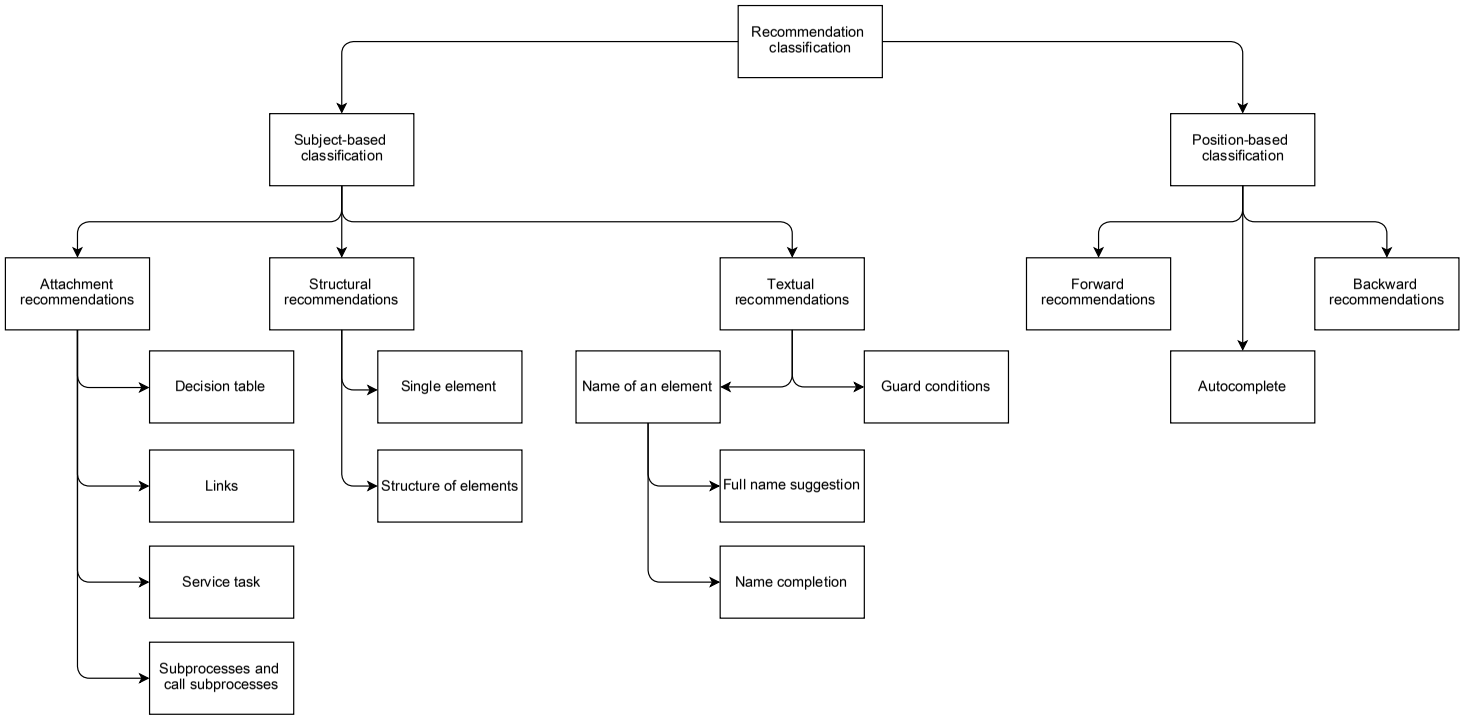


Figure 1. Identified types of recommendations

(b) **Links** – recommendations for a catching event that should be connected with the selected throwing Intermediate Link Event. See an example on Figure 3.

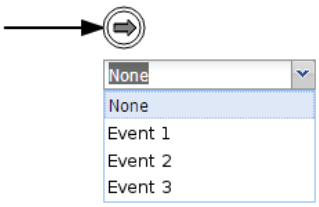


Figure 3. Throwing Intermediate Link Event suggestion

(c) **Service task** – recommendation for a service task performed in the given task item. See an example in Figure 4.

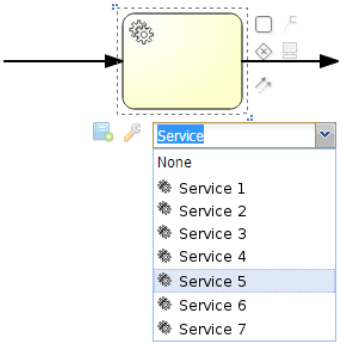


Figure 4. Service task selection with recommendation

(d) **Subprocess and call subprocess** – recommendation for a subprocess or call subprocess that should be linked with the given activity (see Figure 5).

2. **Structural recommendations** – a new part of the diagram is suggested. One or more elements with, for example, missing incoming or outgoing flows are selected. The suggested structure is connected with old chosen elements.
   (a) **Single element** – a single item (activity, gate, swimlane, artifact, data object or event) is suggested. This is a more straightforward extension of editors like Oryx/Signavio that can already insert single elements quite easily.
   (b) **Structure of elements** – two or more items are suggested. A more sophisticated solution where an entire part of the process is inserted into existing, unfinished structure.
3. **Textual recommendations** are suggestions of names of elements or guard conditions. Either the full text can be suggested or suggestions may show while the text is being typed.
   (a) **Name of an element** – a name of activity, swimlane or event may be suggested.
      i. **Name completion** happens when user is typing the name. Several possible completions of partially entered name are suggested to the user.

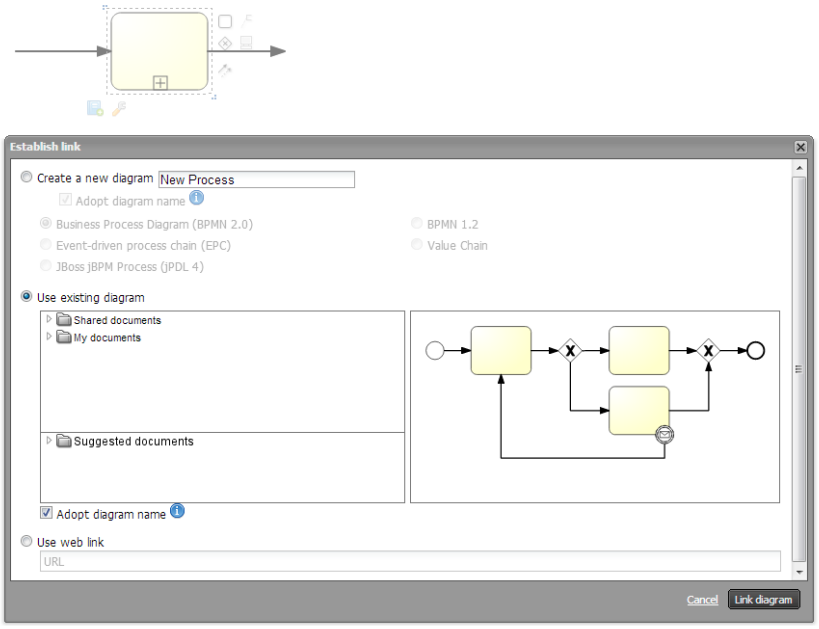


Figure 5. Subprocess selection with recommendation

ii. **Full name suggestion** happens when the user wants the name to be suggested by the system based on the context in which the element is placed.

(b) **Guard condition suggestions** are different from name suggestions because more than one text (condition) may be suggested at once and these conditions must satisfy the requirements of the gateway. The latter requirement implies that semantic analysis of conditions is necessary to give meaningful suggestions. See example in Figure 6.

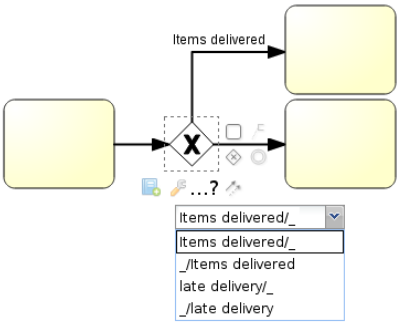


Figure 6. Guard condition suggestion

2.2 Position-based classification

1. **Forward completion** – a part of the process is known and the rest of the process, starting with one selected activity, is to be suggested. See Figure 7.

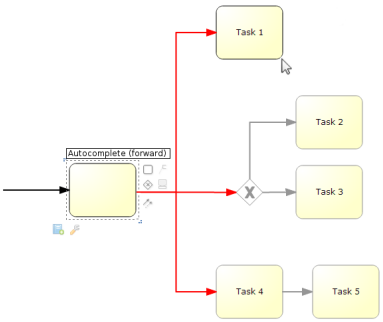


Figure 7. Forward completion

2. **Backward completion** – a part of the process is known and the rest of the process, ending with one selected activity, is to be suggested. See Figure 8.

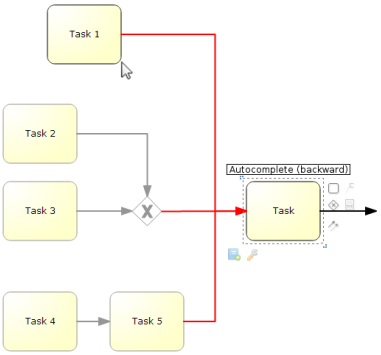


Figure 8. Backward completion

3. **Autocomplete** – a part of the process is known and the rest of the process is to be suggested. A number of items with no outgoing or incoming flows is selected – missing flows will lead to or from the suggested structure. See Figure 9.

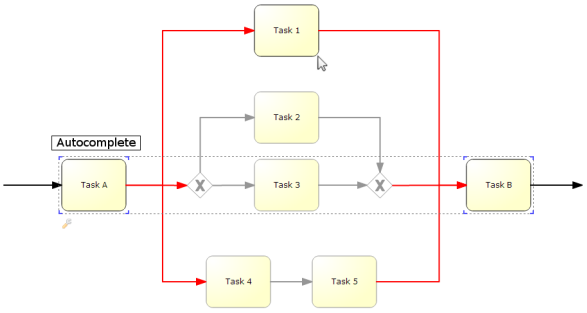


Figure 9. Autocompletion

3 Recommendation Techniques for Business Process Models

Empirical studies have proven that modelers prefered to receive and use recommendation suggestions during design [4]. Recommendations can be based on many factors, including labels of elements, current progress of modeling process, or some additional pieces of information, such as process description. There are several existing approaches which can be assigned to the following subject-based categories:

1. **Attachment recommendations**: Born et al. [5] presented an approach that supports modelers during modeling tasks by finding appropriate services, meaningful to the modeler. More complex approach which helps process designers facilitate modeling by providing them a list of related services to the current designed model was proposed by Nguyen et al. [6]. They capture the requested service's composition context specified by the process fragment and recommend the services that best match the given context. The authors also described an architecture of a recommender system which bases on historical usage data for web service discovery [7].
2. **Structural recommendations**: Mazanek et al. [8] proposed a syntax-based assistance in diagram editor which takes advantage of graph grammars for process models. Based on this research they proposed also a sketch-based diagram editor with user assistance based on graph transformation and graph drawing techniques [9].

   Hornung et al. [10] presented the idea of interpreting process descriptions as tags and based on them provide a search interface to process models stored in a repository. Koschmider and Oberweis extended this idea in [11] and presented their recommendation-based editor for business process modeling in [4]. The editor assists users by providing search functionality via a query interface for business process models or process model parts and using automatic tagging mechanism in order to unveil the modeling intention of a user at process modeling time. An approach proposed by Wieloch et al. [12] delivers a list of suggestions for possible successor tasks or process fragments based on analysis of context and annotations of process tasks. Case based reasoning for workflow adaptation was discussed in [13]. It allows for structural adaptations of workflow instances at build time or at run time. The approach supports the designer in performing such adaptations by an automated method based on the adaptation episodes from the past. The recorded changes can be automatically transferred to a new workflow that is in a similar situation of change.
3. **Textual recommendations**: Naming strategies for individual model fragments and whole process models was investigated in [14] They proposed an automatic naming approach that builds on the linguistic analysis of process models from industry. This allows for refactoring of activity labels in business process models [15].

   According to Kopp et al. [16] it is not to automatically deduct concrete conditions on the sequence flows going out from the new root activity as we cannot guess the intention of the fragment designer. However, they presented how a single BPMN fragment can be completed to a BPMN process using autocompletion of model fragments, where the types of the joins are AND, OR, and XOR.

4 Machine Learning Approach for Recommendation

The idea of recommender systems was evolving along with a rapid evolution of the Internet in mid-nineties. Methods such as collaborative filtering, content-based and knowledge-based recommendation [17] gained huge popularity in the area of web services [18] and recently most often in context-aware systems [19]. The principal rule that most of the recommendation methods are based on, exploits an idea of similarities measures. This measures can be easily applied to items that features can be extracted (eg. book genre, price, author) and ranked according to some metrics (customer liked the book or not). However, when applied to BPMN diagrams, common recommender systems face a big problem of non existence of standard metrics that will allow for comparison of models. What is more, feature extraction of the BPMN diagrams that will allow for precise and unambiguous description of models is very challenging and, to our knowledge, still unsolved issue.

Therefore, other machine learning methods should be investigated according to an objective aiming at providing recommendation mechanisms for a designer. The following Section contains an analysis of possible application of machine learning methods to recommendations described in Section 2. A comprehensive summary is also provided in Table 1. The black circle denotes full support of particular machine learning method to recommendation; half-circle denoted partial support of particular machine learning method to recommendation, and empty circle means no, or very limited support.

	Clustering algorithms[a]	**Decision trees**[b]	**Bayesian networks**[c]	**Markov chains**
Attachment recommendations	❍	●	❍	❍
Structural recommendations	❍	●	●	●
Textual recommendations	❍	❍	◗	●
Position based classification	❍	●	●	●

Table 1. Comparison of different machine learning methods for recommending features denoted in Section 2

[a] Useless as an individual recommendation mechanism, but can boost recommendation when combined with other methods

[b] No cycles in diagram

[c] No cycles in diagram

4.1 Classification

Clustering methods Clustering methods [20] are based on optimization task that can be described as an minimization of a cost function that are given by the equation 1. K denotes number of clusters that the data set should be divided into.

$$\sum_{n=1}^{N}\sum_{k=1}^{K}\|X_n - \mu_n\|^2 \tag{1}$$

This cost function assume existence of a function f that allows for mapping element's features into an M dimensional space of $X \in R^m$. This however requires developing methods for feature extraction from BPMN diagrams, which is not trivial and still

unsolved task. Nevertheless, clustering methods can not be used directly for recommendation, but can be very useful with combination with other methods.

Decision trees Decision trees [21] provide a powerful classification tool that exploits the tree data structure to represent data. The most common approach for building a tree, assumes possibility of calculation entropy (or based on it, so-called *information gain*) that is given by the equation 2.

$$E(X) = -\sum_{i=1}^{n} p(x_i) log(p(x_i))) \tag{2}$$

To calculate the entropy, and thus to build a decision tree, only a probability p of presence of some features in a given element is required. For the BPMN diagram, those features could be diagram nodes (gateways, tasks, etc) represented by a distinct real numbers. Having a great number of learning examples (diagrams previously build by the user), it is possible to build a tree that can be used for predicting next possible element in the BPMN diagram. However, the nature of the tree structure requires from BPMN diagram to not have cycles, which not always can be guaranteed.

4.2 Probabilistic Graphical Models

Probabilistic Graphical Models use a graph-based representation as the basis for compactly encoding a complex probability distribution over a high dimensional space [22]. The most important advantage of probabilistic graphical models over methods described in Section 4.1 is that it is possible to directly exploit the graphical representation of BP diagrams, which can be almost immediately translated into such model.

Bayesian networks Bayesian network (BN) [23] is an acyclic graph that represents dependencies between random variables, and provide graphical representation of the probabilistic model. The example of a Bayesian network is presented in Figure 10.

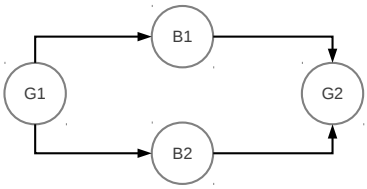


Figure 10. Bayesian network

The advantage of BN is that the output of a recommendation is a set of probabilities, allowing for ranking the suggestion from the most probable to the least probable. For example to calculate the probability of the value of the random variable *B1* from the Figure 10, the equation 3 can be used. The G1,2 can be denoted as BPMN gateways, and B1,2 as other blocks, e.g. Tasks or Events. Thus, having any of these blocks given, we can calculate a probability of a particular block being a missing part.

$$P(B1) = \sum_{G1} \sum_{G2} \sum_{B2} P(G1)P(B1|G1)P(B2|G1)P(G2|B1, B2) \tag{3}$$

This method however, will not be efficient for large diagrams, since exact inference in Bayesian networks is NP-hard problem. To solve this problem either the small chunks of BPMN diagram can be selected for the inference, or approximate inference applied.

Markov Chains Markov chain [24] is defined in terms of graph of state space $Val(X)$ and a transition model τ that defines, for every state $x \in Val(X)$ a next-state distribution over $Val(X)$. These models are widely used for text auto-completion and text correction, but can be easily extended to cope with other problems such as Structural recommendations, or position-based classification.

We can assume different BPMN block types as states x , and connections between them as a transition model τ.

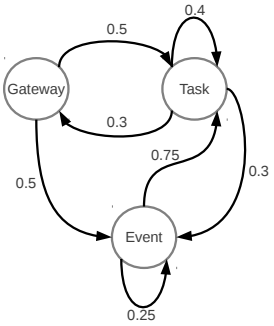


Figure 11. Markov Chain

An example Markov chain model is presented in Figure 11. The number above the arrows denotes transition probability from one state to another. The characteristic of the Markov chains allows for cycles.

5 Case Study

The presented recommendation approaches can be applied to process models, especially modeled on the basis of the existing processes in a model repository. For the purpose of evaluation, we prepared 3 different BPMN models of bug tracking systems (Django and JIRA) and the model of the issue tracking approach in VersionOne. A bug tracking system is a software application that helps in tracking and documenting the reported software bugs (or other software issues in a more general case). Such a system is often integrated with other software project management applications.

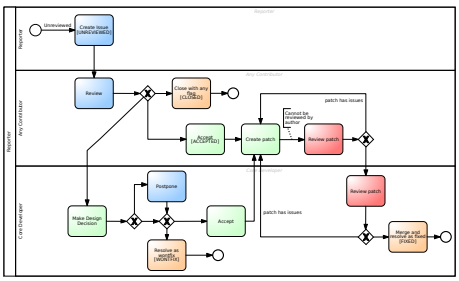


Figure 12. Django

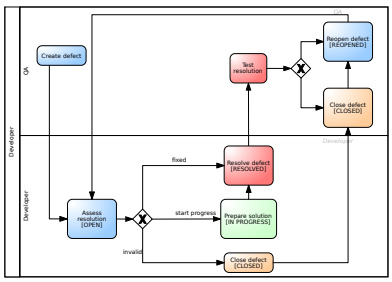


Figure 13. Jira

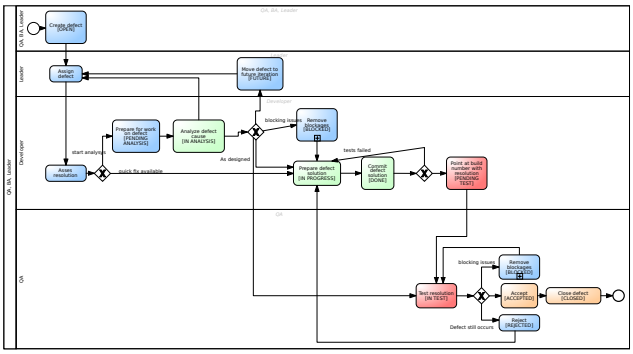


Figure 14. VersionOne

We selected such models as a case study because of their similarity. As the processes of different bug trackers present the existing variability, such example can be easily used to present for recommendation purposes when modeling a new bug tracking flow for a bucktracking system.

6 Conclusion and future work

This paper focuses on a problem of recommendation methods in BP modeling. Such methods help in speeding up modeling process and producing less error prone models than modeling from scratch. The original contribution of the paper is introducing a categorization of recommendation approaches in BP modeling and short overview of machine learning methods corresponding to the presented recommendations.

Our future work will focus on specifying recommendation approach for company management systems in order to enhance modeling process and evaluation of the selected recommendation methods. We plan to carry out a set of experiments aiming at testing recommendation approaches on various model sets.

References

1. OMG: Business Process Model and Notation (BPMN): Version 2.0 specification. Technical Report formal/2011-01-03, Object Management Group (2011)
2. Silver, B.: BPMN Method and Style. Cody-Cassidy Press (2009)
3. Mendling, J., Reijers, H.A., van der Aalst, W.M.P.: Seven process modeling guidelines (7pmg). Information & Software Technology **52** (2010) 127–136
4. Koschmider, A., Hornung, T., Oberweis, A.: Recommendation-based editor for business process modeling. Data & Knowledge Engineering **70** (2011) 483 – 503
5. 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
6. 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

7. 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
8. 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
9. 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)
10. 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
11. 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
12. 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
13. 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
14. 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
15. Leopold, H., Smirnov, S., Mendling, J.: On the refactoring of activity labels in business process models. Information Systems **37** (2012) 443–459
16. 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
17. Jannach, D., Zanker, M., Felfernig, A., Friedrich, G.: Recommender Systems An Introduction. Cambridge University Press (2011)
18. Wei, K., Huang, J., Fu, S.: A survey of e-commerce recommender systems. In: Service Systems and Service Management, 2007 International Conference on. (2007) 1–5
19. Bobek, S., Nalepa, G.J.: Overview of context-aware reasoning solutions for mobile devices. proposal of a rule-based approach. Computer Science and Information Systems (2014) accepted.
20. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag New York, Inc., Secaucus, NJ, USA (2006)
21. Mitchell, T.M.: Machine Learning. MIT Press and The McGraw-Hill companies, Inc. (1997)
22. Koller, D., Friedman, N.: Probabilistic Graphical Models: Principles and Techniques. MIT Press (2009)
23. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian network classifiers. Machine Learning **29** (1997) 131–163
24. Rabiner, L., Juang, B.H.: An introduction to hidden markov models. ASSP Magazine, IEEE **3** (1986) 4–16

A PROLOG Framework for Integrating Business Rules into JAVA Applications

Ludwig Ostermayer, Dietmar Seipel

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

Abstract. Business specifications – that formerly only supported IT development – increasingly become business configurations in the form of rules that can be loaded directly into IT solutions. PROLOG is well–known for its qualities in the development of sophisticated rule systems. It is desirable to combine the advantages of PROLOG with JAVA, since JAVA has become one of the most used programming languages in industry. However, experts of both programming languages are rare.

To overcome the resulting interoperability problems, we have developed a framework which generates a JAVA archive that provides methods to query a given set of PROLOG rules; it ensures that valid knowledge bases are transmitted between JAVA and PROLOG. We use XML Schema for describing the format for exchanging a knowledge base between PROLOG and JAVA. From the XML Schema desciption, we scaffold JAVA classes; the JAVA programmer can use them and fill in the open slots by statements accessing other JAVA data structures. The data structure on the JAVA side reflects the complex structured knowledge base, with which the PROLOG rules work, in an object–oriented way.

We can to some extend verify the correctness of the data set / knowledge base sent from JAVA to PROLOG using standard methods for XML Schema. Moreover, we can add constraints that go beyond XML. For instance, we can specify standard integrity constraints known from relational databases, such as primary key, foreign key, and not–null constraints. Since we are dealing with complex structured XML data, however, there can be far more general integrity constraints. These can be expressed by standard PROLOG rules, which can be evaluated on the PROLOG side; they could also be compiled to JAVA by available PROLOG to JAVA converters such as Prolog Cafe – since they will usually be written in a supported subset of PROLOG.

We have used our framework for integrating PROLOG business rules into a commercial E–Commerce system written in JAVA.

Keywords. Business Rules, Logic Programming, PROLOG, JAVA.

1 Introduction

PROLOG is well–known for its qualities in rapid prototyping and agile software development, and for building expert systems. In this paper we present an approach that

allows to integrate PROLOG rules seamlessly into JAVA applications. We could largely automate the integration process with our framework PBR4J (PROLOG Business Rules for JAVA). PBR4J uses XML Schema documents, from which it generates (scaffolds) JAVA classes containing the information necessary for utilizing the business rules. The business rules are accessed from JAVA simply by invoking the generated JAVA methods. From the JAVA point of view, the fact that a set of PROLOG rules is requested is hidden. The derived facts can be accessed as a result set by JAVA getter methods. In terms of Domain Specific Languages (DSL) [8], we use PROLOG as an external DSL for expressing rules. Thus, our approach enables a clean separation between a JAVA application and the business logic, and applications can benefit from the strengths of both programming languages.

There exists the following *related work*. We have already discussed the usage of DROOLS [6], a popular JAVA tool for business rules development, and the advantages of knowledge engineering for business rules in PROLOG [11, 12]. There are several solutions for a communication between JAVA and PROLOG, for instance the bidirectional PROLOG/JAVA interface JPL [17] that combines certain C functions and JAVA classes. On the JAVA side, JPL uses the JAVA Native Interface (JNI), on the PROLOG side it uses the PROLOG Foreign Language Interface (FLI). When working with JPL, one has to create rather complex query strings and explicitly construct term structures prior to querying. Slower in performance than JPL, INTERPROLOG [5] provides a direct mapping from JAVA objects to PROLOG terms, and vice versa. PROLOG CAFE [2] translates a PROLOG program into a JAVA program via the Warren Abstract Machine (WAM), and then compiles it using a standard JAVA compiler. PROLOG CAFE offers a core PROLOG functionality, but it lacks support for many PROLOG built–in predicates from the ISO standard.

However, the challenge of our work was not to develop another interface between JAVA and PROLOG, but to simplify the access to the PROLOG rules and data structures in JAVA. We did not mix PROLOG and JAVA syntax for querying the PROLOG rules in JAVA. Rules can be developed independently from JAVA, and our framwork ensures only valid calls from JAVA to the PROLOG rules. We just write the rules in PROLOG and use PBR4J to generate JAVA classes; request and result handling are encapsulated in standard JAVA objects. Therefore in JAVA, the flavour of programming is unchanged. On the other side, the easy–to–handle term structures and the powerful meta–predicates of PROLOG can be used to develop sophisticated rule systems. Furthermore, using PROLOG's parsing techniques (DCGs) and infix operators, the rule syntax can be largely adapted to a natural language level, which simplifies the rule creation process and improves the readability of the rules. In particular, this is important to bridge the gap between software engineers and business analysts without programming background.

The structure of this paper is as follows. Section 2 presents a set of business rules written in PROLOG, which will serve as a running example. In Section 3, we describe our framework: first, we represent a knowledge base in XML and generate a corresponding XML Schema. Then, we generate JAVA classes from the XML schema. In Section 4, we give an example of a JAVA call to the business rules in PROLOG. Finally, we summarize our work in Section 5.

2 Business Rules in PROLOG

In the following, we present a set of business rules in PROLOG, that is part of a real commercial Enterprise Resource Planning (ERP) system for online merchants. The purpose of the business rules is to infer financial key data and costs in a given E–Commerce scenario that is dealing with articles, online shopping platforms, shipping parameters, and various other parameters. The derived data support the business intelligence module of the application, which is implemented in JAVA.

Due to space restrictions, we present only a simplified version of the original set of business rules used in the application. We focus on a constellation consisting of order, platform and shipment charges. For every shipment, taxes have to be paid according to the country of dispatch. In our example, the inferred financial key data are gross margin, contribution margin and profit ratio. First, we describe the input data format necessary for a valid request, then we take a closer look at the business rules. Finally, we explain how to describe relationships between facts in a knowledge base and how to check them in PROLOG.

2.1 The Knowledge Base

The input knowledge base consists of global data and orders. A PROLOG fact of the form `tax(Country, Rate)` describes the purchase tax rate of a country. The PROLOG facts of the form `platform_charges(Category, Commission, Discount)` describe the different commissions that online shopping platforms charge according to article categories and merchants discount [7]. A PROLOG fact of the form `shipping_charges(Country, Logistician, Charges)` shows the price that a logistician charges for a shipment to a given country.

Listing 1.1: Global Data

```
tax('Germany', 0.190).
platform_charges('Books', 0.11, 0.05).
shipping_charges('Germany', 'Fast Logistics', 4.10).
```

An order is a complex data structure – represented by a PROLOG term – consisting of an article, the country of dispatch, and the used logistician. An article is a data structure relating a category and the prices (in Euro), i.e., base price and market price; every article has a unique identifier EAN (European Article Number; usually 13 digits, but we use only 5 digits in this paper).

Listing 1.2: An Order

```
order( article('98765', 'Books', prices(29.00, 59.99)),
   'Germany', 'Fast Logistics' ).
```

2.2 The Rule Base

The following business rule demonstrates the readability and compactness offered by PROLOG. Join conditions can be formulated easily by common variable symbols, and

the term notation offers a convenient access to objects and subobjects in PROLOG; more than one component can be accessed in a single line. Usually, If–Then–Else statements with many alternatives are hard to review in JAVA, but much easier to write and read in PROLOG. Due to the rules approach, multiple results are inferred implicitly; in a DATALOG style evaluation, there is no need to explicitly encode a loop. In a PROLOG style evaluation, all results can be derived using the meta–predicate `findall/3`.

Using the PROLOG package DATALOG* [14] from the DISLOG Developers' Kit (DDK), we can, e.g., support the development phase in PROLOG by visualizing the rule execution with proof trees [11]. DATALOG* allows for a larger set of connectives (including conjunction and disjunction), for function symbols, and for stratified PROLOG meta–predicates (including aggregation and default negation) in rule bodies.

The main predicate in the business rule base computes the financial key data for a single order. The facts of the input knowledge base will be provided by the JAVA application, as we will see later. Derived `financial_key_data/2` facts are collected in a PROLOG list, which will be presented as a result set to JAVA.

Listing 1.3: Business Rules for Financial Key Data

```
financial_key_data(Order, Profits) :-
   order_to_charges(Order, Charges),
   Order = order(article(_, _, prices(Base, Market)), _, _),
   Charges = charges(Shipping, Netto, Fees).
   Gross_Profit is Netto - Base,
   C_Margin is Gross_Profit - Fees - Shipping,
   Profit_Ratio is C_Margin / Market,
   Profits = profits(Gross_Profit, C_Margin, Profit_Ratio).

order_to_charges(Order, Charges) :-
   Order = order(Article, Country, Logistician),
   Article = article(_, Category, prices(_, Market)),
   call(Order),
   tax(Country, Tax_Rate),
   shipping_charges(Country, Logistician, Charges),
   Shipping is Charges / (1 + Tax_Rate),
   Netto is Market / (1 + Tax_Rate),
   platform_charges(Category, Commission, Discount),
   Fees is Market * Commission * (1 - Discount),
   Charges = charges(Shipping, Netto, Fees).
```

The predicate `order_to_charges/4` first computes the charges for the shipment, then an article's netto price using the tax rate of the country of dispatch, and finally the fees for selling an article on the online platform in a given category. We use the PROLOG terms `Order`, `Profits`, and `Charges` to keep the argument lists of the rule heads short. E.g., `order_to_charges/4` extracts the components of `Order` in line 2 and calls the term `Order` in line 4. Thus, we can avoid writing the term `Order` repeatedly – in the head and in the call. In the code, we can see nicely, which components of `Order` are used in which rule, since the other components are labeled by underscore variables.

2.3 Constraints in PROLOG

In knowledge bases, facts often reference each other. E.g., in our business rules application, we have the following foreign key constraints: for every `order/3` fact, there must exist corresponding facts for `tax/2` and `shipping_charges/4`, whose attribute values for `Country` match the attribute value for `Country` in `order/3`. The same holds for `category` in `platform_charges/3` and `category` in `order/3`. Another frequently occuring type of constraints are restrictions on argument values; e.g., the values for `Country` could be limited to countries of the European Union.

This meta information between facts in a knowledge base usually remains hidden; the developer of the rule set knows these constraints, and only sometimes they are easy to identify within the set of business rules. For validation purposes of knowledge bases, however, this information is crucial, in particular when a knowledge base for a request is arranged by a programmer other than the creator of the set of rules.

Constraints, such as the foreign key constraints from above, can simply be specified and tested in PROLOG. The execution of the PROLOG predicate `constraint/1` is controlled using meta–predicates for exception handling from SWI PROLOG. With `print_message/2`, a meaningsful error message can be generated, and exceptions can be caught with `catch/3`. In Listing 1.4, the foreign key constraints on `Country` and `Category` are checked.

We can also represent standard relational constraints in XML. XML representations for `create table` statements have been developed and used in [3, 16]. Thus the knowledge base – including the constraints – can be represented in XML.

Listing 1.4: Foreign Key Constraints

```
constraint(fk(shipping_charges)) :-
   forall( shipping_charges(Country, _, _),
      tax(Country, _) ).

constraint(fk(article_charges)) :-
   forall( article(_, Category, _),
      platform_charges(Category, _, _) ).
```

3 Integration of PROLOG Business Rules into JAVA

The workflow of PBR4J follows three steps, cf. Figure 1. First, PBR4J extracts an XML Schema description for the knowledge base and the result set of a given set of PROLOG rules. Then, the user must extend the extracted XML Schema by names for atomic arguments, numbers and strings from PROLOG and review the type description. Finally, PBR4J uses the XML Schema to generate JAVA classes and packs the generated classes into a JAVA Archieve (JAR). After embedding the JAR into the JAVA application, the set of PROLOG rules can be called from JAVA. The facts derived in PROLOG are sent back to JAVA, where they are parsed; then, they can be accessed by the generated classes.

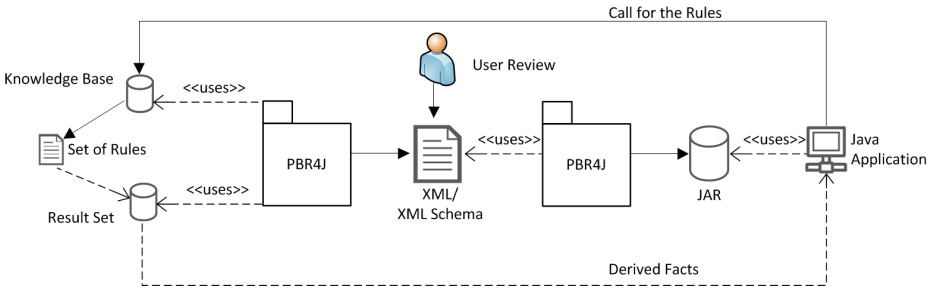


Fig. 1: Workflow of PBR4J

In the following, we describe the transformation of the knowledge base to an XML representation, from which we subsequently extract the XML Schema. Then we show that the JAVA classes generated from the XML Schema reflect the complex structured knowledge base in an object–oriented way. The result set is handled in a similar manner; thus, we describe only the transformation of the knowledge base and omit further processing details for the result set.

3.1 An XML Schema for the Knowledge Base

XML is a well–known standard for representing and exchanging complex structured data. It allows for representing PROLOG terms and improves the interoperability between PROLOG and JAVA programs, since XML is easy to read. We extract an XML Schema from the XML representation of the knowledge base, and we generate JAVA classes from the extracted XML Schema.

We use the predicate `prolog_term_to_xml(+Term, -Xml)` for the transformation of a PROLOG term to XML. Listing 1.5 shows the XML representation for the PROLOG term with the predicate symbol `order/3`. Notice the XML attribute `type` and the names of elements representing arguments of complex terms on the PROLOG side.

Listing 1.5: An Order in XML Format

```
<order type="class">
   <country type="string">Germany</country>
   <logistician type="string">Fast Logistics</logistician>
   <article type="class">
      <ean type="integer">98765</ean>
      <category type="string">Books</category>
      <prices type="class">
         <base type="decimal">29.00</base>
         <market type="decimal">59.99</market>
      </prices>
   </article>
</order>
```

These are necessary, because JAVA is a typed language, whereas PROLOG builds data structures from a few basic data types. The representation for class attributes in JAVA is a typed `Name="Value"` pair. In order to map the knowledge base from PROLOG to JAVA, we must give names to arguments of PROLOG facts, if they are atomic, numbers, or strings, and we must add a type information. The functor of a complex PROLOG term is mapped to the tag of an element with `type="class"`. The structure of the XML representation easily can be generated from the PROLOG term structure, and some of the type information can be inferred automatically from the basic PROLOG data types. But, type preferences and meaningful names for atoms, numbers, and strings must be inserted manually.

From the XML representation of the knowledge base and the result set, we can extract a describing XML Schema using PROLOG. The XML Schema is a natural way to describe and to define the complex data structure. Known techniques are available for validating the XML representation of the knowledge base w.r.t. the XML Schema. Listing 1.6 shows the description of an `order/3` term in XML Schema. The XML Schema of the knowledge base can contain further information in attributes like `minOccurs` and `maxOccurs`.

Listing 1.6: Fragment of the XML Schema describing `order/3`

```
<xsd:element name="order" type="order_Type"
   minOccurs="1" maxOccurs="unbounded" />

<xsd:complexType name="order_Type">
   <xsd:sequence>
      <xsd:element name="article" type="article_Type" />
      <xsd:element name="country" type="xsd:string" />
      <xsd:element name="logistician" type="xsd:string" />
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="article_Type">
   <xsd:sequence>
      <xsd:element name="ean" type="xsd:integer" />
      <xsd:element name="category" type="xsd:string" />
      <xsd:element name="prices" type="prices_Type" />
   </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="prices_Type">
   <xsd:sequence>
      <xsd:element name="base" type="xsd:decimal" />
      <xsd:element name="market" type="xsd:decimal" />
   </xsd:sequence>
</xsd:complexType>
```

3.2 Scaffolding of JAVA Code

From the XML Schema, we generate JAVA classes using the PROLOG–based XML transformation language FNTRANSFORM [13]. FNTRANSFORM offers recursive transformations of XML elements using a rule formalism similar to – but more powerful than – XSLT. Every `xsd:element` in the schema with a complex type will be mapped to a JAVA class. Child elements with simple content are mapped to attributes. Figure 2 shows a fragment of the UML diagram for the generated classes.

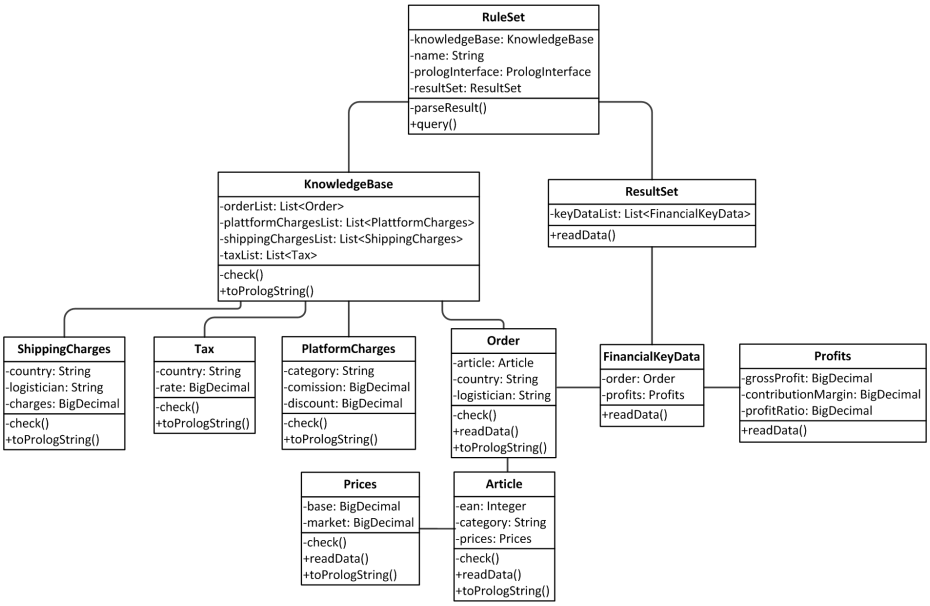


Fig. 2: Generated Classes

All classes associated with the class `KnowledgeBase` implement the methods `check` and `toPrologString`. An example of the method `toPrologString` of the generated class `Order` is shown in Listing 1.7. A recursive call of `check` controls that all necessary input data are set before the method `toPrologString` is called to build a knowledge base in a string format, which can be parsed easily by PROLOG using the predicate `string_to_atom/2`. The transformation to a PROLOG term can be achieved by `atom_to_term/3`.

Parts of the generated class `RuleSet` are shown in Listing 1.8. The method `query` sends a PROLOG goal together with a knowledge base in string format from JAVA to PROLOG. As a default interface between JAVA and PROLOG, we have implemented a simple connection with a communication layer based on standard TCP/IP sockets. Other interfaces can be implemented and set as the value for the attribute `prolog-Interface` of the class `RuleSet`. The default interface is represented by the class `PrologInterface`, which is fix and not generated every time a set of PROLOG rules

is integrated into a given JAVA application via PBR4J. The class `PrologInterface` must be integrated into the JAVA application once, and it must be accessible for all generated classes of the type `RuleSet`.

Listing 1.7: `toPrologString` in `Order`

```
public String toPrologString() {
   this.check();
   StringBuilder sb = new StringBuilder();
   sb.append( "order" + "(" +
      this.article.toPrologString() + ", "
      "'" + this.getCountry() + "'" + ", "
      "'" + this.getLogistician() + "'" + ")" );
   return sb.toString();
}
```

The result set that is sent back from PROLOG to JAVA is parsed by the method `parseResult` of the class `RuleSet`. As for the class `PrologInterface`, the class `PrologParser` is not generated and must be integrated into the JAVA application once and be accessible for all generated classes of the type `RuleSet`. The method `parseProlog` of `PrologParser` saves the content of the string returned from PROLOG in a structured way to a hashmap. The hashmap than can be further processed efficiently by the method `readData` that all classes associated with the class `ResultSet` must implement. The method `readData` analyses the hashmap and fills the data list of the class `ResultSet`.

Listing 1.8: The Class `RuleSet`

```
package pbr4j.financial_key_data;

public class RuleSet {
   private PrologInterface prologInterface = null;
   private String name = "financial_key_data";
   private KnowledgeBase knowledgeBase = null;
   private ResultSet resultSet = null;
   // ... code that we omit...
   private void parseResponse(String prologString) {
      DataList data = PrologParser.parseProlog(prologString);
      this.resultSet = new ResultSet();
      this.resultSet.readData(data); }
   // ... code that we omit...
   private void query(KnowledgeBase kb) {
      if (prologInterface == null) {
         this.setDefaultInterface(); }
      String response = prologInterface.callProlog(
         this.name, kb.toPrologString());
      this.parseResponse(response); }
   // ... code that we omit...
}
```

All generated classes are organised in a namespace via a JAVA package. The package access protection ensures that the class `RuleSet` can only contain a `KnowledgeBase` from the same package. The package can be stored in a JAVA Archive (JAR) – a compressed file that can not be changed manually. This creates an intentional generation gap, cf. Fowler [8]. The JAR file can be embedded into any JAVA application easily, and all classes in the JAR become fully available to the JAVA developers.

4 A JAVA Call to the Business Rules

In the following, we will give a short example of a request to a set of PROLOG rules using the classes generated with PBR4J. Listing 1.9 shows a test call from JAVA to the set of business rules described in Section 2. We omit object initialisation details, but we assume that the necessary objects for a successful call are provided. For improving the readability of the result of the call, we assume that all classes associated with the class `ResultSet` implement the method `toPrologString`.

Listing 1.9: A JAVA Call to the Business Rules

```
import pbr4j.financial_key_data.*;

public class TestCall {

   public static void main(String[] args) {
      RuleSet rules = new RuleSet();
      KnowledgeBase kb = new KnowledgeBase();
      // ... filling the knowledge base with data ...
      rules.query(kb);
      ListIterator<Object> it =
         rules.getResultSet().listIterator();
      while (it.hasNext()) {
         System.out.println(it.next().toPrologString() + ".");
      } } }
```

It is not visible in JAVA that a request is made to a rule set written in PROLOG. Running the JAVA code from above creates the system output shown in Listing 1.10; we have added some newlines to improve readability. The first fact is derived from the data described in Subsection 2.1. The second fact is derived from another order of the same article, that is shipped to France. Charges for the shipment to a foreign country are higher, and the tax rate of France is 0.196, which explains the slightly lower argument values of `profits/3`.

Listing 1.10: order Result Set

```
financial_key_data(
   order( article('98765', 'Books', prices(29.00, 59.99)),
      'Germany', 'Fast Logistics' ),
   profits(21.41, 11.70, 0.195) ).

financial_key_data(
```

```
order( article('98765', 'Books', prices(29.00, 59.99)),
   'France', 'Fast Logistics' ),
profits(21.16, 7.70, 0.128) ).
```

5 Conclusions

We have presented a largely automatic approach for integrating a set of PROLOG rules seamlessly into JAVA applications. XML Schema is used for specifying the XML format for exchanging a knowledge base and a result set, respectively, between PROLOG and JAVA.

On the PROLOG side, we use a generic mapping from the PROLOG representation of the knowledge base and the result set to an XML representation enriched by data type information and names for atoms or numbers, and we extract a describing XML Schema from the XML representation. On the JAVA side, we scaffold JAVA classes from the XML Schema, that reflect the complex structured PROLOG terms in an object–oriented way. Accessing a set of rules from JAVA is simply done by invoking the JAVA methods of the generated classes without programming PROLOG or creating complex query strings.

We have illustrated our approach using a set of business rules that we have already integrated with our framework into a commercial E–Commerce system written in JAVA.

Acknowledgement. We acknowledge the support of the Trinodis GmbH.

References

1. S. Abiteboul, P. Bunemann, D. Suciu: *Data on the Web – From Relations to Semi–Structured Data and* XML*,* Morgan Kaufmann, 2000.
2. M. Banbara, N. Tamura, K. Inoue.: *Prolog Cafe: A Prolog to Java Translator,* Proc. Intl. Conf. on Applications of Declarative Programming and Knowledge Management (INAP) 2005, Springer, LNAI 4369.
3. A. Böhm, D. Seipel, A. Sickmann, M. Wetzka: *Squash: A Tool for Designing, Analyzing and Refactoring Relational Database Applications.* Proc. Intl. Conf. on Applications of Declarative Programming and Knowledge Management (INAP) 2007, Springer, LNAI 5437.
4. H. Boley: *The Rule Markup Language:* RDF–XML *Data Model,* XML *Schema Hierarchy, and* XSL *Transformations.* Proc. Intl. Conf. on Applications of Declarative Programming and Knowledge Management (INAP) 2001, Springer, LNAI 2543.
5. M. Calejo: *InterProlog: Towards a Declarative Embedding of Logic Programming in Java,* Proc. 9th European Conference on Logics in Artificial Intelligence, JELIA, 2004.
6. *Drools – The Business Logic Integration Platform.* `http://www.jboss.org/drools/`.
7. *Ebay Seller Fees.* `http://pages.ebay.de/help/sell/seller-fees.html`.
8. M. Fowler. *Domain–Specific Languages.* Addison–Wesley, 2011.
9. E. Gamma, R. Helm, R. Johnson, J. Vlissides. *Design Patterns. Elements of Reusable Object–Oriented Software.* Addison–Wesley Longman, 2010.
10. B. v. Halle. *Business Rules Applied.* Wiley, 2002.
11. L. Ostermayer, D. Seipel. *Knowledge Engineering for Business Rules in* PROLOG. Proc. Workshop on Logic Programming (WLP), 2012.

12. 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.
13. D. Seipel. *Processing* XML*–Documents in Prolog.* Proc. 17th Workshop on Logic Programming (WLP), 2002.
14. D. Seipel. *Practical Applications of Extended Deductive Databases in* DATALOG*. Proc. Workshop on Logic Programming (WLP) 2009.
15. D. Seipel. *The* DISLOG *Developers' Kit (*DDK*).* http://www1.informatik.uni-wuerzburg.de/database/DisLog/
16. D. Seipel, A. Boehm, M. Fröhlich: *Jsquash: Source Code Analysis of Embedded Database Applications for Determining* SQL *Statements.* Proc. Intl. Conf. on Applications of Declarative Programming and Knowledge Management (INAP) 2009, Springer, LNAI 6547.
17. P. Singleton, F. Dushin, J. Wielemaker: *JPL: A Bidirectional Prolog/Java Interface* http://www.swi-prolog.org/packages/jpl/, 2004.
18. J. Wielemaker. SWI PROLOG *Reference Manual* http://www.swi-prolog.org/pldoc/refman/

Knowledge Acquisition for Life Counseling

Régis Newo and Klaus-Dieter Althoff

German Research Center for Artificial Intelligence, DFKI GmbH,
Research Group Knowledge Management,
Competence Centre Case-Based Reasoning
Email: firstname.surname@dfki.de

Abstract. In this paper, we explain how highly unstructured domain knowledge can be acquired and integrated in a case-based reasoning system. We apply our approach to the life counseling domain. We introduce the two steps of our knowledge acquisition approach in such unstructured domains. The first step is manual and relies on domain experts. The second step is automatic and uses information extraction techniques. Our approach has the potential to contribute to the formalizing and establishing of an important subset of life counseling terminology. In addition, our approach could serve as an example for comparable weak theory domains.

1 Introduction

Case-Based Reasoning (CBR) is a methodology for problem solving based on the fact that previously experienced knowledge can be used to solve new problems [1]. It has been successfully applied in different domains like for example medicine [2], help-desk [3] or technical diagnosis [4]. The needed knowledge used in a CBR system is stored in the so-called knowledge containers (vocabulary, similarity measures, adaptation knowledge and the case base) [5]. The amount of knowledge available for each container depends on the application domain. For application domains, in which a certain level of formalization is already achieved, it might be easier to fill the vocabulary and similarity measures containers. Whereas it might be easier to fill the case base container in unformalized and/or unstructured application domains.
Our application SeBaPort (Portal for counseling, in German Seelsorge- und Beratungsportal) deals with life counseling. Life counseling deals with the wellbeing of humans. Life counselors conduct converstions with consulters, give advices and help them to help themselves. Counselors often rely on past expriences for the counseling. The goal of SeBaPort is to help counselors by providing them with counseling cases (depending on their requests), which they can learn from. This makes CBR an ideal methodology to process the knowledge used in that domain.
Life counseling is a domain which is highly unstructured. This makes it very difficult to develop a CBR system for life counseling and be able to provide knowledge in the previously mentioned knowledge containers. For this application domain, we would have to develop an initial set of vocabulary and similarity

measures, and also find a methodology to process the available (unstructured) cases and store them in our case base. SeBaPort does not aim at providing solutions for a given counsel or problem, primarily because the acceptance of the counselors would significantly diminish, if we claim to be able to provide complete solutions to counseling problems.
In order to build a life counseling CBR system, we started by developing an initial CBR model and acquiring structured cases. In this paper, we describe in Section 3 how we designed our initial CBR model and our approach for the acquisition of (structured) cases in Section 4. Afterwards we will present some related work in Section 5 and will conclude the paper in Section 6. In the next section, we will first give an elaborate presentation of the life counseling domain.

2 Life Counseling

Life counseling is concerned with the welfare of human beings, more precisely the thinking, feeling, acting, and also the faith of persons. Life counselors help people deal with their problems and conflicts. They conduct several counseling interviews with the consulters. The main idea is to help people help themselves by having several discussions with them, give them multiple views on their problem and give them basic hints. Life counselors for example give exercises, which are part of a counseling method, to consulters after an interview. During the following interviews, they try to find out, whether it helped the consulter or it should be changed.
In order to do that, counselors themselves mainly rely on their experience in the domain, but also on the methodical knowledge they learned during their formation. They are grouped in small communities to share their experiences. As they do not only build on self-made experiences but also on those from others, they often rely on peer consulting and supervision to critically analyse past cases (and be able to learn from them). Further they contact other colleagues when they need help in an actual or past counseling case. Such help might comprise a whole counseling case or just information about parts or aspects (e.g., the method or exercise that can be used in a given situation) of life counseling.
Our goal is to provide a system that can be used to help life counselors in their work. We want to provide a decision support system that helps the counselors to document and share their experiences. They would also learn new cases from others and be able to find hints and references (e.g., to counseling methods) while looking for help (when they deal with a given case). The intended functionality of our system is presented in figure 1 on the basis of the CBR cycle.

An example of the description of the patient's problem in a documented case is given below.

> Woman, 48 years old, married and 3 children: 12, 15 and 18 years old. She has been working shifts as a full-time midwife for 2 years. She attends counseling because of insomnia due to her often alternating shift work. She has particularly problems with insomnia after night shifts. She

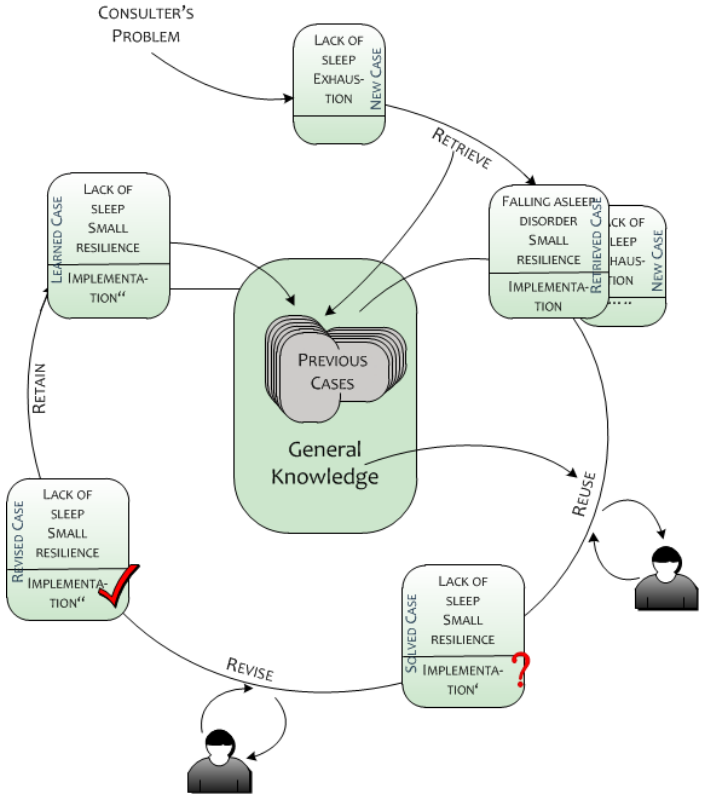


Fig. 1. CBR Cycle in Life Counseling

cannot ignore surrounding noises and she cannot completely darken her bedroom. As a consequence of this, she is often tired, is not able to work under pressure and suffers from headaches.

The documentation of the case also contains the documentation of each interview with for example the applied methods, goal validation and solution interventions.

3 Case Model for Life Counseling

When developing CBR systems, one of the first challenges is to fill the knowledge containers. We have to evaluate which kind of knowledge is availaible in order to know which containers can be filled. In life counseling, the knowledge that is easier to acquire is the experiences made by the experts represented as cases. Due to lack of formalization in the domain, we need to find a way to extract formalized knowledge from the available cases. For that, we want to structure the information contained in cases.

Section	Subsection	Parameters
Anamnesis	Personal data	gender, age, nationality, marital status, children, housing situation, religion, relevant pre-existing illnesses, treatments and therapies, medication,
	Biography	origin family, present family, close persons, life crises, life breach crises, graduation, career, financial security
Relationship		(relations between the consulters, if there were at least two in the counseling session)
Problem review		cause, symptoms, further effects, suffering level, suffering description, first appearance and development, previous approaches, inner resources, outer resources
Mandate		whose idea, who has the problem, expectations, indications for success, counselor's decision
Goal review		primary goal, secondary goal, planning, goal verification, goal restatement
Setting		counseling mode, place, unit length, number of sessions, frequency, recording, global information
History		solution interventions, applied methods, goal validation, past approaches

Fig. 2. Structure of a life counseling case

Most experts have their own way to write down their cases. Furthermore, the cases do not contain the same kind of information, have different levels of detail and elaborateness. It is thus nearly impossible to automatically detect a structure in raw cases.

Our approach to structure the available cases is to get the needed information from the experts. Instead of getting unstructured cases, we want to be able to get semi-structured cases from the experts. For that purpose, we elaborated a structure that should be used by the experts. The used structure has to reflect the way of thinking of life counselors. We thus have to involve experts in order to define such a structure.

Table 2 shows the structure for life counseling cases that we developed. It is based on a preliminary study done with domain experts (i.e. counselors) [6]. We validated the structure by comparing it with a doctor's report used in a clinic for psychotherapy and psychosomatic medicine. It shows that a case can contain a multitude of information. Although a given case must not contain all possible information (i.e. each parameter of the structure must not be filled), it would be very difficult to automatically map the knowledge from a given case to the parameters. We used the defined case structure to develop a CBR model with an initial vocabulary and an initial set of similarity measures. The CBR model is more detailled, so we can have a better description of the cases and also more

precise similarity measures. For example, the medication (in personal data) has following attributes:

- the name of the medication,
- the generic type,
- the active substance,
- the daily dosage, etc.

As another example, the CBR has attributes for the number of children as well as the gender and the age of each children.

4 Two-Step Case Acquisition

Now that we defined a case model, the next challenge is to fill our case base with life counseling cases following the model. Unfortunately, counselors do not have a formal manner to document their cases. This leads to unstructured case descriptions. In order to be able to use those cases in our CBR system, we have to find methods to formalize the existing knowledge (i.e. cases). This is a difficult task because of the diversity of information available in a case, as can be seen in the last section. Our approch for the knowledge acquisition consists of two steps.

The goal of the first step is to organize the available information. This is a manual step in which the available diversified information is mapped to the structure defined in Section 3. We rely on domain experts to cope with this assignment. In SeBaPort, this step is realized by providing experts with web forms, which can be used to enter the cases. At the end of this step, we have a case description that matches the structure defined in Table 2.

The second step of the knowledge acquisition is automatic and consists of using information extraction to obtain structured CBR cases. The complexity of this step depends on the type of information given by the experts in the first step. Some information, like the gender or the nationality of the patients can be easily matched to a formal case model. Other, like the medication or the children, need more effort for the formalization. For example, the way information about medication is given differs from one expert to another and can be more or less expressive. Nevertheless we have to be able to match the natural language description of the medication information to our formal model which contains the additional attributes defined at the end of Section 3. Another example is the attribute children, for which the same holds. From the given description, we have to be able to identify, if given, the number of children, the age and/or gender of each children and so on. In the example given in Section 2, we would identify 3 children and the ages of the children. However, the gender are not documented. We used information extraction techniques provided by the component ANNIE of the framework GATE (see [7]) to tackle this challenge.

Another goal we pursue in SeBaPort, is to be able to learn from the acquired cases in order to formalize the application domain. We thus want to perform a stepwise knowledge formalization for life counseling. This has to be done from

scratch because the domain, as explained earlier, is highly unstructured. We are actually trying to gain the formalized knowledge from the acquired cases. The purpose is to be able to tackle the fact that there are not only several ways to document a counseling case, but also different counseling perspective. The representatives of each perspective often have problems to deal with case documentations from other perspectives. A formaliztion like the one we are targeting would promote the intercommunication between the representatives of the diffent perspectives.

5 Related Work

The idea of using CBR in medical related domains has been explored in the last couple of years. In [2] the authors present four recent CBR applications in different medical domains. The applications deal with:

- Long-term follow-up of oncology patients
- Assistance of type 1 Diabetes patients
- Support of physicians in the domain of end stage renal disease
- Diagnosis and treatment of stress.

There have been many other CBR applications in medical domains. Nevertheless, to our knowledge, SeBaPort is the first one to deal with life counseling.
As for knowledge formalization, there are also other approaches that deal with that topic. One of them is the knowledge formalization continuum presented in [8]. The authors present a process for knowledge development based on a flexible organization of knowledge. The main difference between our aproach and this one (as well as many other knowledge formalization approaches) is that the only initially available knowledge in life counseling are the unstructured case descriptions. That is, our inital information can hardly be used for learning, classification or even formalization.
In [9], the authors present an approach for knowledge extraction from data taken from forums, which are communities of experts. This approach relies on a initial auxiliary data to extract the knowledge and uses the extracted knowledge to improve the knowledge extraction.

6 Conlusion

In this paper, we presented the domain of life counseling and how the available knowledge can be used to develop a CBR system (SeBaPort). SeBaPort will help life counselors to extend their knowledge and learn from past cases. We showed how we are actually extracting the knowledge from available cases and we want to use it for the knowledge formalization. We intend to test our knowledge acquisition by evaluating the similarity measures with the acquired cases. The evaluation is still an ongoing work. Furthermore we will develop an approach to incorporate experts' feedback to our formalization process.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning : Foundational issues, methodological variations, and system approaches. AI Communications **1**(7) (March 1994)
2. Marling, C., Montani, S., Bichindaritz, I., Funk, P.: Synergistic case-based reasoning in medical domains. Expert Systems with Applications (2013)
3. Roth-Berghofer, T.: Learning from homer, a case-based help desk support system. In Melnik, G., Holz, H., eds.: Advances in Learning Software Organizations. Volume 3096 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2004) 88–97
4. Althoff, K.D.: Machine Learning and Knowledge Acquisition in a Computational Architecture for Fault Diagnosis in Engineering Systems. In Weintraub, M., ed.: Proc. International Machine Learning Conference (ML92), Workshop on "Computational Architectures for Supporting Machine Learning and Knowledge Acquisition". (1992)
5. Richter, M.M.: Fallbasiertes Schließen. In: Handbuch der Künstlichen Intelligenz. Oldenbourg Wissenschaftsverlag Verlag (2003) 407–430
6. Newo, R., Althoff, K.D., Bach, K., Althoff, M., Zirkel-Bayer, R.: Case-Based Reasoning for Supporting Life Counselors. In Cassens, J., Roth-Berghofer, T., Kofod-Petersen, A., Massie, S., Chakraborti, S., eds.: Proceedings of the Workshop on Human-Centered and Cognitive Approaches to CBR at the ICCBR 2011. (Sept 2011)
7. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V.: GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In: Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02). (2002)
8. Baumeister, J., Reutelshoefer, J., Puppe, F.: Engineering intelligent systems on the knowledge formalization continuum. International Journal of Applied Mathematics and Computer Science (AMCS) **21**(1) (2011)
9. Bach, K., Sauer, C.S., Althoff, K.D.: Deriving case base vocabulary from web community data. In Marling, C., ed.: ICCBR-2010 Workshop Proceedings: Workshop on Reasonng From Experiences On The Web. (2010) 111–120

HaDEsclipse – Integrated Environment for Rules (Tool Presentation)*

Krzysztof Kaczor[1], Grzegorz J. Nalepa[1], Krzysztof Kutt[1]

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

Abstract. In the paper a presentation of HaDEsclipse is given. It is an environment for design and implementation of rule-based systems within the Semantic Knowledge Engineering (SKE) approach. It is build with the use of the Eclipse framework, integrating the previously developed components of the HaDEs environment. HaDEsclipse integrates modules for conceptual prototyping of rule bases, and a visual editor for logical design of extended decision tables that group rules working in similar context. It also allows for generating an executable form of the system, that can be later executed by an inference engine. While the SKE is targeted mainly at knowledge engineers, the use of the Eclipse framework makes the development easier for software engineers.

1 Introduction and Motivation

Rule-based systems (RBS) play an important role in knowledge engineering and software engineering, e.g. with the business rules approach [1]. However, practical design of rules is a challenging task. It requires both efficient knowledge representation methods for rule bases, as well as practical design tools that support them. The Semantic Knowledge Engineering (SKE) [2] addresses these problems, by providing the XTT2 [3] and ARD+ [4] representation methods, and a dedicated design framework HaDEs, previously presented at KESE [5].

However, HaDEs turned out to be hard to use for knowledge engineers not familiar with SKE as well as for software engineers. This gave the motivation to develop a new front end to HaDEs, based on the popular Eclipse IDE. In this paper we present this new tool called HaDEsclipse [6]. First, we shortly discuss the SKE design process and how it is supported by HaDEs. Then we present the architecture and selected aspects of implementation of HaDEsclipse.

2 SKE Design Process with HaDEs

Our research concerns the representation and formal verification of RBS. An important result of our research is the SKE (*Semantic Knowledge Engineering*) [2]

* The paper is supported by the AGH UST Grant 15.11.120.361.

methodology, which derives from the HeKatE (*Hybrid Knowledge Engineering*) research project [7]. It aims at providing an integrated process for design, implementation, and analysis of the RBS supported by HADEs (HeKatE *Design Environment*) framework.

The main features of this methodology are:

1. *Visual rule representation.* The provided XTT2 [3] rule representation method that visualizes the rule base in the form of interconnected decision tables, which makes the design more transparent.
2. *Supported rule modeling.* The HADEs framework provides a set of dedicated tools, which facilitate the design process.
3. *Easy rule maintenance.* The HADEs-based design process consists of three stages. The transitions between stages are formally defined and automatically performed. The modification made in one stage can be automatically propagated into the following stages.
4. *One rule type.* As opposed to Business Rules, SKE provides only one type of rule – production rule. However, the methodology provides different inference strategies that correspond to different types of Business Rules, e.g. derivation rule type corresponds to backward chaining inference mode.
5. *Formal rule description and verification.* The provided formal rule language based on the ALSV(FD) (*Attributive Logic with Set of Values over Finite Domains*) logic [7] allows for formalized representation and verification of rules. Moreover, the semantics of rules is precisely defined.

The SKE approach can be applied to a wide range of intelligent systems. In this context, two main areas have been identified in the project: control systems, in the field of intelligent control, and Business Rules [1] and Business Intelligence systems, in the field of software engineering.

The HADEs framework aims at supporting the SKE approach. In this approach, the application logic is expressed using forward-chaining decision rules. They form an intelligent rule-based controller or simply a business logic core. The logic controller is decomposed into multiple modules represented by decision tables. HADEs supports a complete hierarchical design process for the creation of knowledge bases. The whole process consists of three stages: conceptual, logical and physical design and is supported by a number of tools providing the visual design and automated implementation[1].

The conceptual design is the first stage of the process. During this step, the ARD+ (*Attribute Relationships Diagrams*) method is used. The principal idea for this stage is to build a graph defining functional dependencies between attributes on which the rules are built. This stage is supported by two visual tools: VARDA (*Visual* ARD+ *Rapid Development Alloy*), and HQEd.

The logical design is the second stage of the process. During this stage, rules are designed using the visual XTT2 (*Extended Tabular Trees version 2*) [3] method. This phase can be performed as the first one in the design or as the

[1] See: `https://ai.ia.agh.edu.pl/wiki/hekate:hades`

second one, when the input is provided from the conceptual design. It is supported by the dedicated editor HQEd (HeKatE *Qt Editor*). HQEd supports the HML format, which allows for importing models generated by VARDA, as well as for saving and loading the state of the design.

Having a complete XTT2-based model the physical implementation can be generated automatically. In this stage, a logical model is transformed into an algebraic presentation syntax called HMR[2] (HeKatE *Meta Representation*). HMR is a textual representation of the XTT2 logic. It is a human readable form, as opposed to the machine readable HML format. The HMR representation can be directly executed by the dedicated inference engine tool, called HeaRT[3] (HeKatE *Run Time*) [8]. The HeaRT engine has communication and integration facilities. It supports Java integration based on callback mechanism and Prolog JPL library, called JHeroic.

HaDEs proved to be an efficient framework for designing rule bases within the SKE approach. However, its main limitation is that it is a set of loosely connected tools. Moreover, these tools have custom GUIs, which is problematic for engineers not familiar with SKE. This gave motivation for the development of a new platform, providing a more user friendly front end to HaDEs.

3 Architecture of HaDEsclipse

A decision was made to use the popular Eclipse IDE, which is a widely used tool in the software engineering community. Using it a new integrating front end to HaDEs was developed [6]. *HaDEsclipse* was implemented as a plugin for Eclipse. It integrates modules for conceptual prototyping of rule bases, and a visual editor for logical design of extended decision tables grouping rules. It also allows for generating an executable form of the system, that can be later executed by an inference engine. Within this plugin, one can manage the whole SKE design process described in the previous section.

The main functional requirements of HaDEsclipse are aimed at integrating the existing components of HaDEs using Eclipse:

1. ARD+ support:
   (a) Code editor with syntax highlighting, formatter, content assistant and error checking,
   (b) Integration with VARDA,
   (c) Wizard to create new ARD+ files.
2. HML support:
   (a) Code editor with syntax highlighting and checking, content assistant,
   (b) Integration with HQEd,
   (c) Wizard to create new HML files.
3. HMR support:

[2] See `https://ai.ia.agh.edu.pl/wiki/hekate:hmr`.
[3] See `https://ai.ia.agh.edu.pl/wiki/hekate:heart`

(a) Code editor with syntax highlighting, code formatter, content assistant and error checking,
(b) Integration with HeaRT.

4. Preferences card:
   (a) Code editors settings,
   (b) HaDEs environment parameters.
5. Intuitive Eclipse wizards, views and perspective.

The architecture of HaDEsclipse is presented on Fig. 1. It consists of 5 parts: three of them support HaDEs languages and the other two are responsible for view and wizards. Communication with HaDEs environment (VARDA, HQEd, HeaRT) is handled using JHeroic library.

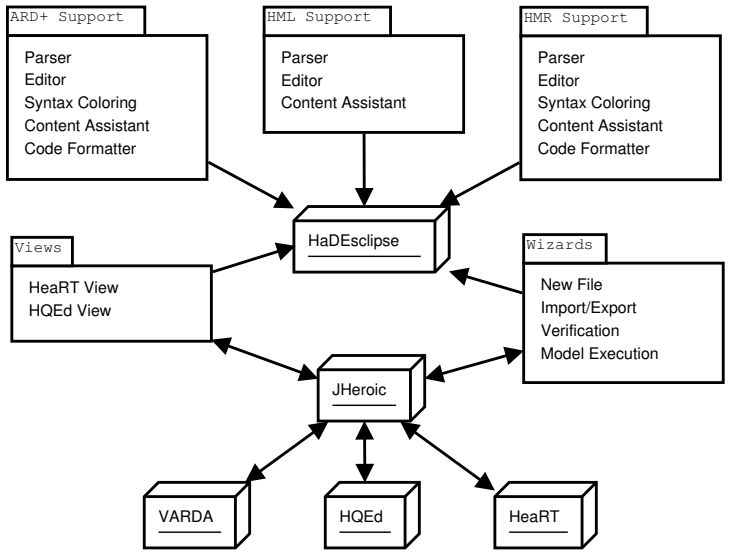


Fig. 1. Architecture of HaDEsclipse

The tool was implemented in Java as a plugin for Eclipse. All of the functional requirements where met. Five modules of HaDEsclipse successfully support the design with SKE. Thanks to HaDEsclipse the models created in the subsequent design phases are easily interchanged between the HaDEs tools. Moreover, the tool allows to run and verify rule models in HeaRT. For the visual design of the XTT2 tables HQEd is used, but files produced with it are exchanged with other tools transparently for the user.

An example session with the tool is presented in Figures 2 and 3. In the first figure the conceptual design with ARD+ is presented. The conceptual model of the rule base is described using a set of attributes and dependencies between them. The HML file contains prototypes of decision tables holding the conditional and decision attributes. In the second figure the HMR editing process is presented. The plugin supports both syntax highlighting and hinting, as well as

a structured XML editor in the case of ARD+. Schemas (headers) of the tables are defined base on the HML description. In given tables rules are defined using the `xrule` construct.

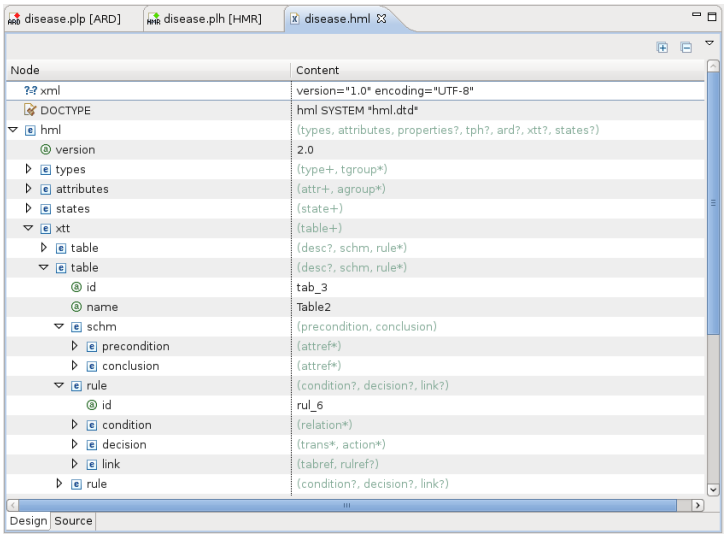


Fig. 2. HML Editor

4 Summary and Future Work

In the paper the HaDEsclipse tool was presented. It is an integrating front-end for the HADEs framework, which supports the knowledge engineering process in the SKE approach [2]. The new tool makes HADEs more accessible and useful for software engineers.

Our future works include further integration of HaDEsclipse with other tools we developed. This includes design tools for business processes, and integration with business process engines. Finally, recent results include an Eclipse-based tool for generating test cases for unit testing based on a rule-based specification. This framework will be integrated with HaDEsclipse, bringing more practical benefits from the area of knowledge engineering to software engineers [9].

References

1. von Halle, B.: Business Rules Applied: Building Better Systems Using the Business Rules Approach. Wiley (2001)
2. Nalepa, G.J.: Semantic Knowledge Engineering. A Rule-Based Approach. Wydawnictwa AGH, Kraków (2011)

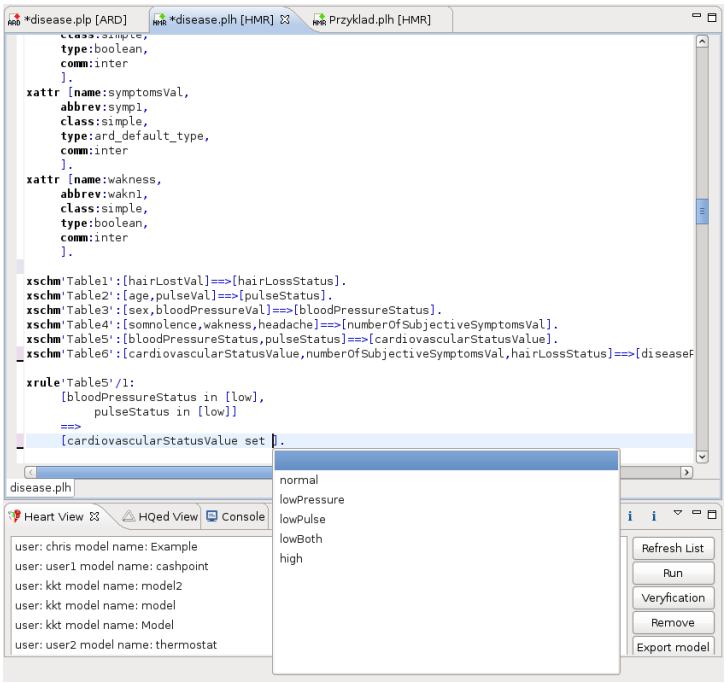


Fig. 3. HMR Editor

3. Nalepa, G.J., Ligęza, A., Kaczor, K.: Formalization and modeling of rules using the XTT2 method. International Journal on Artificial Intelligence Tools **20**(6) (2011) 1107–1125
4. Ligęza, A.: Logical Foundations for Rule-Based Systems. Springer-Verlag, Berlin, Heidelberg (2006)
5. Kaczor, K., Nalepa, G.J.: HaDEs – presentation of the HeKatE design environment. In Baumeister, J., Nalepa, G.J., eds.: 5th Workshop on Knowledge Engineering and Software Engineering (KESE2009) at the 32nd German conference on Artificial Intelligence: September 15, 2009, Paderborn, Germany, Paderborn, Germany (2009) 57–62
6. Bator, P.: Projekt i implementacja narzędzi do edycji wiedzy regułowej HeKatE na platformie Eclipse. Master's thesis, AGH University of Science and Technology (2012) supervisor: Grzegorz J. Nalepa.
7. Nalepa, G.J., Ligęza, A.: HeKatE methodology, hybrid engineering of intelligent systems. International Journal of Applied Mathematics and Computer Science **20**(1) (March 2010) 35–53
8. 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
9. Grzegorz J. Nalepa, K.K.: Proposal of a rule-based testing framework for the automation of the unit testing process. In: Proceedings of the 17th IEEE International Conference on Emerging Technologies and Factory Automation ETFA 2012, Kraków, Poland, 28 September 2012. (2012)