

**4th Workshop on
Knowledge Engineering
and Software Engineering (KESE2008)**
at the 31st German Conference on Artificial Intelligence

Grzegorz J. Nalepa and Joachim Baumeister (Editors)

September 23, 2008, Kaiserslautern, Germany

Preface

Artificial Intelligence (AI) has always been both a challenging and exciting field. Knowledge Engineering remains one of the main areas of research and development of Artificial Intelligence, penetrating and finding applications in specific fields of Computer Science. Research on Knowledge Engineering and Software Engineering have exchanged very fruitfully during the last decades.

Both consider the development of advanced tools and processes for building complex intelligent systems. Especially, declarative Software Engineering techniques such as knowledge-based systems, logic programming, constraint programming, have been established in various areas of Software Engineering, and in the context of the Semantic Web. Knowledge Engineering extensively uses current software tools, environments and languages for the development of intelligent systems. Some recent developments within the Business Rules community brought classic Knowledge Engineering even closer to the development of business-oriented software systems.

The 4th workshop on Knowledge Engineering and Software Engineering (KESE2008), held with 31st German Conference on Artificial Intelligence (KI2008), brought together both researchers and practitioners from the fields of Software Engineering and applied Artificial Intelligence. Latest research results as well as practical experience in the field was exchanged during the workshop. The topics of interest included the following:

- Application of AI and Knowledge Engineering methods in Software Engineering, such as knowledge and experience management, declarative, logic-based approaches, constraint programming, agent-oriented software engineering, issues of maintenance, and Business Rules.
- The use of Software Engineering tools in AI and Knowledge Engineering, e.g. engineering the Semantic Web, database and knowledge base management in AI systems, tools for intelligent systems, evaluation of intelligent systems, and process models.

This volume contains nine contributions presented at KESE2008, seven regular and two short papers. Diaconescu et al. discuss mapping ERDF(S) to Take inference engine vocabulary, and extending Take to support Open World Assumption with strong and weak negation predicates. Giurca and Pascalau introduce a JSON-based rule language and its JavaScript-based rule engine towards providing Web 2.0 applications with rule-based inference capabilities. Kluegl et al. introduce a system that uses a rule representation for information extraction tasks. Moreover they introduce a test-driven development process to engineer such rules. Nalepa and Kluza consider practical issues concerning the use of UML as a knowledge representation method for XTT rules, with an ultimate goal of combining the classic AI approach to knowledge-based systems design with Software Engineering modeling methods. Nalepa and Wojnicki propose a Knowledge Wiki System where knowledge is expressed in the Prolog language and an inference engine is coupled with the Wiki, providing means for an automated knowledge processing and interpretation. Newo and Althoff present a

model for simulating and researching people's behaviors in critical situations. The model is implemented by means of a multiagent system approach, realized by distributed knowledge-based systems with a specific focus on case-based reasoning technology. Peylo considers requirements engineering and argues that it is feasible to establish a domain ontology based on meta information and explanations that are represented as scripts. Moreover, he shows that this ontology has to be constructed in a dynamic way, to reflect the dynamics of requirements engineering process. Reichle and Bach present the retrieval and adaptation mechanisms used in an information system on travel medicine, docQuery. The retrieval method's main feature is an overall improved accuracy of retrieval results' similarities through a more diverse distribution of similarities over the retrieved result sets. Finally Ruh and Stolzenburg discuss a method for engineering and programming multi-robot systems, based on a combination of statecharts and hybrid automata. The formal specification method they use allows for a graphical presentation of the whole multiagent system behavior, and the specification can be directly executed on mobile robots.

The organizers would like to thank all who contributed to the success of the workshop: for the technical program 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 quality of the initial submissions. The reviewing process as well as proceeding preparation was performed with the use of EasyChair, for which the organizers thank Andrei Voronkov, the developer of the system. We would also like to thank Thomas Roth-Berghofer the KI2008 workshops chair for his efforts and support.

September 2008

Grzegorz J. Nalepa and Joachim Baumeister
Workshop Organizers
KESE2008

Workshop Organization

The 4th Workshop on Knowledge Engineering and Software Engineering
(KESE2008)

was held as a one-day event at the
31st German Conference on Artificial Intelligence (KI2008)
on September 23, 2008 in Kaiserslautern, Germany.

Workshop Chairs and Organizers

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

Programme Committee

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

External Reviewers

Kerstin Bach, University Hildesheim, Germany
Jochen Reutelshoefer, University Würzburg, Germany

Table of Contents

Towards a Mapping from ERDF(S) to Take Vocabulary	1
<i>Ion-Mircea Diaconescu, Adrian Giurca, Gerd Wagner, Jens Dietrich</i>	
JSON Rules	7
<i>Adrian Giurca, Emilian Pascalau</i>	
Test-Driven Development of Complex Information Extraction Systems using TextMarker	19
<i>Peter Kluegl, Martin Atzmueller, Frank Puppe</i>	
UML Representation Proposal for XTT Rule Design Method	31
<i>Grzegorz Nalepa, Krzysztof Kluza</i>	
Proposal of a Prolog-based Knowledge Wiki	43
<i>Grzegorz Nalepa, Igor Wojnicki</i>	
Learning to Cope with Critical Situations – An Agent based Approach . .	55
<i>Régis Newo, Klaus-Dieter Althoff</i>	
On Restaurants and Requirements: How Requirements Engineering may be Facilitated by Scripts	61
<i>Christoph Peylo</i>	
Improving Result Adaptation through 2-step Retrieval	73
<i>Meike Reichle, Kerstin Bach</i>	
Translating Cooperative Strategies for Robot Behavior	85
<i>Florian Ruh, Frieder Stolzenburg</i>	
Author Index	97

Towards a Mapping from ERDF(S) to Take Vocabulary

Ion-Mircea Diaconescu¹, Adrian Giurca¹, Gerd Wagner¹, Jens Dietrich²
E-Mail: {M.Diaconescu, Giurca, G.Wagner}@tu-cottbus.de,
J.B.Dietrich@massey.ac.nz

¹ Brandenburg University of Technology, Germany

² Massey University, New Zealand

Abstract. This paper presents a mapping solution from ERDF(S) to Take vocabulary. The work is related to an investigation of integrating ERDF Derivation Rules into Take inference engine. Some steps are required to finalize this task: a mapping between ERDF(S) and Take vocabulary, integration of ERDF knowledge base in Take, and empowering Take inference engine to deal with both Closed World Assumption and Open World Assumption, specifically with Open and Closed predicates and two types of negation: strong negation and weak negation.

1 Introduction and Motivation

The paper presents an approach of a mapping from ERDF(S)[3, 4, 2] to Take [6] vocabulary. This mapping is part of our project to extend Take with support for ERDF Derivation Rules.

Influenced by Mandarax [8], Take is a backward inference engine, designed to deal with objects as facts. In the actual implementation, Java objects are used. The engine use a polymorphic negation: (1) *strong negation* for `JPredicate` and `PropertyPredicates` - predicates generated for boolean methods respectively boolean properties from Java Beans. Those are computed without using rules. (2) *negation as failure* for `SimplePredicate` - predicates defined by rules and used in rules, facts, queries or external fact stores. Take compiles rules to an optimized Java code before the inference process is started, this having as primary advantage a better scalability.

Since explicit negative information is not provided (with exception of Boolean properties), and the inference is under *CWA* hypothesis, false information can be inferred in some cases. For instance, reasoning on top of a FOAF³ facts base, a rule set can conclude that two persons does not `foaf:knows` one each other since no occurrence of a `foaf:knows` property is found in at least one FOAF file of the two persons refereing the other person (*CWA*). Since it is not mandatory to refer all known persons in your FOAF file (not relating a known person in your FOAF file does not means that you don't know him), the derived conclusion

³ FOAF Specification - <http://xmlns.com/foaf/spec/>

might be not appropriate. In this example, it is obvious that `foaf:knows` can be represented as an *Open Property*.

Based on Partial Logic [7], ERDF comes to solve this problem by providing the possibility of expressing negated facts, and two types of negation in rules: *strong negation* and *weak negation*. Also it defines new types of classes and properties. In [9] detailed information and use cases are provided about those extensions and the rule language.

We are not aware of any substantial work regarding the mapping from ERDF (and implicitly RDF(S)[5]) vocabularies to Java classes. Such a mapping is a key point towards mapping RDF/ERDF descriptions to Java objects. The next paper section deals with this issue tailored to the Take inference engine mechanism.

2 Mapping ERDF Schemas to Take Vocabulary

2.1 Resolving URI's to Java Identifiers

Since in ERDF Schemas classes and properties are defined by URI's, first step is to define a mapping from URI's to Java identifiers. The following rules define this mapping:

1. Using MD5⁴ hashes, each URI is mapped to an unique ID. It will be prefixed with an '_' (underscore), since Java identifiers cannot start with digits.
2. Optional, at the end of the identifier the local name (where is possible) is added. Additional, comments may be generated to improve readability.
3. The reversibility is obtained by storing each URI expressing a class or a property and its generated Java identifier.

Example 1. Mapping URI's to Java identifiers

```
http://example.org/voc/Person ⇒ _35EDA2C34D57C09DCDB4D1544C674779_Person
http://example.org/v/Person ⇒ _DD3A8FFF3850417783DC9651D86B0395_Person
```

Using MD5 hashes, for each different URI a different signature is obtained. Two or many URI's may express the same concept (such as *Person* from the above example), but since different URI's are used, a distinct class or property is generated for each of them. The mapping from Java identifiers to ERDF classes/properties is implemented by using a Java `Map`.

2.2 Mapping Classes and Properties

This section describes the mapping from ERDF classes and properties to Take vocabulary. The following rule expresses the general mapping from ERDF vocabulary to Java vocabulary:

⁴ The MD5 Message-Digest Algorithm - <http://www.ietf.org/rfc/rfc1321.txt>

Rule 1 (a) For each ERDF class a Java class is generated; (b) Each ERDF property maps to a Java property.

Example 2. Mapping classes and properties from ERDF(S) to Java Vocabulary

```
<erdf:Class erdf:about="http://example.org/v/Person"/>
<rdf:Property rdf:about="http://example.org/v/name">
  <rdfs:domain rdf:resource="http://example.org/v/Person" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
</rdf:Property>

// Class related URI: http://example.org/v/Person
public class _DD3A8FFF3850417783DC9651D86B0395_Person {
  // Property related URI: http://example.org/voc/name
  private Collection<String> _4D9F16067649DF5A90773F7D832D9122_name;

  public Collection<String> get_4D9F16067649DF5A90773F7D832D9122_name() {
    return this._4D9F16067649DF5A90773F7D832D9122_name;
  }
}
```

Each ERDF property maps in Java as a collection containing all values of that property. This allows to express multi-valued ERDF properties. Accessors are generated for each property. Adding an element to collection must be done by checking some constraints depending by the type of the added value (datatype or object type).

2.3 Solving rdfs:range, rdfs:domain and rdfs:subClassOf relations

In ERDF multiple ranges are allowed for a property. Types are mapped according with their nature: (1) *datatype* mappings and (2) *object type* mappings.

Datatypes If all property ranges are datatypes then, according with RDF Semantics [1], the property type is the intersection of all those types. For compatible datatypes this intersection is defined, for all the rest the intersection will be an empty set and therefore the property will not be generated and an exception is thrown. Only datatypes defined by XML Schema datatypes⁵ are allowed. A mapping from XML datatypes to Java types is defined⁶. All datatypes in Java will be expressed by using corresponding wrapper classes (e.g. `Integer` for `int`) and collections representing properties are parameterized with the corresponding wrapper class. The RDF datatype, `rdf:XMLLiteral`, maps to `String` type in Java.

Example 3. Mapping ranges for datatypes

```
<rdf:Property rdf:about="http://example.org/v/salary">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#float" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#double" />
</rdf:Property>
```

⁵ XML Schema Datatypes - <http://www.w3.org/TR/xmlschema-2/>

⁶ XML datatypes mapping - <http://java.sun.com/javase/5/docs/tutorial/doc/bnazq.html>

```

    <rdfs:domain rdf:resource="http://example.org/v/Employee" />
  </rdf:Property>

  public class _f2989a52fddb17fa72653625cd9c0374_Employee {
    private Collection<Integer> _93ad619912a976c6c56623d3c6b73491_salary;

    public Collection<Integer> get_93ad619912a976c6c56623d3c6b73491_salary() {
      return this._93ad619912a976c6c56623d3c6b73491_salary;
    }
  }
}

```

Since the Take engine manages itself properties having type `Collection`, we just have to define a `get` method which returns the collection. Before adding new values for a specific property, some constraints have to be verified (e.g. do not have the same value many times). An implementation of `Collection` (i.e. extending `ArrayList` by overriding the `add` method) is provided.

Example 4. Implementing `DatatypeArrayList`

```

  public class DatatypeArrayList extends ArrayList<Object> {
    public boolean add(Object value) {
      if(!this.contains(value)) {super.add(value); return true;}
      return false;
    }
  }
}

```

Object types All types which are not XML datatypes are considered object types. In this case, a proper intersection solution cannot be defined using an automatic method. When a single *range* is defined for a ERDF property, the collection will be parameterized with that type. If many ranges are used for the same property, then the collection representing the property is parameterized with the Java super type `Object`. The intersection of ranges is resolved at runtime by the `add` method, checking if the value wanted to be added is instance of all types expressed by its `rdfs:range` occurrences. For object types, we define a new `Collection` implementation, `ObjectArrayList`, which will be used to instantiate all object properties from our Java classes.

Example 5. Implementing `ObjectArrayList`

```

  public class ObjectArrayList extends ArrayList<Object>{
    private ArrayList<String> types;

    public ObjectArrayList(ArrayList<String> types) {this.types = types;}

    public boolean add(Object value) {
      boolean valid = true;
      for(int i=0;i<types.size();i++) {
        try {
          valid = valid && (Class.forName(types.get(i)).isInstance(value));
        }catch(Exception e){System.out.println(e.toString());}
      }
      if(valid) {super.add(value); return true;}
      return false;
    }
  }
}

```

Mapping rdfs:domain Unlike in Java, ERDF properties are global and applies to all classes defined as their domains. The proposed mapping creates a property with the same name in each class appearing in their `rdfs:domain` occurrences.

Example 6. Mapping properties having multiple domains

```

<erdf:Class erdf:about="http://example.org/v/Person" />
<erdf:Class erdf:about="http://example.org/v/City" />
<rdf:Property rdf:about="http://example.org/v/name" />
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
  <rdfs:domain rdf:resource="http://example.org/v/Person" />
  <rdfs:domain rdf:resource="http://example.org/v/City" />
</rdf:Property>

public class _dd3a8fff3850417783dc9651d86b0395_Person {
  private Collection<String> _ba6a578397a5a376500712b8cb11e9e9_name;
}
public class _0545f96c913977fbf95b0c52de54d6af_City {
  private Collection<String> _ba6a578397a5a376500712b8cb11e9e9_name;
}

```

Mapping rdfs:subClassOf Since Java does not allows multiple inheritance, but ERDF supports that by using `rdfs:subClassOf` property, we need to emulate multiple inheritance by using Java interfaces. The following rule defines the multiple inheritance approach:

Rule 2 (a) For each class implied in an inheritance chain, an empty interface is generated; (b) Each class from the inheritance chain must implements all interfaces corresponding to its superclasses; (c) Each class implied in the inheritance chain contain a copy of all properties (and corresponding accessors and comments) of its superclasses.

This methodology allows the usage of the Java operator `instanceof` for emulating multiple inheritance. All generated interfaces does not contains methods and are used only for the above purpose.

2.4 Expressing Closed, Open and Partial Properties and classes

ERDF specializes its classes and properties as *closed*, *open* or *partial*. Such a classification is important during the inference process (OWA or CWA may applies depending on the classification). The mapping uses Java annotations as a solution for classifying classes and properties. The following annotations are defined: (1) `@closed` - meaning *closed* class and/or property; (2) `@open` - to express *open* class and/or property; (3) `@partial` - denoting *partial* class and/or property. The default annotation is `@open`. Using Java annotation provides us an elegant solution for identifying types of classes and properties.

Example 7. Annotating properties

```

<erdf:Class rdf:about="http://example.org/v/Person" />
<erdf:ClosedProperty rdf:about="http://example.org/v/authorOf" />
  <rdfs:range rdf:resource="http://example.org/v/Publication" />

```

```

    <rdfs:domain rdf:resource="http://example.org/v/Person" />
  </erdf:ClosedProperty>

  public class _dd3a8fff3850417783dc9651d86b0395_Person {
    @closed
    private Collection<_64742878a633276917290732b0301d3b_Publication>
      _5470da641d7738e11e2e1ef15e3c23a8_author0f;
  }

```

3 Conclusion and Future work

The paper provides a mapping solution from ERDF Schemas to Take vocabulary. Some issues such as the intersection of ERDF property ranges are considered and a concrete distinction between datatypes and object types is made.

Further work includes empowering Take engine with support for reasoning in both *OWA* and *CWA* taking in account classes and properties types and considering two kinds of negation, namely *strong negation* and *negation as failure*.

References

1. RDF Semantics. W3C Recommendation 10 February 2004. <http://www.w3.org/TR/rdf-mt/>.
2. A. Analyti, G. Antoniou, C. V. Damasio, and G. Wagner. Extended RDF as a Semantic Foundation of Rule Markup Languages. *Journal of Artificial Intelligence Research*, 32:37–94, 2008.
3. Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damasio, and Gerd Wagner. Negation and Negative Information in the W3C Resource Description Framework. *Annals of Mathematics, Computing and Teleinformatics*, 1(2):25–34, 2004.
4. Anastasia Analyti, Grigoris Antoniou, Carlos Viegas Damasio, and Gerd Wagner. Stable Model Theory for Extended RDF Ontologies. In Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen, editors, *Proceedings of the 4th International Semantic Web Conference*, volume 3729 of *Lecture Notes in Computer Science (LNCS)*, pages 21–36, Galway, Ireland, 6-10 November 2005. Springer-Verlag.
5. D. Brickley and R.V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation February 2004. <http://www.w3.org/TR/rdf-schema/>.
6. Jens Dietrich, Jochen Hiller, and Bastian Schenke. Take - A Rule Compiler for Derivation Rules. In *Proceedings of the International RuleML Symposium on Rule Interchange and Applications (RuleML-2007)*, volume 4824 of *Lecture Notes in Computer Science (LNCS)*. Springer, 2007.
7. Heinrich Herre, Jan O. M. Jaspars, and Gerd Wagner. Partial Logics with Two Kinds of Negation as a Foundation for Knowledge-Based Reasoning. In D.M. Gabbay and H. Wansing, editors, *What is Negation?* Kluwer Academic Publishers, 1999.
8. Mandarax. project website. <http://www.mandarax.org/>.
9. Gerd Wagner, Adrian Giurca, Ion-Mircea Diaconescu, Grigoris Antoniou, and Carlos Viegas Damasio. ERDF Implementation and Evaluation. Technical report, March 2008.

JSON Rules

Adrian Giurca¹ and Emilian Pascalau¹

Brandenburg University of Technology, Germany
{giurca,pascalau}@tu-cottbus.de

Abstract. This paper presents a JSON based rule language and its JavaScript-based rule engine towards providing Web 2.0 applications with rule-based inference capabilities. For interoperability purposes the Rule Interchange Format is used. While the rule engine is enough general, its main purpose is to execute production rules and Event-Condition-Action rules related to the web page DOM processing. This way the user's browsing experience will be enriched with the ability to modify on the fly the DOM of the current document as well as the browser user interface (Firefox).

1 Introduction

In the last 10 years business rules were employed to declaratively describe policies, business processes and practices of an enterprise. Applications in domains such as insurance, financial services, government, telecom, and e-commerce benefit greatly from using rule engines. Moreover, rules are becoming increasingly important in business modeling and requirements engineering, as well as in Semantic Web applications. In each of these fields different rule languages and tools are being used. At the same time the amount of Web 2.0 applications increases heavily. Actual technologies such as Asynchronous JavaScript and XML (AJAX) [8] allows the development of Rich Internet Applications (RIAs). This concept was introduced in [2] to denote a web application that typically runs in a web browser, and do not require software installation. Several Web 2.0 applications use heavily AJAX in order to provide desktop-like behavior to the user. The number of RIAs is increasing because of the broad bandwidth of today's Internet connections, as well as the availability of powerful and cheap personal computers. However, traditional ways of programming Internet applications no longer meet the demands of modern rule-enabled rich Internet applications. For example a highly responsive Web 2.0 application such as Gmail¹, might be much easily customized by using a declarative description such as rules.

The goal of this paper is to describe a rule language and a client-side rule engine. The rule language uses JavaScript Object notation(JSON) notation [5] as its main format. However, for interoperability purposes the Rule Interchange Format (RIF) [4] is used. The choice of using JSON is due to it's widely usage by JavaScript developers. JSON is used for rule descriptions as well as serialization

¹ <http://mail.google.com>

for data that is going to be transmitted over network. While the rule engine is enough general, its main purpose is to execute production rules and Event-Condition-Action rules related to the web page Document Object Model (DOM) processing. This way the user's browsing experience will be enriched with the ability to modify on the fly the DOM of the current document as well as the browser user interface (Firefox).

2 Related Work

While the ideas of RIAs are not new (see [2]) the rule-based RIAs proposals are quite recent. A project was started in May 2008 by Project 6 Research ². However, the goals of this project are limited to XPath processing i.e. rules conditions are similar with test from XSLT while the actions are not clearly specified. In overall, the concepts are far away to be clear and we did not see to much advance. Also this product is commercial and no demos are available. There are also concerns to emulate a rule parser in Adobe Flex framework³ but the goal seems to be a client side Drools[10] parser.

The most advanced work seems to be in [12] (May 2008) where two-layer architecture for rule-enabled RIAs is described. This paper is a good starting point but as a general architecture document, it does not formally provide a Model-Driven Architecture like, platform independent model. In addition the paper is not focused on the rule language description neither to the client-side rule execution. This work was also related in [11].

3 The Rule Language

JSON notation combined with JavaScript function calls offers large capabilities to express various kinds of rules. Recall that we deal both with production rules and with Event-Condition-Action (ECA) rules i.e. rules of the form

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

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

The metamodel of a JSON Rule is depicted in Figure 1.

Example 1 (Production Rule).

For all elements of class 'note' having as first child a 'ul' change the first child background color to blue. Expressed in a logical form the above example looks like: $\forall x \exists y (Element(x) \wedge x.class = 'note' \wedge y = x.firstChild \wedge y.nodeName = 'ul') \rightarrow changeBackground(y, 'blue')$

² <http://www.p6r.com/articles/2008/05/22/an-xpath-enabled-rule-engine/>

³ <http://archives.devshed.com/forums/compiler-129/writing-a-rules-parser-in-actionscript-javascript-2370024.html>

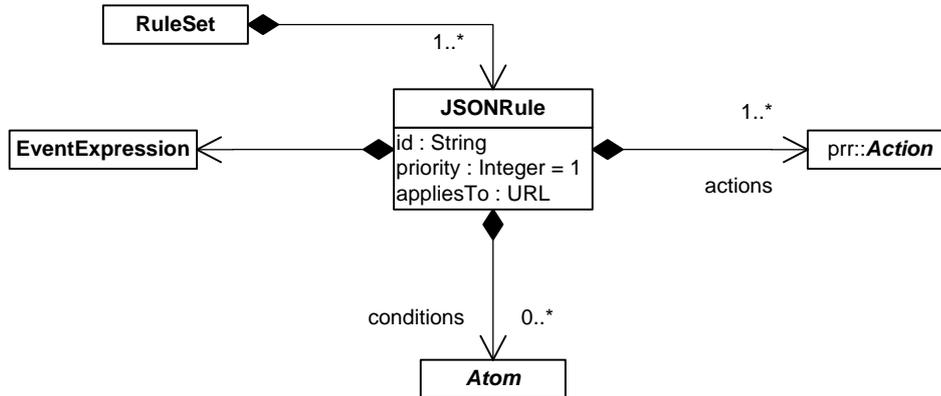


Fig. 1. Rules and Rulesets

```

{"id":"rule101",
  "appliesTo": ["http://www.example.org/JRules",
               "http://www.google.com/"],
  "condition": "$X:Element( class=='note',
                           $Y:firstChild)
               &&
               ($Y.nodeName == 'ul')",
  "actions":["changeBackground($Y, 'blue')"]
}
  
```

The above example shows that a JSON rule is first of all, a JSON object. The `appliesTo` property states that the rule will apply only on the specific indicated pages (URLs).

The `condition` uses a Drools like syntax and state that all elements with the class attribute equals with `note` (i.e. `$X:Element(class=='note')`) and with the first child an unsorted list (i.e. `$Y.nodeName == 'ul'`) must participate in the action.

The action `changeBackground($X, 'blue')` should be an available user-defined function call. If no such function is available then no action is performed.

JSON Rules are also ECA Rules i.e. they are triggered by events (see Example 4). Below we provide descriptions of the rule constituents.

3.1 Condition

A rule condition is always a conjunction of atoms. Empty conditions are interpreted as true conditions. As can be seen in the Figure 2, the language supports three kinds of atoms: *JavaScriptBooleanCondition*, *Description* and *XPathCondition*. The reader should notice that future extensions may involve other kinds of atoms.

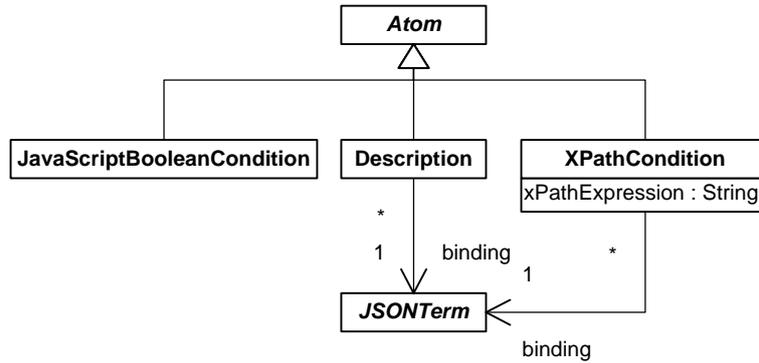


Fig. 2. The language atoms

JavaScriptBooleanCondition This is the simplest conditional atom. Any JavaScript boolean expression is allowed. For example `window.find('rule')` or `document.getElementById('id').value==10` are allowed.

Description This atom is related to the syntax of Drools pattern conditional. The metamodel of language descriptions is depicted in Figure 3.

A *Description* is bound to a *JSONTerm*, has a *type* and has a list of *constraints*. The *type* is one of the values described by *DescriptionType* enumeration. These values correspond to the node types defined in DOM Level 2 Core specification⁴.

The *Description* offers two types of constraints *PropertyRestriction* and *PropertyBinding*.

- A *PropertyRestriction* (see Figure 3) describes a set of value restrictions to properties of the *JSONTerm* that are bound to it.
 - The string *property* encodes a property name of a property belonging to the corresponding bounded *JSONTerm*.
 - *operator* is relational one.
 - The *value* is either a *JSONTerm* (*Variable* or *DOM:Node*), or a *RegularExpression*, or a *String* or a *Number*.
- A *PropertyBinding* performs a variable binding of a property belonging to the related *JSONTerm*. After that this variable becomes available at the rule level (See example 1).

The condition below stands for all DOM entities of type *Element* that are text input elements with the value date of the form *yyyy-mm-dd*. Notice that for the value we used a regular expression that checks its format. Month can be only between 01-12. The regular value for day can not be greater than 31, and for month 02 the value for day can not be greater than 29.

⁴ <http://www.w3.org/TR/DOM-Level-2-Core/idl-definitions.html>

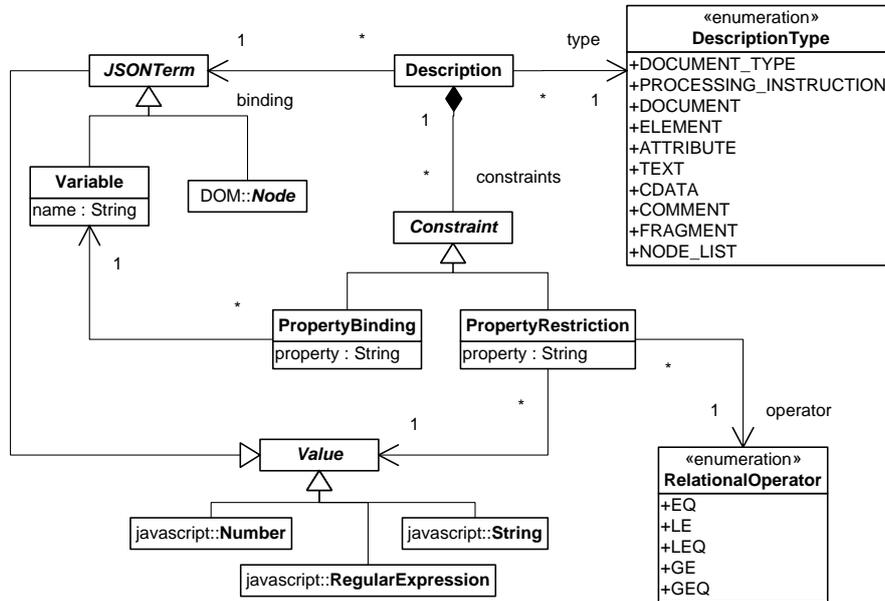


Fig. 3. Descriptions

Example 2 (Condition).

```

$X : Element(nodeName=="input", type=="text",
value=="match(^[0-9]{4}-(((0[13578]|(10|12))-(0[1-9]|[1-2][0-9]|3[0-1]))|(02-(0[1-9]|[1-2][0-9]))|((0[469]|11)-(0[1-9]|1-2)[0-9]|30)))$)")
  
```

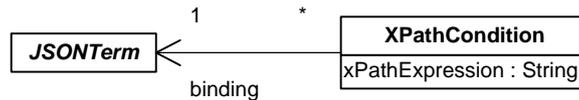


Fig. 4. The XPath Condition

XPathCondition The *XPathCondition* (see the metamodel in Figure 4) is the third type of conditional atom. This one compared with the other might be a little peculiar. For a usage example of this type of conditional we are going to use the example page below.

Example 3 (An XPath Condition). Consider the following page with the view depicted in Figure 5:

T1:row 1, cell 1	T1:row 1, cell 2
T1:row 2, cell 1	T1:row 2, cell 2
T2:row 1, cell 1	T2:row 1, cell 2
T2:row 2, cell 1	T2:row 2, cell 2

Fig. 5. Page view for XPathCondition example

```

<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
  <head>
    <title>XPathCondition</title>
  </head>
  <body>
    <div id="_div1" >
      <table border="1">
        <tr> <td>T1:row 1, cell 1</td> <td>T1:row 1, cell 2</td> </tr>
        <tr> <td>T1:row 2, cell 1</td> <td>T1:row 2, cell 2</td> </tr>
      </table>
    </div>
    <div id="_div2">
      <table border="1">
        <tr> <td>T2:row 1, cell 1</td> <td>T2:row 1, cell2 </td> </tr>
        <tr> <td>T2:row 2, cell 1</td> <td>T2:row 2, cell 2</td> </tr>
      </table>
    </div>
  </body>
</html>

```

Such a conditional can be bound to a *JSONTerm* (Figure 3. Recall that a *JSONTerm* is either a *Variable* or a *DOM:Node*.

Assuming that we want to change the background for all rows in all tables in the DOM of the current page, using the *XPathCondition*, the condition is:

```
"$X in "html//table//tr"
```

The variable *\$X* has been used along with the reserved word *in*. The meaning is "forall *\$X* in the collection..." The evaluation of the *xPathExpression* returns a list of nodes (In the previous example, the evaluation of the *xPathExpression* returns 4 nodes).

On the other hand, if we want to change the background only for a specific node by using an *XPathCondition* the condition is:

```

{"nodeName":"tr",
  "firstChild":{"nodeName":"td",
                "textContent":"T2:row1, cell 1"
              }
} in "html//table//tr"

```

After evaluating the condition the background should be changed only for the first row of the second table (see Figure 5).

3.2 Actions

Our approach deals with standard actions of OMG Production Rule Representation (PRR), [9]. The reader should notice that any user-defined JavaScript functions can be called in the rule actions part. Below is the mapping of the PRR standard actions to our language:

PRR Standard Actions	JSON Rules
AssignExp	change properties of an element
InvokeExp	any JavaScript function call
AssertExp	insert a DOM node
RetractExp	remove a DOM node
UpdateExp	update a DOM node

Table 1. PRR Standard Actions and their representation

An *invoke action* is already provided in Example 1. It corresponds to a JavaScript function-call. The function must be available otherwise the action is ignored (not executed).

An *assign action* is usually intended to change the properties of an element. For example

```
$X.setAttribute("value", "25")
```

is an assign action changing the `value` attribute of an `input` element bounded to the variable `$X`. If `$X` is not bounded to an element allowing the attribute `value` then the engine will ignore such action.

An *assert action* is related to the creation of new nodes in the DOM e.g.

```
$X.appendChild(document.createElement("input"))
```

is an assert action.

A *retract action* is the inverse of the assert action i.e. deletes a node from the DOM.

An *update action* is usually related to the content update of a DOM node. For example

```
$X.replaceChild($Y,$Z)
```

is an assert action.

3.3 Event Expressions

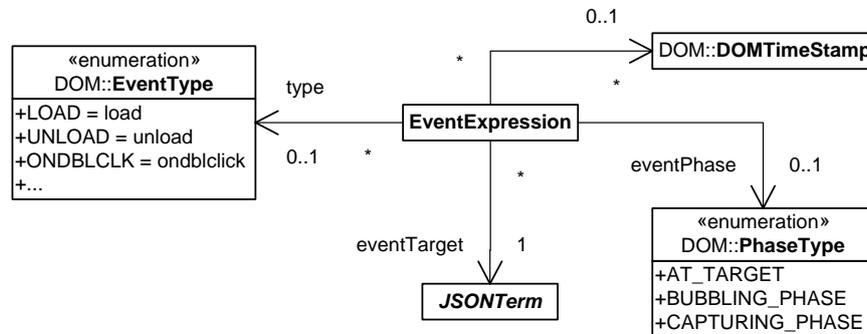


Fig. 6. Event Expressions

The JSON event expression is related to the *Event* interface specification in DOM Level 2 Events⁵, therefore the properties of this expression have the same meaning as in the *Event* specification. At the runtime these properties of this expression are matched against the incoming DOM events and their values can be processed in the rule conditions and actions.

Example 4 (ECA Rule).

```

{"id": "rule102",
  "appliesTo": ["http://mail.yahoo.com/"],
  "eventExpression": { "eventType": "click",
                      "eventTarget": "$X"
                    },
  "condition": " ($X.nodeName == 'a',
                $X.href == 'match(showMessage?fId=Inbox)')",
  "actions": ["append($X.textContent)"]
}
  
```

The rule from example 4 concerns the Yahoo mail and states that when click event is raised, if the event came from an **a** element, and if the **href** property matches the regular expression (rudimentary check that the link is an inbox Yahoo message link) then call **append** function with the message subject as parameter.

3.4 Additional parameters

In addition to its main constituents a rule provides some other parameters:

⁵ <http://www.w3.org/TR/DOM-Level-2-Events/>

- The *id* is required and denotes the unique identifier of a rule.
- The *appliesTo* property is required and holds a list of URLs on which the rule must apply e.g. the rule from Example 1 can be applied to the pages <http://www.example.org/JRules> and to <http://www.google.com/>.
- *Priority* expresses the order of a rule in a ruleset. If no value is provided for it, default value is "1". Based on priorities the rule engine must execute the rules in a down counting order (from greater values to lower values). The execution order is not relevant for rules having the same priority.

4 The Rule Engine

The main characteristics of the rule engine are:

- Is a forward chaining rule engine using a modified RETE algorithm (see [6] for the standard version) for production rules ;
- Uses the above rule language as well as RIF XML.
- Deals with two different types of rules: production rules and ECA Rules
- DOM events are processed as atomic events (i.e. no duration).
- Rules are stored locally or remote or both.
- The engine execute rulesets (a ruleset is the set of all rules referring to a specific URL).
- The RETE working memory is the document DOM itself. Rule property restrictions are matched against DOM entities (such as elements processing instructions, attributes an so on).

The component view of the engine is depicted in the Figure 7.

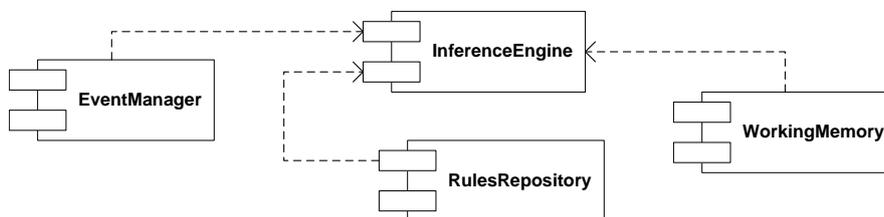


Fig. 7. Rule Engine

A UML state diagram describing the functionality of the engine is depicted in the Figure 8.

When the *InferenceEngine* is started it loads the corresponding rules from the *RuleRepository*. After the rules are loaded, the *EventManager* gets active and it listens for events from the *WorkingMemory*. When it receives events from the *WorkingMemory*, the *EventManager* informs the *InferenceEngine* about it. The *InferenceEngine* computes rules to fire. If rules are found then it fires them, and the *WorkingMemory* is changed. When no rules are computed then it stops.

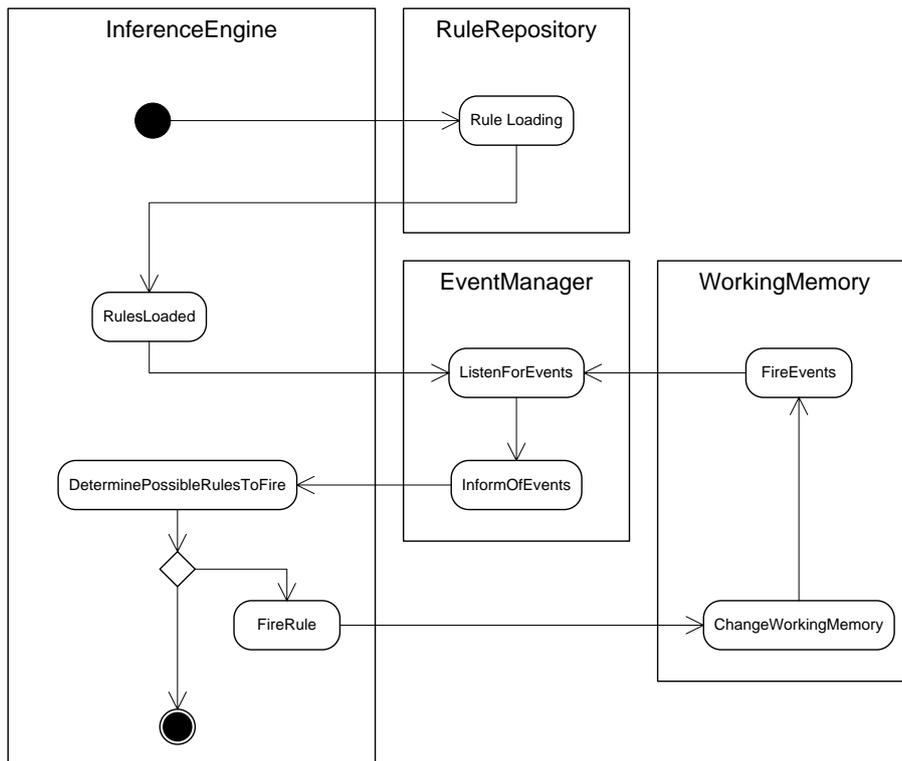


Fig. 8. The Rule Engine State Diagram

Opposed to usual rule engines, that stop their activity when no rules can be fired, this stops only when the WorkingMemory exists no more. When rules can not be fired, but the WorkingMemory still exists, the engine gets into an idle state, and waits to be informed about new events from the EventManager.

4.1 The Working Memory

Is represented by the DOM of the current page. The WorkingMemory is special because it is event based, DOM is changed through events.

4.2 The Inference Engine

Is the "brain" of the system. Based on the facts (DOM) of the Working Memory and based on the current page corresponding ruleset, it performs the matching operation and executes rules.

4.3 The Event Manager

The event manager is a combination between JavaScript and the Working Memory - DOM of the current page. The Event Manager takes advantage of the fact that the DOM itself is event based. All changes and in general all DOM events are reported to the main document object. This is based on the bubbling effect of DOM events.

4.4 The Rule Repository

Rules are stored in the repository. The repository can be local or remote. In both cases the storing language is JSON based as described in Section 3.

5 Conclusion and future work

This paper describes an approach of enriching RIAs with rule-based reasoning. JSON Rules provides the JavaScript engine with reasoning capabilities and the users can write their own rules. The rules syntax is based on JSON notation, therefore does not require high effort to accommodate it. Rules are simpler but powerful, their main goal being to change the DOM of the page they apply. The rule actions comply with the proposed OMG standard for production rules and are enough general to achieve all kind of DOM changes as well as any kind of side effects. The next immediate step will take into account the engine interaction with both user defined events and *XMLHttpRequest* events to increase the power of the reaction rules to dynamically handle the page AJAX-based interactions. On a medium term JSON Rules should deal with metadata (with emphasis on RDF[7]) both for embedded metadata (such as RDFa [1]) and external metadata (such as RSS [3] and Atom[13]). In addition, sharing rules is a feature that can improve user experience and also, might spare him the time

of writing rules by himself, in the case he can find rules for the page being interested in. The user can share with his friend, or with everybody. Rule sharing goes hand in hand with rule publishing.

References

1. Ben Adida and Mark Birbeck. RDFa Primer: Embedding Structured Data in Web Pages. W3C Working Draft 17 March 2008. <http://www.w3.org/TR/xhtml-rdfa-primer/>.
2. Jeremy Allaire. Macromedia Flash MXA next-generation rich client. <http://www.adobe.com/devnet/flash/whitepapers/richclient.pdf>, March 2002.
3. RSS Advisory Board. RSS 2.0 Specification. <http://www.rssboard.org/rss-specification>.
4. Harold Boley and Michael Kifer. RIF Basic Logic Dialect. <http://www.w3.org/2005/rules/wiki/BLD>, October 2007.
5. Douglas Crockford. The application/json Media Type for JavaScript Object Notation (JSON). <http://tools.ietf.org/html/rfc4627>, July 2006.
6. Charles Forgy. Rete – A Fast Algorithm for the Many Pattern / Many Object Pattern Match Problem. *Artificial Intelligence*, 19:17–37, 1982.
7. Klyne G. and Carroll J.J. Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. <http://www.w3.org/TR/rdf-concepts/>.
8. Jesse James Garrett. Ajax: A new approach to web applications. <http://www.adaptivepath.com/ideas/essays/archives/000385.php>, February 2005.
9. OMG. Production Rule Representation (PRR), Beta 1. Technical report, November 2007.
10. Mark Proctor, Michael Neale, Michael Frandsen, Sam Griffith Jr., Edson Tirelli, Fernando Meyer, and Kris Verlaenen. Drools 4.0.7. http://downloads.jboss.com/drools/docs/4.0.7.19894.GA/html_single/index.html.
11. Kay-Uwe Schmidt, Jörg Dörflinger, Tirdad Rahmani, Mehdi Sahbi, Susan Thomas, and Ljiljana Stojanovic. An User Interface Adaptation Architecture for Rich Internet Applications. In *Proceedings of 5th European Semantic Web Conference 2008, ESWC 2008, Tenerife, Spain*, volume 5021 of *Lecture Notes in Computer Science*. Springer Verlag, June 2008. (accepted).
12. Kay-Uwe Schmidt and Ljiljana Stojanovic. From Business Rules to Application Rules in Rich Internet Applications. In *Proceedings of Business Information Systems 11th International Conference, BIS 2008, Innsbruck, Austria, May 5-7, 2008*, volume 7 of *Lecture Notes in Business Information Processing*, pages 447 – 458. Springer Berlin Heidelberg, 2008. http://dx.doi.org/10.1007/978-3-540-79396-0_39.
13. Atom WG. Atom Publishing Format and Protocol . <http://tools.ietf.org/wg/atompub/>.

Test-Driven Development of Complex Information Extraction Systems using TEXTMARKER

Peter Kluegl, Martin Atzmueller, and Frank Puppe

University of Würzburg,
Department of Computer Science VI
Am Hubland, 97074 Würzburg, Germany
{pkluegl, atzmueller, puppe}@informatik.uni-wuerzburg.de

Abstract. Information extraction is concerned with the location of specific items in textual documents. Common process models for this task use ad-hoc testing methods against a gold standard. This paper presents an approach for the test-driven development of complex information extraction systems. We propose a process model for test-driven information extraction, and discuss its implementation using the rule-based scripting language TEXTMARKER in detail. TEXTMARKER and the test-driven approach are demonstrated by two real-world case studies in technical and medical domains.

1 Introduction

There are two main paradigms for information extraction approaches: Either the systems provide automatic extraction techniques, i.e., based on machine-learning techniques, or they are based on manually acquired knowledge, mainly considering rule-based knowledge. The first approach allows an easy application of the system since no knowledge acquisition phase is required, and the extraction knowledge can be automatically learned from annotated examples. However, the latter often performs better for rather complex domains. There are several reasons to prefer a knowledge engineering approach to a machine learning approach [1]: The knowledge can usually be captured and extended quite easily by a domain specialist and therefore provides flexible techniques if the requirements change, for extensions, and for exceptions for certain documents.

In this paper, we present an approach for the test-driven development of complex text extraction systems using the rule-based scripting language TEXTMARKER: We discuss a process model for the test-driven extraction process, and discuss the contained steps in detail. Furthermore, we describe the TEXTMARKER system for rule-based information extraction. After that, we provide two case studies for demonstrating and discussing the presented approach in detail.

The rest of the paper is structured as follows: Section 2 presents the process model for the test-driven development of complex text extraction systems using TEXTMARKER. Section 3 gives a conceptual overview on the TEXTMARKER system, describing the core concepts, the syntax and semantics of the TEXTMARKER language and its special features. Section 4 discusses two real-world case-studies for the presented approach. Finally, Section 5 concludes with a discussion of the presented work, and provides several interesting options for future work.

2 Test-Driven Process Model

In the following section, we describe the process model for test-driven development of complex text extraction systems using TEXTMARKER. We distinguish two roles, the knowledge engineer and the domain specialist. The latter is concerned with the annotation, selection, and formalization of documents/test cases, whereas the former performs the development and incremental refinement of the rule base for text extraction. The process is shown in Figure 1, and discussed below.

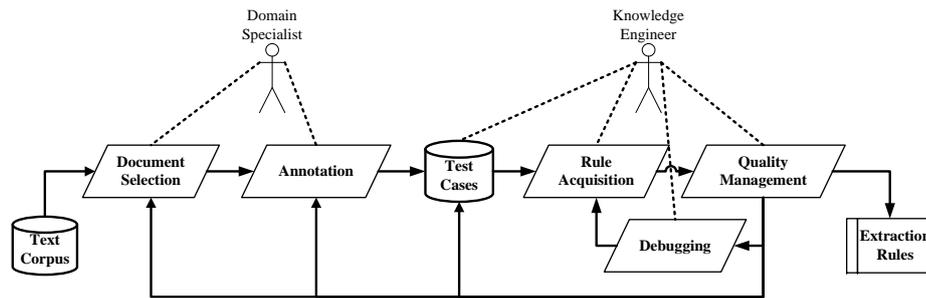


Fig. 1. Process Model: Semi-Automatic Rule-Based Instance Generation from Texts

- **Document Selection:** The document selection step provides a subset of the cases contained in the text corpus that are applied as test cases later. Synthetic cases can also be added (as test cases), therefore new cases can also be easily integrated for coping with new requirements. For document selection also cluster techniques can be applied in order to select a limited, heterogeneous and diverse set of cases.
- **Annotation:** During the annotation step, the domain specialist selects text fragments and assigns pre-specified types (i.e., annotations) to these. Each added annotation is either an input type (pre-specified information) or an output type (expected result). Input annotations are used for testing specific features/rules in their context (unit-tests). Output types are the *expected* types of the test. Visual editors allow the user to annotate the text interactively.
- **Rule Acquisition:** During the rule acquisition step new extraction rules are formalized, tuned and refined according to the given test corpus based on the results of the subsequent quality management and debugging steps.
- **Quality Management:** In the quality management step, the knowledge engineer assesses the correctness and completeness of the system based on the set of formalized test cases. Also, erroneous test cases that were marked during the quality management step can be incrementally corrected.
- **Debugging:** The debugging step is mainly concerned with debugging specific rules. The information about rule applications and their contexts can significantly help for the improvement of the rule base and also for correcting test cases.

3 Conceptual Overview on the TEXTMARKER System

Whenever humans perform manual information extraction they often apply a strategy according to a *highlighter metaphor*: First, relevant headlines are considered and classified according to their content by coloring them with different highlighters. The paragraphs of the annotated headlines are then considered further. Relevant text fragments or single words in the context of that headline can then be colored. Necessary additional information can be added that either refers to other text segments or contains valuable domain specific information. Finally the colored text can be easily analyzed concerning the relevant information. The TEXTMARKER¹ system tries to imitate this manual extraction method by formalizing the appropriate actions using *matching rules*: The rules mark sequences of words, extract text segments or modify the input document depending on textual features.

The current TEXTMARKER implementation is based on a prototype described in [2] that supports a subset of the TEXTMARKER language described below. The present TEXTMARKER system is currently being extended towards an integration as a UIMA (*Unstructured Information Management Architecture*) component [3]. The default input for the TEXTMARKER system is semi-structured text, but it can also process structured or free text. Technically, HTML is often used as a input format, since most word processing documents can be easily converted to HTML.

In the following sections we first give a conceptual overview on the TEXTMARKER language by introducing its core concepts. After that, we discuss the syntax and the semantics of the TEXTMARKER language in detail, and provide some illustrating examples. Next, we present special characteristics of the language that distinguishes the TEXTMARKER system from other common rule based information extraction systems.

3.1 Core TEXTMARKER Concepts

As a first step in the extraction process the TEXTMARKER system uses a scanner to tokenize the input document and to create a stream of basic symbols, providing the initial feature selection. The types of the possible tokens are predefined by a manually created taxonomy of annotation types. Annotations simply refer to a section of the input document and assign a type or concept to the respective text fragment. Figure 2 shows an excerpt of a basic annotation taxonomy: For example, *CW* describes all tokens, that contain a single word starting with a capital letter, *MARKUP* corresponds to HTML/XML tags, *ANY* combines all symbols that are not classified as *MARKUP* and *PM* refers to punctuation.

Using the taxonomy, the knowledge engineer is able to choose the most adequate types and concepts when defining new matching rules. If the capitalization of a word, for example, is of no importance, then the annotation type *W* that describes words of any kind can be used. The initial scanner creates a basic set of annotations that are used by the matching rules of TEXTMARKER. Most information extraction applications require domain specific concepts and annotations. Therefore, the knowledge engineer is able to define new annotation types depending on the requirements of the given domain. These types can then be flexibly integrated in the taxonomy of annotation types.

¹ textmarker is a common german word for a highlighter

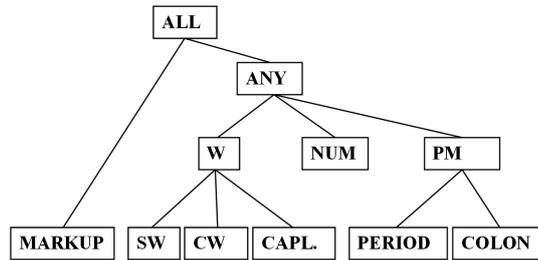


Fig. 2. Part of a taxonomy for basic types. (W=Word, NUM=Number, PM=Punctuations, SW=Word without capitals, CW=Word starting with a capital letter)

3.2 Syntax and Semantics of the TEXTMARKER Language

One of the goals in developing a new information extraction language was to maintain an easily readable syntax while still providing a scalable expressiveness of the language. Basically, the TEXTMARKER language consists of definitions of new annotation types and matching rules. These rules are based on a list of rule elements headed by the type of the rule.

The purpose of the different rule types is to increase the readability of rules by making their semantic intention explicit. Each rule element contains at least a basic matching condition referring to text fragments or already given annotations. Additionally a list of conditions and actions may be specified for a rule element. Whereas the conditions describe necessary attributes of the matched text fragment, the actions point to operations and assignments on the current fragments. Needless to say these actions will only be executed if all basic conditions matched on a text fragment or annotation and the related conditions are fulfilled. Table 3.2 contains a short and simplified excerpt of the TEXTMARKER syntax concerning matching rules.

Rule	→ RuleType RuleElement+ ';'
RuleType	→ 'ADDTYPE' 'DEFAULT' ...
RuleElement	→ MatchType Conditions? Actions? '+'?
MatchType	→ Literal Annotation
Annotation	→ 'ALL' 'ANY' 'MARKUP' 'W' ...
Conditions	→ '{' Condition (';' Condition)* '}'
Condition	→ '-'? CondType (';' parameter)*
CondType	→ 'PARTOF' 'CONTAINS' 'NEAR' ...
Actions	→ '(' Action (';' Action)* ')'
Action	→ ActionType (';' parameter)*
ActionType	→ 'MARK' 'FILTER' 'REPLACE' ...

Table 1. BNF-Extract of the TEXTMARKER language

Due to the limited space it is not possible to describe all of the various conditions and actions available in the TEXTMARKER system. However, the common usage of the language and its readability can be demonstrated by simple examples:

```
ADDTYPE CW{INLIST, animals.txt} (MARK, animal);
ADDTYPE animal 'and' animal
      (MARK, animalpair, 0, 1, 2);
```

The first rule considers all capitalized words that are listed in an external document *animals.txt* and creates a new annotation of the type *animal* using the boundaries of the matched word. The second rule searches for an annotation of the type *animal* followed by the literal *and* and a second *animal* annotation. Then it will create a new annotation *animalpair* covering the text segment that matched the three rule elements (the digit parameters refer to the number of matched rule element).

```
ADDTYPE ANY{PARTOF, paragraph, ISINTAG,
      font, color=red} (MARK, delete, +)+;
ADDTYPE firstname (MARK, delete, 0, 1) lastname;
DEFAULT delete (DEL);
```

Here, the first rule looks for sequences of any kind of tokens except markup and creates one annotation of the type *delete* for each sequence, if the tokens are part of a *paragraph* annotation and colored in red. The + signs indicate this greedy processing. The second rule annotates first names followed by last names with the type *delete* and the third rule simply deletes all text segments that are associated with that *delete* annotation.

3.3 Special Features of the TEXTMARKER Language

The TEXTMARKER language features some special characteristics that are usually not found in other rule-based information extraction systems. The possibility of creating new annotation types and integrating them into the taxonomy facilitates an even more modular development of information extraction systems than common rule-based approaches do. Beside others, there are three features that deserve a closer look in the scope of this work: The robust extraction by filtering the token or annotation set, the usage of scoring rules for uncertain and heuristic extraction and the shift towards a scripting language.

Robust extraction using filtering Rule-based or pattern-based information extraction systems often suffer from unimportant fill words, additional whitespace and unexpected markup. The TEXTMARKER System enables the knowledge engineer to filter and to hide all possible combinations of predefined and new types of annotations. Additionally, it can differentiate between any kind of HTML markup and XML tags. The visibility of tokens and annotations is modified by the actions of rule elements and can be conditioned using the complete expressiveness of the language. Therefore the TEXTMARKER system supports a robust approach to information extraction and simplifies the creation of new rules since the knowledge engineer can focus on important textual

features. If no rule action changed the configuration of the filtering settings, then the default filtering configuration ignores whitespaces and markup. Using the default setting, the following rule matches all four types of input in this example (see [2]):

```
DEFAULT 'Dr' PERIOD CW CW;  
  
Dr. Peter Steinmetz, Dr.PeterSteinmetz,  
Dr. <b><i>Peter</i> Steinmetz</b>
```

Heuristic extraction using scoring rules Diagnostic scores [4] are a well known and successfully applied knowledge formalization pattern for diagnostic problems. Single known findings evaluate a possible solution by adding or subtracting points on an account of that solution. If the sum exceeds a given threshold, then the solution is derived. One of the advantages of this pattern is the robustness against missing or false findings, since a high number of findings is used to derive a solution.

The TEXTMARKER system tries to transfer this diagnostic problem solution strategy to the information extraction problem. In addition to a normal creation of a new annotation, a *MARK* action can add positive or negative scoring points to the text fragments matched by the rule elements. If the amount of points exceeds the defined threshold for the respective type, then a new annotation will be created. Further, the current value of heuristic points of a possible annotation can be evaluated by the *SCORE* condition. In the following, the heuristic extraction using scoring rules is demonstrated by a short example:

```
ADDTYPE p{CONTAINS,w,1,5}(MARK,h1,5);  
ADDTYPE p{CONTAINS,w,6,10}(MARK,h1,2);  
ADDTYPE p{CONTAINS,emph,80,100,%}(MARK,h1,7);  
ADDTYPE p{CONTAINS,emph,30,80,%}(MARK,h1,3);  
ADDTYPE p{CONTAINS,w,0,0}(MARK,h1,-50);  
ADDTYPE h1{SCORE,10}(MARK,realh1);  
LOGGING h1{SCORE,5,10}(LOG,'Maybe a h1');
```

In the first part of this rule set, annotations of the type *p* (*paragraph*) receive scoring points for a *h1* (*headline*) annotation, if they fulfill certain *CONTAINS* conditions. The first condition, for example, evaluates to *true*, if the paragraph contains at least one and up to five words, whereas the fourth condition is fulfilled, if the paragraph contains thirty up to eighty percent of *emph* annotations. The last two rules finally execute their actions, if the score of a *headline* annotation exceeds ten points, or lies in the interval of five and ten points, respectively.

Shift towards a scripting language Some projects using the TEXTMARKER system have indicated that a rule-based language with a knowledge representation only based on annotations may not overcome all challenges of a high level information task. Often it is not possible to express the complex background knowledge with simple matching rules or to control the matching process without control structures like loops. Therefore the TEXTMARKER language is being extended with several common features of scripting languages.

- **Imperative programming:** The TEXTMARKER system does not impose an execution order on the rules based on the fulfilled conditions and/or their complexity. The development of bigger sets of rules has shown that a dynamic execution order holds almost no advantages over imperative program execution order. Additionally, the linear processing allows a comprehensible usage of filtering rules.
- **Variables:** The usage of variables can significantly improve the expressiveness of a simple language. It is already possible in the TEXTMARKER system to e.g., count the number of certain annotation types and evaluate the result in a condition. But practice has shown that the additional concept of variables and expressions on variables is very useful and helps to solve complex rule engineering problems in an elegant and simple way.
- **Conditioned loop blocks:** Besides variables and operations another construct known by scripting and programming languages are conditioned statements and loops. In the TEXTMARKER system we combine both constructs to the concept of the conditioned loop blocks. These complex statements contain an identifier, a rule and a list of rules, declarations or even other conditioned loop blocks. The rule defines both the condition statement and the loop statement: The annotations of the rule match on adequate text fragments. The conditions of the rule determine if the contained rules may be executed. Yet the rule can match several times and therefore defines a list of text fragments, on which the contained rules are applied.

```
BLOCK(' ID' ) headlinedParagraph
    {CONTAINS, relevantAnnotation} ( ...
        rules, declarations or blocks ...
    )
```

In this short and simplified example, rules, declarations or blocks of rules are only executed if an annotation of the type *headlinedParagraph* is located in the text and if that annotation contains at least one annotation of the type *relevantAnnotation* (condition statement). The statements in the block will be applied on all found text fragments the rule matched and only on them.

More precisely, if the rule has matched on five *headlinedParagraph* annotations, the contained statements will be executed five times overall, one time for each matched annotations. This additional block structure can therefore increase the performance of the TEXTMARKER system, because the considered text area can be restricted and the rules do not need to be applied on the complete document.

- **Method calls:** Another common feature is the declaration and the reference of methods or procedures. For this purpose we are using the conditioned loop blocks again. The identifier is used to call the block like a method by the action of a rule. If the calling rule is not part of the same block or rule file, additional identifiers, respectively file names must be used to reference the complete namespace of the block. Introducing method calls is enabling the TEXTMARKER system to utilize rule libraries and further increases its modularity and reuse.

These efforts for extending the TEXTMARKER language towards a scripting language was one of the reasons to replace the existing and successful development environment [2]. The new implementation is built on the Dynamic Language Toolkit² in order to support the described scripting functionality in the development process.

In test driven development automatic test cases are ideally written for small, atomic units [5]. The smallest unit of a TEXTMARKER scripting file is a single rule. However, with the integration in UIMA, the interfaces of a UIMA component and the information structure [6] are especially suitable for the specification of test cases. The TEXTMARKER system is integrated in UIMA as a component, i.e. an *Analysis Engine* that provides the functionality of including several scripting files. If the functionality of a large rule set is split into several modular scripting files with internal block structure, then it is still possible to create small and self-contained *testing units*. Therefore the test cases are specified independently of the TEXTMARKER implementation by using the UIMA interfaces, but they can still refer to specific parts of functionality, especially single block definitions of a TEXTMARKER scripting file.

4 Case Studies

In the following sections we describe two real-world case studies applying parts of the presented approach. The first case study is concerned with high-level information extraction in a technical domain. The second case study considers the generation of structured data records given semi-structured medical discharge letters.

4.1 High-Level Information Extraction

The case study is about a high level information extraction, automatic content segmentation and extraction task. Unfortunately, we can only describe the case study in a very general way due to non-disclosure terms. As a general setting, word processing documents in common file formats³ like Microsoft Word or OpenOffice are mined for information specific to certain projects, with temporal margins, e.g., similar to curricula vitae. The input documents feature an extremely heterogeneous layout and are each written by a different person. Interesting text fragments may relate from plain text to structured tables or even combinations or parts of them. Additionally, the layout is not sufficient enough for a correct classification, since also domain dependent semantics may change the relevance of a fragment. The output of a document are a set of templates that contain exact temporal information, the exact text fragment related to the template and various domain specific information, e.g., a project name or company names in our curriculum vitae example.

Although the application is still under development, it already involves 479 rules and several domain specific dictionaries with up to 80000 entries. Basically, the TEXTMARKER system tries to imitate the human perception of text blocks. For this purpose interesting named entities, e.g., temporary information, are recognized. Then, the application locates text structures of different types of complexity and size, e.g., a headlined

² Dynamic Language Toolkit: <http://www.eclipse.org/dltk/>

³ The input documents are converted to HTML

paragraph or a row of a table. If one of these text fragments or a set of text fragments of the same type contains a significant pattern of interesting named entities, then they are marked as a relevant block of text. Finally, additional rules are used to detect the domain specific information which is also used to refine the found segments. In the current state the TEXTMARKER application was evaluated on correct text fragments and temporal data only. It achieved an F1 measure of 89% tested on 58 randomly selected documents with 783 relevant text fragments. These results seem to indicate potential for further improvements, however, in order to obtain more reliable results we need to perform more evaluations together with our project partners first.

The development of the application used a process model similar to the common model with ad-hoc testing. Normally, an information extraction application is tested automatically for quality assurance. But due to the characteristics of the high level information extraction task, it is often not suitable to utilize complete annotated documents for back testing. Therefore a semi-automatic approach with several supporting tools was used. The applied process works as follows: At the beginning a new application or a new requirement is defined by the domain specialist. He or she manually selects a representative set of documents and creates a test corpus. The knowledge engineer develops new rules using the test corpus and informal specifications. The domain specialist tests the new rules with the test documents for their functionality. Then he or she creates a feedback document, a documentation of the errors with examples. Furthermore, the new rules are additionally tested on a new test corpus with randomly selected documents. The feedback document is extended with the new reported errors. The knowledge engineer writes new rules to correct the documented errors. If the functionality or the quality of the rules is not sufficient enough, the process is iterated: Either new features are added or the rule set has to be improved further. In both possibilities the knowledge engineer receives a new representative corpus for testing.

The experience with this application motivated the development of the presented test-driven process model. The process has already been partially implemented, and especially the controlled formalization of test cases, the isolated specification of new features and the automatic back testing of different kinds of test cases provide distinct advantages over the current ad-hoc testing process model.

4.2 Diagnostic Case Extraction from Textual Discharge Letters

The second case study considers the generation of cases from semi-structured medical discharge letters. These letters are written by the physicians when a patient has been diagnosed and leaves after a hospital stay. The letters are typically written by the responsible physicians themselves and are stored as MS Office (Word) documents. These contain the observations, for example, the history of the patient, results from certain examinations, measurements of laboratory parameters, and finally the inferred diagnoses of the patient. Figure 3 shows an example of a partial (anonymized) discharge letter with the diagnosis, anamnesis, and some laboratory values. The available electronic discharge letters provide the basis for various purposes, for example, for quality control with respect to a hospital information system, for medical evaluations, or for creating case-based training sessions.

Diagnosen:

Leberzirrhose ethyltoxisch CHILD C
Therapieresistenter Aszites (chylös)
Indikation zur TIPSS-Anlage
Indikation zur Lebertransplantation
Z. n. Ösophagusvarizenblutung
Z. n. spontan bakterielle Peritonitis
Z. n. Prostataektomie bei Prostata-Karzinom.
Hyperplastischer Magenpolyp
Z. n. Polypektomie eines Colon-Polypens an der Bauhin'schen Klappe.

Anamnese:

Herr X wurde uns mit therapieresistentem Aszites bei Leberzirrhose vorgestellt. Auch unter gesteigerter Diuretika-Dosierung waren Aszitespunktionen in kurzen Abständen notwendig. Herr X wurde nun zur erneuter Aszitespunktion und Neueinstellung der medikamentösen Therapie stationär aufgenommen. Des Weiteren sollte eine Gastroskopie bei bekannten hyperplastischen Magenpolypen durchgeführt werden. Der Patient berichtete über eine Gewichtszunahme von 6 kg innerhalb von einer Woche. Die Trinkmenge läge z. Z. bei 2,5 – 3 Liter pro Tag. Wegen zahlreicher Nebenwirkungen wurde die aktuelle Medikation in Rücksprache mit unterer gastroenterologischen Ambulanz abgesetzt. Zuletzt nahm der Patient an Torasemid 20 mg und Spironolacton 200 mg täglich ein.

Labor:

XX.XX.20XX 10:20:00)

Klinische Chemie: Eisen: 38 [59 - 158] µg/dl;

Gerinnung: Thromboplastinzeit n. Quick: 44 [70 - 130] %; Ratio int. norm.: 1.63 [0.85 - 1.18] ; PTT: 64.1 [23 - 36] s;

Antithrombin III: 27 [75 - 125] %; Fibrinogen (Clauss): 2.6 [1.8 - 3.5] g/l; Faktor II: 27 [70 - 120] %; Faktor V: 27 [70 - 140] %; D-

Dimere (immunol.): 0.420 [0 - 0.190] mg/l;

Serumproteine und Tumormarker: Ferritin: 30 [30 - 400] µg/l; Transferrin: 169 [200 - 380] mg/dl; Transferrinsättigung: 15.9 [16 -

Fig. 3. Example of a partial discharge letter (in german): The screenshot shows the diagnoses, anamnesis, and laboratory examination part ("Diagnosen, Anamnese, Labor"). Then, the segments corresponding to these need to be extracted, and post-processed for data extraction.

The text corpus is made up of a set of discharge letters for a set of patients. The goal is to process these and to extract the relevant information (observations, diagnoses) from the discharge letters. We started with a training corpus of 43 discharge letters. For extracting the relevant information, we developed a set of rules that take the structure of the document into account. A discharge letter needs to follow a certain standard structure: The document is started by the salutation, the diagnosis part, the history of the patient, textual paragraphs describing the results of various examinations like computer tomography (CT), and the result of laboratory examinations, i.e., the measured parameters. For applying the TEXTMARKER system, we can therefore focus on these building blocks of the document. Therefore, the domain specialist provided this information and annotated several documents concerning the important text blocks, and the respective concepts. Each of the documents contained in the test corpus was annotated with the concepts that are mentioned in the document. In this way, we developed a set of rules for extracting segments of the letter first, for example, considering the diagnosis block. After that, those segments were split up further, for example, considering the fact that individual diagnoses are almost always contained in separate lines.

The corpus is still being extended, and new diagnoses and observations are being added to the set of important concepts. Therefore, this provides for an ideal option for further applying and testing the presented approach. The new concepts can be integrated and captured with new rules, and their application can be debugged in context using the new framework.

5 Conclusions

Information extraction is part of a widespread and still growing scientific community that originates a multiplicity of new systems, tools and approaches. The initial development of the TEXTMARKER system was influenced by the LAPIS system [7] and the LIXTO SUITE [8] with its LIXTO VISUAL WRAPPER.

Test-driven development is a well known and successfully applied development strategy. It is often combined with agile methods like extreme programming and is supported by an automatic testing framework. Test-driven development is not only a test first approach for quality management, but also for the analysis and design process [5].

Various studies have shown that test-driven development reduces the defect rate and detects defects earlier. Maximilien et al. [9] have shown a reduction of defect by 50 percent compared to an ad-hoc unit testing approach. Baumeister et al. [10] applied automatic tests and restructuring methods for an agile development of diagnostic knowledge systems. They defined different types of tests, e.g., on correctness, anomalies or robustness, and noticed significant improvements for the evolutionary development.

In the area of text mining and information extraction, ad-hoc testing against a hand annotated *gold standard* is common practice. The tool *CFE* (Common Feature Extraction) [11] is a system for testing, evaluation and machine learning of UIMA based applications. It provides the declarative language *Feature Extraction Specification Language* (FESL) that is interpreted and executed by a generic UIMA component. However, to the best knowledge of the authors, there is no prominent tool that supports a test-driven development of information extraction applications beyond common back testing. The strategy of test-driven development can be used for the development of complex information extraction applications. Yet, the transfer is not straight forward for common rule-based or pattern-based tools. The test specification and the test framework has to incorporate the imprecise nature of the unstructured information domain.

In this paper, we have presented a test-driven approach for the development of complex text extraction systems using TEXTMARKER. We have proposed a process model for the discussed task, and we have introduced the necessary components and features of TEXTMARKER in detail. Additionally, we have discussed two real-world case studies for exemplifying the presented approach.

The test-driven strategy is being integrated in our case studies. We expect a significant improvement in the development in general and especially in defect detection, defect reduction and an accelerated development. The process model described in this paper has some prominent features that are not found in the common development strategy. The presented process model supports an incremental development with minimal initial test cases. Real world test cases that define the common requirements can be combined with synthetic test cases for specific features and quality management. Furthermore, the proposed test-driven development process contains short iterations of different steps and provides a flexible way to create complex information extraction applications. The debugging step combines the advantages of the rule-based approach of the language, the powerful integrated development environment and the information contained in the test cases. Therefore detailed debugging information about the rule applications, the matched text and the conditions of each rule elements can explain each occurred error. The common *red-green* metaphor of the automatic testing frameworks is

therefore extended, because the test information can be displayed in combination with debugging information directly in the textual document of the test case.

In the future, we plan to consider one major part of test-driven development that was not addressed yet: Refactoring techniques for TEXTMARKER scripts in order to further enhance the user experience and applicability of the presented approach. Additionally, we aim to integrate machine learning techniques, e.g., knowledge-intensive subgroup discovery methods [12], for a more semi-automatic development approach.

Acknowledgements

This work has been partially supported by the German Research Council (DFG) under grant Pu 129/8-2.

References

1. Appelt, D.E.: Introduction to Information Extraction. *AI Commun.* **12**(3) (1999) 161–172
2. von Schoen, P.: Textmarker: Automatische Aufbereitung von Arztbriefen für Trainingsfälle mittels Anonymisierung, Strukturerkennung und Terminologie-Matching [TextMarker: Automatic Refinement of Discharge Letters for Training Cases using Anonymization, Structure- and Terminology-Matching]. Master's thesis, University of Wuerzburg (2006)
3. Ferrucci, D., Lally, A.: UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Nat. Lang. Eng.* **10**(3-4) (2004) 327–348
4. Puppe, F.: Knowledge Formalization Patterns. In: *Proc. PKAW 2000*, Sydney, Australia (2000)
5. Janzen, D., Saiedian, H.: Test-Driven Development: Concepts, Taxonomy, and Future Direction. *Computer* **38**(9) (2005) 43–50
6. Götz, T., Suhre, O.: Design and Implementation of the UIMA Common Analysis System. *IBM Syst. J.* **43**(3) (2004) 476–489
7. Kuhlins, S., Tredwell, R.: Toolkits for Generating Wrappers – A Survey of Software Toolkits for Automated Data Extraction from Web Sites. In Aksit, M., Mezini, M., Unland, R., eds.: *Objects, Components, Architectures, Services, and Applications for a Networked World. Volume 2591 of Lecture Notes in Computer Science (LNCS)*, Berlin, International Conference NetObjectDays, NODe 2002, Erfurt, Germany, 2002, Springer (2003) 184–198
8. Baumgartner, R., Flesca, S., Gottlob, G.: Visual Web Information Extraction with Lixto. In: *The VLDB Journal*. (2001) 119–128
9. Maximilien, E.M., Williams, L.: Assessing Test-Driven Development at IBM. In: *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, Washington, DC, USA, IEEE Computer Society (2003) 564–569
10. Baumeister, J., Seipel, D., Puppe, F.: Using Automated Tests and Restructuring Methods for an Agile Development of Diagnostic Knowledge Systems. In: *FLAIRS'04: Proc. 17th Intl. Florida Artificial Intelligence Research Society Conference*. (2004) 319–324
11. Sominsky, I., Coden, A., Tanenblatt, M.: CFE - a System for Testing, Evaluation and Machine Learning of UIMA based Applications. In: *LREC '08: The sixth international Conference on Language Resources and Evaluation. Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP*. (2008)
12. Atzmueller, M., Puppe, F., Buscher, H.P.: Exploiting Background Knowledge for Knowledge-Intensive Subgroup Discovery. In: *Proc. 19th Intl. Joint Conference on Artificial Intelligence (IJCAI-05)*, Edinburgh, Scotland (2005) 647–652

UML Representation Proposal for XTT Rule Design Method*

Grzegorz J. Nalepa¹ and Krzysztof Kluza¹

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

Abstract In the paper practical issues concerning the use of UML as a knowledge representation method for rules are discussed. A proposal of an UML-based representation for the XTT structured representation for rules is presented. Since some deep semantical differences between UML and rule-based representation exist, several possible UML representations are evaluated. A practical algorithm for building an UML representation using activity diagrams from XTT tables is proposed.

1 Introduction

Using Knowledge Engineering (KE) methods in practical Software Engineering [1] (SE) has gained some momentum in recent years. One of the best examples is the business rules approach [2]. The fact is that KE is being developed in parallel with SE, and both approaches use different methods and tools to actually model and build systems. Important semantical differences between these two make the use of the KE methods in SE non-trivial, whereas using SE methods with KE problems is often of limited use.

The paper focuses on analyzing possibilities of the *practical* use of the UML language to model XTT rule-based systems. Extended Tabular Trees (XTT for short) [3] is a structured knowledge representation for rules, based on some classic KE notions of decision tables and decision trees. Representing XTT with UML artifacts encounters number of issues addressed in the paper. A new algorithm for encoding an XTT diagram using UML is introduced.

The paper is organized as follows. In the Sect. 2 possibilities of using UML as a knowledge representation language are discussed. Then, in Sect. 3 knowledge representation issues in the HEKATE project are presented. The paper presents an UML-based representation for the XTT rule representation method. The application of this method is discussed using the example thermostat control system presented in Sect. 4, whereas the method itself is introduced in Sect. 6 and 5. The UML representation of the thermostat is then discussed in Sect. 7. Directions for future work are presented in the final section.

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

2 UML as a Knowledge Representation Language

UML as a design language identifies two distant domains of SE, that is software structure, and behavior modelling. So there are two main diagram classes: Structure Diagrams and Behavior Diagrams containing different artifacts.

Apparently, Structure Diagrams are often considered the basis of UML. They are fairly complete and allow for expressing software components and denoting relationship among them easily (i.e.: Class Diagram, Component Diagram). Behavior diagrams model software logic at different abstraction levels. First of all there is a big picture perspective: modelling what particular software should do, from the user point of view (i.e.: Use Case Diagram). There is also a detailed perspective: what particular software components defined by the Structure Diagrams should do (i.e.: State Machine Diagram, Interaction Diagram). The problem is that these two perspectives do not fit together very well with the Structure Diagram. While the detailed perspective corresponds to classes, the big picture one serves more as a guideline, than a real modelling tool. So it seems that the relationship between modelling software behavior and its structure is unclear.

Another issue regards the *semantic gap* between the design and the implementation [4]. Even if discussed diagrams support the implementation process by describing software in a comprehensive way, it is impossible to verify in a reasonable time if the implementation matches the design. There is also another gap in the specification-design-implementation process called *analysis specification gap* [5]. It regards a difficulty with the transition from a specification to the design. Formulating a specification which is clear, concise, complete and amenable to analysis turns out to be a very complex task, even in small projects.

Applying UML as a Knowledge Engineering method is not straight forward [6]. Existing diagrams are not suitable for rule modeling or expressing knowledge in general. Using an UML profile, which is a redefinition of the semantics of certain diagrams, does not help much, and in some cases might complicate the design. It forces the use of existing diagrams for purposes they were not designed for i.e. representing rule sets is tricky and inefficient.

There are several possible approaches when it comes to practical UML application for knowledge engineering. The first solution is the “classic” and definitely the easiest one. It consists in modelling the system with a knowledge-based approach, that uses some classic knowledge representation method, such as decision trees, then design the software implementation using UML, and generate an object-oriented (OO) code. In this case, KE methods are used in the “design” stage, while SE methods provide “implementation” means.

In the second approach the rule-based knowledge is modelled with UML diagrams, and then the corresponding OO code is generated. This approach relies on either extending, or redefining the original semantics of UML. Some early beginning can be observed in *OMG Production Rule Representation* [7], where some ideas of extending existing semantics of UML were contained. However, a complete example of this approach may be found in the *Unified Rule Modelling Language* (URML), (see [8]). In this case, existing UML diagrams are used to model different type of rules. In URML a simple production rule

If car is new, then increase premium by \$400. is modeled in URML as shown in Fig. 1.

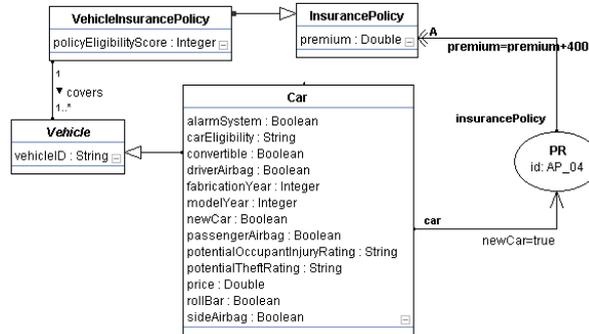


Figure 1. Production rule in URML

The last one is possibly the most complicated approach. It relies at incorporating a complete rule-based logic core into an OO application. It aims at minimizing the semantic gap between SE and KE. Such a solution is being developed in the *HeKatE* project, where a declarative, rule-based core is integrated into an OO application as a logical model (as in the Model-View-Controller design pattern [9]). In this paper it is proposed to find a UML representation corresponding, to XTT. Such a representation could then be used to present a complete UML application model designed with KE methods.

3 Knowledge in the HeKatE Design Process

The HEKATE project aims at applying selected AI tools into SE. It is based on incorporating an extended rule model as a logical application core. The model uses the *XTT* rule representation and design method, with the support of the ARD (Attribute Relationship Diagrams) [10,11] rule prototyping method. The project provides a hierarchical design process, which should ultimately be represented by both custom XTT/ARD methods, as well as an UML-bases representation. The main difference between the HeKatE knowledge representations and UML diagram is, that UML, after all, does not provide a design process. Whereas, HeKatE is about the integrated design process. So the methods on which HeKatE is based, have been invented with the design process in mind.

The *XTT* (*EXtended Tabular Trees*) knowledge representation [3], has been proposed in order to solve some common design, analysis and implementation problems present in rule-based systems. In this method three important representation levels has been addressed: *visual* – the model is represented by a hierarchical structure of linked extended decision tables, *logical* – tables correspond to sequences of extended decision rules, and *implementation* – rules are

processed using a Prolog representation. On the visual level the model is composed of extended decision tables, see Fig. 2. The table represents a set of rules, having the same attributes. On the logical level, a table corresponds to a number of rules, processed in a sequence. If a rule is fired and it has a link, the inference engine processes the rule in another table.

In addition to XTT which represents rules, there is a rule design process involved. The process begins with the ARD model, and ends with the XTT model. The key underlying assumption in the ARD design with knowledge specification in attributive logics is that, similarly as in the case of Relational Databases [12], the attributes are *functionally dependent*. An ARD *diagram* is a conceptual system model at a certain abstract level. Attributes are subsequently identified at more and more detailed levels. The process includes all levels. At the most detailed level, XTT diagrams are added to define dependencies among attributes and to describe how to calculate attribute values. The ARD process is similar, in terms of its goals, to Structure Diagrams. However, while the Structure Diagrams tend to describe what elements the software consists of, ARD describes what is *known* about it.

4 Thermostat Case Study

The analysis of the UML representation is conducted using a classic rule-based control system example, a *Thermostat case*, found in [13]. The main problem consists in creating a temperature control system for an office. The system needs to take into account current date, including the day of the week, as well time of the day. The original design has 18 rules, and has been studied in detail in the HEKATE project. Here only the complete XTT design is presented in Fig. 2.

In the subsequent sections several approaches to the XTT representation in UML are presented, with the optimal one used to represent the whole XTT Thermostat design.

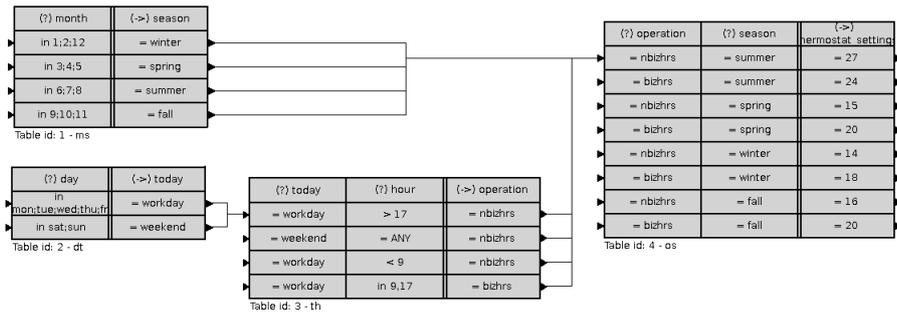


Figure 2. XTT Thermostat Design

5 Modeling XTT with UML

In this section the evolution of the UML representation for XTT is discussed. Several attempts to provide such a representation are presented, in order to expose the semantic and conceptual differences between UML and knowledge-based systems. Finally, an optimal solution is proposed.

Decision tables in XTT represent rules that have the same attributes. The rules in a single table are processed sequentially. So at this point a reasonable idea is to try to use diagrams that show not so much the structure of the system, but its behavior (dynamics). It could be diagrams such as use case diagrams, activity diagrams or state diagrams, as well as diagrams of interaction (sequence or collaboration diagrams).

According to [6] State Machine Diagrams and Activity Diagrams seem to be the best UML candidates for rule modeling, but not good enough to serve the purpose of rule modeling with similar expressiveness as XTT. It is possible to use them to express rules in case of smaller systems. However, in case of larger systems, where the number of states grows fast their use poses some practical problems. So the first attempt to use UML for XTT is to investigate activity and state diagrams.

Activity and state diagrams are related. However, it is important to understand the differences between them. A state diagram shows the possible states of the object and the transitions that cause a change in state. It focuses on an object undergoing a process (or on a process as an object). However, an activity diagram focuses on the flow of activities involved in a single process and shows how they depend on one another [14].

The first attempts were carried out using state diagram for XTT modelling. State diagrams capture the behavior of a software system and state machine uses graph notation to represent the behavior of a component of a system [15]. The Table 1 lists the types of pseudostates used in state diagrams.

Table 1. Types of used pseudostates

	Initial pseudostate – the starting point of a state machine.
	Choice – allows the execution of a state machine to choose between several different states based on guard conditions on the transitions.
	Fork and join – represents a split in the execution of the state machine into orthogonal regions. The join reunites the regions into a single transition. The state machine won't transition from the join until all regions have transitioned to the join pseudostate.

All of the attempts to find a UML representation are presented on a single XTT table from the Thermostat systems, the *TH* table.

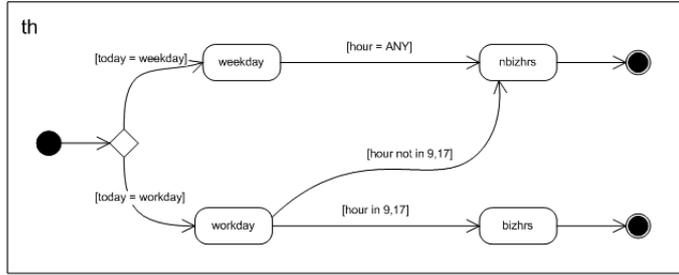


Figure 3. First attempt: State diagram corresponding to XTT TH diagram

The first abandoned attempt The Fig. 3 shows the corresponding UML state diagram for the sample XTT table. However, in this case, where XTT is transformed into activity diagrams it is not clear which attributes should be transformed to the states and which to guard conditions.

The second abandoned attempt To avoid the problem from the first attempt (how to transform XTT attributes), let:

- values of the XTT output attributes (H in XTT) become states,
- individual rows from XTT (conjunction of values in their cells) become guard conditions.

The Fig. 4 shows the modified UML state diagram corresponding to the sample XTT table. Unfortunately, with when the number of rules in XTT table grows, the diagram becomes poorly readable.

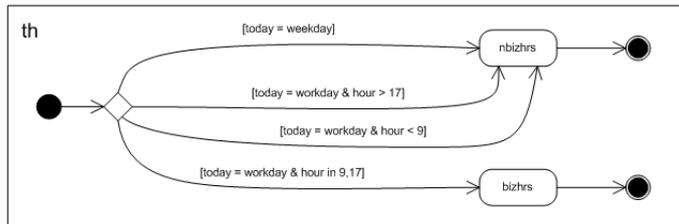


Figure 4. First attempt: State diagram corresponding to XTT TH diagram

The third abandoned attempt The diagram could be more readable if we use a fork pseudostate instead of a choice pseudostate. The Fig. 5 shows the corresponding UML state diagram for the sample XTT table with the fork pseudostate. If there is an deficiency in an XTT table (e.g. as a result of a mistake)

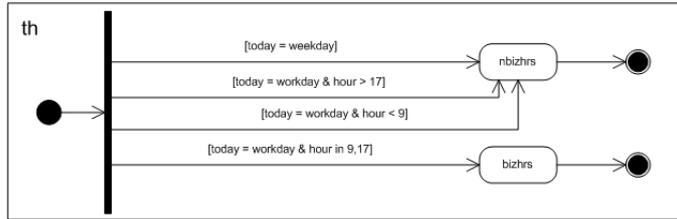


Figure 5. Third attempt: State diagram corresponding to XTT TH diagram

and different rows will not exclude each other, than the fork pseudostate duplicates the input value and may transfer the control to more than one edge of the subsequent states.

The limitation of all of these three approaches is the lack of the output attribute naming, and for getting the names of input attributes it is needed to search for them in the guard conditions.

6 UML Model for XTT

Considering the previously analyzed approaches, in this section a more optimal translation is proposed. In general, activity diagrams are related to flow diagrams and can illustrate the activities taking place in the system. The Table 2 lists the types of nodes used in activity diagrams.

Table 2. Types of used pseudostates

	Action
	Decision node
	Merge node
	Fork node
	Join node
	Partitions (swimlanes)
	Parameter of activity

Finally, an algorithm to transform XTT diagrams to UML activity diagrams has been formulated. The proposed transition algorithm from XTT table to UML activity diagram is as follows:

1. All input attributes become input parameters and output attribute becomes output parameter of an activity (for the demarcation the diagram can be divided into the partitions with a swimlane), see Fig. 6.

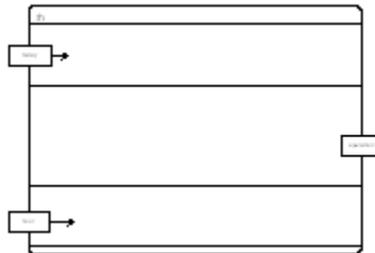


Figure 6. Example of applying of the first step of algorithm

2. For each attribute (activity parameter), if there is more than one unique value in the XTT, a decision node and for every unique value of attribute needs to be added (see Fig. 7):
 - (a) the control flow with guard condition is introduced (with that unique value in it),
 - (b) if the value occurs frequently, the flow is finished with a fork node with number of outputs equal to the number of times the value appears in XTT table.

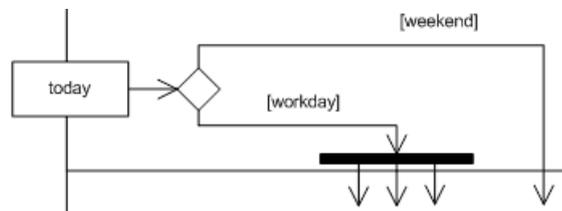


Figure 7. Example of applying of the second step of algorithm

3. For each rule (a row in XTT) a join node with the number of inputs equal to the number of input parameters is drawn and another one for output, see Fig. 8. For each join node:
 - (a) inputs are connected using an adequate flow control (in accordance with the values of attributes in the rule),
 - (b) outputs are connected using a flow control with the action having a value corresponding to the output attribute in the rule:
 - i. directly, if the value of attribute occurred in XTT only once,

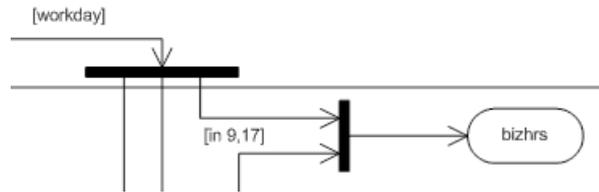


Figure 8. Example of applying of the third step of algorithm

- ii. otherwise through a merge node.
- 4. Outputs of all actions are merged in a merge node and a control flow is lead to output parameter of activity, see Fig. 9.

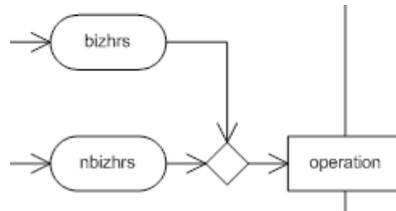


Figure 9. Example of applying of the fourth step of algorithm

It is worth noting, that in general a counter-wise transformation could be considered. This would allow for UML-based XTT rule design in any standard-compliant UML editor. However, this is not possible without introducing some kind of special annotations in the UML model. Ultimately, an UML profile for XTT is considered as a solution for this problem.

7 XTT for Thermostat in UML

Activity diagrams constructed with the algorithm above for the thermostat will look as in the Tables 3, 4, 5, 6. For the sake of transparency, modeling the diagram of the entire thermostat, these activities are nested. An activity presented in a nested form refers to a number of actions of this activity. (However, it is not presented directly in the diagram). Figure 10 shows the diagram for the whole thermostat system as discussed in the Sect. 4.

8 Evaluation and Future Work

In this paper the use of UML as knowledge representation for rule-based systems has been considered. Several possibilities have been described in order to expose

Table 3. Activity diagram corresponding to XTT MS table

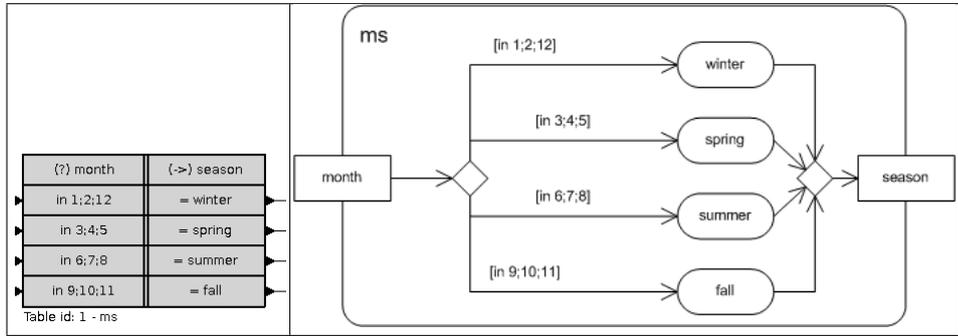
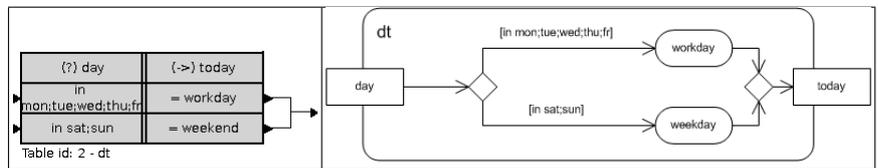


Table 4. Activity diagram corresponding to XTT DT table



some non-trivial issues concerning this representation. The original contribution of the paper is an UML-based representation of XTT diagrams.

Since the problem is not new, some other approaches exist, so it is worth noting how the HeKatE approach compares to the existing solutions. Currently, two most important representations include OMG PRR [7] and REVERSE URML [8]. The fact is, that both of these aim at detailed modelling of single rules. On the other hand, by definition, in the XTT approach the design is focused on the tree like structure of decision tables. So the representation introduced in this paper aims at translating the whole structure of extended decision tables into UML. Another difference is, that the HeKatE approach does

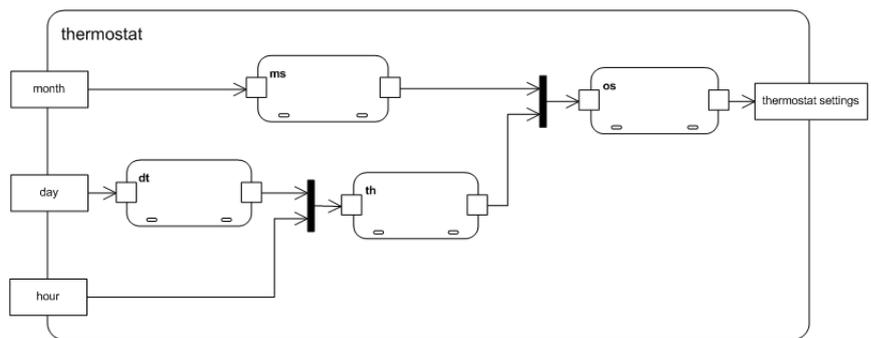


Figure 10. Activity diagram for the whole thermostat

directly aimed at other rule formalisms. However, the approach for providing the transformation is a generic one, so in the future its application to different rule formats may be considered. Ultimately the model designed with this method should be embeddable into any business application using the MVC pattern. In the future, the whole HeKatE design process including rule prototyping with ARD and design with XTT should be put into the context of the MDA [16].

References

1. Sommerville, I.: Software Engineering. 7th edn. International Computer Science. Pearson Education Limited (2004)
2. Ross, R.G.: Principles of the Business Rule Approach. 1 edn. Addison-Wesley Professional (2003)
3. Nalepa, G.J., Ligeza, A.: A graphical tabular model for rule-based logic programming and verification. *Systems Science* **31**(2) (2005) 89–95
4. Mellor, S.J., Balcer, M.: Executable UML: A Foundation for Model-Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA (2002)
5. Rash, J.L., Hinchey, M.G., Rouff, C.A., Gracanin, D., Erickson, J.: A tool for requirements-based programming. In: Integrated Design and Process Technology, IDPT-2005, Society for Design and Process Science (2005)
6. Nalepa, G.J., Wojnicki, I.: Using UML for knowledge engineering – a critical overview. In Baumeister, J., Seipel, D., eds.: 3rd Workshop on Knowledge Engineering and Software Engineering (KESE 2007) at the 30th annual German conference on Artificial intelligence : [September 10, 2007, Osnabrck, Germany]. (september 2007) 37–46
7. OMG: Production rule representation. Technical report, Object Management Group (br/2003-09-03)
8. Lukichev, S., Wagner, G.: Visual rules modeling. In: Sixth International Andrei Ershov Memorial Conference Perspectives Of System Informatics, Novosibirsk, Russia, June 2006. LNCS, Springer (2005)
9. Burbeck, S.: Applications programming in smalltalk-80(tm): How to use model-view-controller (mvc). Technical report, Department of Computer Science, University of Illinois, Urbana-Champaign (1992)
10. Nalepa, G.J., Ligeza, A.: Conceptual modelling and automated implementation of rule-based systems. In Krzysztof Zieliński, T.S., ed.: Software engineering : evolution and emerging technologies. Volume 130 of *Frontiers in Artificial Intelligence and Applications.*, Amsterdam, IOS Press (2005) 330–340
11. Nalepa, G.J., Wojnicki, I.: Towards formalization of ARD+ conceptual design and refinement method. In: FLAIRS2008. (2008) accepted.
12. Connolly, T., Begg, C., Strechan, A.: Database Systems, A Practical Approach to Design, Implementation, and Management. 2nd edn. Addison-Wesley (1999)
13. Negnevitsky, M.: Artificial Intelligence. A Guide to Intelligent Systems. Addison-Wesley, Harlow, England; London; New York (2002) ISBN 0-201-71159-1.
14. Miller, R.: Practical UML: A Hands-On Introduction for Developers, <http://dn.codegear.com/article/31863#activity-diagrams>. (2003)
15. Pilone, D., Pitman, N.: UML 2.0 in a Nutshell. O’Reilly (2005)
16. Miller, J., Mukerji, J.: MDA Guide Version 1.0.1. OMG. (2003)

Proposal of a Prolog-based Knowledge Wiki^{*}

Grzegorz J. Nalepa and Igor Wojnicki

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

Abstract This paper presents a proposal of a knowledge wiki system and describes its prototype implementation. The system design is based on a wiki concept regarding authoring policies and user access, extended with an ability of storing knowledge in addition to the human-readable content. The knowledge is expressed in the Prolog language by means of predicate logic. An inference engine is coupled with the wiki providing means for an automated knowledge processing and interpretation. Selected applications, examples, and further extensions are also presented.

1 Introduction

The Internet has become a single most important resource for instant information sharing. From the point of view of ordinary users it can be considered to constitute a very flexible and powerful version of the so-called blackboard architecture. The rapid growth of the Internet prompted a rapid development of specific software. This software addresses different aspects of information organization and interchange related issues, such as storing, retrieval, searching, indexing, aggregating, sharing, updating, etc. Taking into account the size and flexibility of the system as a whole, these problems constitute a real challenge.

Today web technologies constitute an advanced and universal programming framework. This framework has a very heterogeneous structure including: data encoding and structuring languages (such as HTML and XML), meta-data languages (such as RDF), data transformation languages (such as XSL/T/FO), data presentation languages (such as CSS), server-side programming languages (such as PHP, JSP), and client-side programming languages (such as Javascript).

Building on these *Content Management Systems* (CMS for short) emerged as a technology providing a unified interface for large databases or data warehouses. However, they do not solve all the challenges in the area. Some of the persistent problems include distributed authoring, where number of users provide and modify the content, the difference between content and knowledge, where the system should not just store data. Some of these issues were partially resolved with the introduction of specialized CMS called *wikis*, that introduce a

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

highly distributed authoring model. This technology is being enhanced with the development of *semantic wikis* allowing for semantic annotation of the content. The next step towards true knowledge-based systems is provided by knowledge wikis, that introduce explicit knowledge representation.

In this paper a new approach to the knowledge wiki architecture is proposed. It is based on a simple yet flexible and powerful idea of incorporating Prolog language code into the content stored in the wiki system. In the paper the design of such a wiki, based on the DokuWiki is introduced and a prototype implementation is discussed.

The rest of the paper is organized as follows. In Sect. 2 a discussion of main features and limitations of CMS is contained. Next, in Sect. 3 the wiki systems are discussed. The main problem with classic CMS, including wikis, is the lack of support for true knowledge management, as pointed out in Sect. 4. Some Wiki extensions, such as semantic and knowledge wikis presented in Sect. 5, aim at overcoming it. A knowledge wiki can be built around the classic knowledge representation techniques used in the Prolog language (see Sect. 6). A Prolog based extension to a wiki system is proposed in Sect. 7, with a prototype implementation for the DokuWiki introduced in Sect. 8. Some examples for the prototype are shown in Sect. 9. The paper ends with directions for future work in Sect. 10.

2 Content Management Systems

In a broad sense a *Content Management System* (CMS) consists of: a RDBMS (Relational Data Base System) which stores the data, and a complex web-based application or interface providing the access to the data. The web interface is built around common web technologies. This software requires a proper and efficient run-time environment, including a web server. An important aspect of a CMS system is the possibility of user personalization, and interactive use.

Multiple CMS categories can be identified: web portals, groupware suites, forum sites, and e-learning toolkits. In each of these cases the content they manage differs. Several classes or groups can be identified. The main differentiating factor is the type of services they are oriented for. A special case of CMS is a *wiki*, described in the next section.

From a knowledge engineering and management perspective, content may be seen as a particular kind of knowledge. While all of the CMS systems provide some kind of content structuration, few of them focus on using proper knowledge representation methods. The gathered content is only human readable, and hardly subject to machine processing. This, in fact, exposes an important conceptual gap between content and knowledge management systems. Whereas the former simply store data, the latter should allow for a semantic-aware processing.

3 Wiki Systems

The wiki concept emerged in the 90's. The main idea was to create a simple and expressive tool for communication and knowledge sharing. A wiki system is

mainly a collaboration tool. It allows multiple users to access, read, edit, upload and download documents. It has a regular client-server architecture. Documents are text-based, enriched with so-called wiki markup. It is a simplistic, tag-based, text only language which allows the user to annotate text with information regarding its structure and presentation. Such an enriched text is called *wikitext*. The tags allow users to make sections, subsections, tables, items and other typographic and structural operations. A wikitext remains human readable, tags are intuitive and easy to learn.

Each document is uniquely identified by a keyword, which makes the wiki concept similar to the encyclopedia concept. Furthermore a document, in addition to typographic tags mentioned earlier, can contain hyperlinks to other documents. A hyperlink is a document name enclosed by a link tag. The wiki allows users to upload images, as well as other files and link them together. One of the most important features of a wiki is an integrated version control. Each page modification is recorded. At any time a user can access any previous version of any page.

Wikis are mostly web-based. The web interface allows users to access, see and edit wiki pages. Depending on the particular solution, there might be access control and authorization mechanisms implemented. A wiki system is usually based on server side processing technologies providing the web application, and optionally a database back-end. As a front-end a web browser is used.

It is worth pointing out that wiki systems currently blend with regular Content Management Systems. Some CMS provide wiki functionality while some wikis evolve into CMS. Similarly wikis are more and more often merged with *e-Learning* systems to support collaborative knowledge gathering and sharing.

One of the most interesting wiki systems for developers is DokuWiki (wiki.splitbrain.org/wiki:dokuwiki). It is designed to be both easy to use and easy to set up. DokuWiki is based on PHP and does not require any database back-end. Pages are stored as versioned text files which enables easy backup-restore operations. It allows for image embedding, and file upload and download. Pages can be arranged into so-called namespaces which act as a tree-like hierarchy similar to directory structure. It also provides syntax highlighting for in-page embedded code for programming languages such as: C/C++, Java, Lisp, ADA, PHP, SQL and others, using GeSHi (qbnz.com/highlighter).

Furthermore, it supports extensive user authentication and authorization mechanisms including Access Control Lists (ACL). Its modularized architecture allows the user to extend DokuWiki with plugins which provide additional syntax and functionality. The most popular plugins provide: user and ACL management, blog, gallery of pictures, discussion board, calendar, \LaTeX symbols rendering, and GraphViz visualization.

4 Content vs. Knowledge

A review of the selected CMS, including wikis reveals some common limitations; namely: technical limitations, content management and portability problems, and most importantly, oversimplified knowledge representation.

When it comes to the the actual management of the content the main problem is, that CMS systems provide limited content portability. As long as content is simply data, such as HTML, PDF documents, pictures and so-on, CMS acts as a repository. When it comes to sharing information about the content, meta-data, or meta-knowledge, CMS systems do not provide appropriate facilities. This is mainly related to the lack of some common knowledge representation standards, while focusing on low-level encoding, and visual presentation of content.

From a knowledge management point-of-view, the crucial problem with CMS systems is, that they escape the knowledge representation and management pitfalls by assuming some predefined CMS structure, e.g. a portal. In this sense a CMS is only a set of modifiable portal templates. It is difficult to choose knowledge representation for current CMS solutions, because the knowledge they store can be considered implicit, or even hidden.

This criticism implicitly assumes that what CMS were designed to manage, and make available, is somehow knowledge. That would put the process of deploying a CMS in the context of knowledge engineering. In this field concepts such as knowledge representation, management, or validation are used referring to common tasks. However, considering how most of CMS are built, they cannot be considered Knowledge Management Systems. Turning current CMS into knowledge-oriented CMS should involve an introduction of number of features.

There seems to be generic areas that should be supported by any "knowledge server"; these are: knowledge representation and organization support; the system should provide appropriate structures and languages, knowledge processing and inference; the system should be capable of multi-paradigm reasoning and automated operations on knowledge, inference and operation control, user support, and operational decision making, knowledge acquisition support and extraction of knowledge from different sources, contextual user interface - both for operational use and administration/design, truth-maintenance, verification and validation support, learning and optimization.

There are numerous efforts to meet these requirements. However, because of lack of consistent and uniform theoretical foundations, efforts oriented towards building a knowledge server replacing, subsuming and covering database services, web applications and decision processes, are far from being satisfactory. Recent developments in the area of intelligent web authoring and collaboration tools include the development of semantic wiki systems. They offer the functionality of semantic annotation of the content. The next stage is provided by knowledge wikis, which add explicit knowledge representation and processing capabilities with the introduction of decision rules and trees.

5 Semantic Knowledge Wikis

A first step in the direction of enriching standard wikis with the semantic information has been performed by the introduction of the so-called *semantic wikis*, such as the IkeWiki [1], or OntoWiki [2]. In such systems the standard wikitext is extended with the semantic annotation. Such annotations allow for building an ontology of the domain with which the content of the wiki is related. This extension introduces not just new content engineering possibilities, but also semantic search and analysis of the content. In order to provide annotations semantic wikis allow for RDF or OWL annotations.

However, from the knowledge engineering point of view expressing semantics is not enough. In fact a knowledge-based system should provide effective knowledge representation and processing methods. In order to extend semantic wikis to knowledge-based systems, the concept of *semantic knowledge wikis* has been introduced, see [3,4]. An example of such a system is the *KnowWE* semantic knowledge wiki [3,5]. In such a system the semantic knowledge is extended with the problem-solving domain-specific knowledge. The system allows for introducing knowledge expressed with decision rules and trees related to the domain ontology. So conceptually it is built on top of the semantic wikis.

The approach presented in this paper shares goals with the semantic knowledge wikis. However, it differs with respect to the methods used. In the paper a generic solution based on the use of Prolog as the language for expressing both the semantics, and the knowledge processing information is presented.

6 Prolog Knowledge Representation Model

The Prolog language [6,7] is a prime example of applied programming in logic, where knowledge representation and processing is critical. Knowledge is represented with use of facts and rules as Horn clauses in First-Order Predicate Calculus. This formal representation intuitively corresponds to statements in the natural language. The language offers powerful knowledge processing capabilities, including recursive processing, unification and resolution. The flexibility of this approach, and the extensibility of the language made it an AI tool of choice for many systems.

Prolog natively uses a backwards chaining reasoning strategy, that corresponds to logical abduction. However, thanks to its meta-programming facilities it is easy to build any Prolog-based interpreter for a custom reasoning, including forward chaining.

Knowledge is expressed as clauses being facts or rules, often referred to as the knowledge base. Clauses constitute predicates. The way how facts are stated is close to the natural language, with a verb and a set of nouns being predicate name and predicate arguments respectively. Let us assume that there are some facts regarding student project management: there is a student Frank, a teacher Bob, both of them work on project `knowledgeWiki`, and the project is already finished (`done`).

```

is_student(frank).
is_teacher(bob).
works_on_project(frank, knowledgeWiki).
works_on_project(bob, knowledgeWiki).
project_status(knowledgeWiki, done).

```

Rules are expressed in a similar way with use of Predicate Calculus which introduces a concept of variable (indicated as capitalized identifiers). Finding out which student needs to be graded can be formulated as:

```

need_grade(Who, Project) :-
    is_student(Who), works_on_project(Who, Project),
    project_status(Project, done), \+ grade(Who, Project, _).

```

which can be read as: `Who` needs a grade from `Project` if `Who` is a student and he works on `Project`, and the project status is `done` and `Who` does not have a grade assigned (`\+` means negation).

Combining facts and rules gives a very powerful and expressive technique for storing knowledge. It is far more capable than simple representations such as RDF, which defines facts only.

Furthermore, Prolog allows to specify a goal to the inference engine to actually query the knowledge base. To find out who works on the `knowledgeWiki` project a query could be issued:

```

?- works_on_project(Who, knowledgeWiki).

```

Prolog will try to unify a solution with `Who` variable giving the answer. Queries are just like complex clauses, they can be integrated with the knowledge base. The following displays on standard output who works on `knowledgeWiki`:

```

:- works_on_project(Who, knowledgeWiki), write(Who).

```

Prolog-based knowledge representation can be possibly far richer than the ones used in the current knowledge wikis implementations. In the next section a new proposal of a knowledge wiki is introduced. It allows for a direct integration of Prolog code with the wikitext.

7 Prolog Knowledge Wiki Proposal

A simple concept for a generic knowledge wiki is to integrate Prolog code into the wiki text. Such Prolog annotations or attachments would enhance the content with semantics, as well as provide an inference technique. This idea was first proposed in [8].

The basic functionality should at least include:

1. ability to include Prolog code into wikitext,
2. spawning the Prolog inference engine in order to interpret the code embedded in wiki pages,

3. possibility of analyzing wiki contents by the Prolog program.

This would ultimately mean that a bidirectional Wiki-to-Prolog interface has to be developed.

The first requirement seems to be easy to fulfill. Considering how wikitext is built, a special markup for the Prolog code can be provided. A function to interpret larger Prolog files included as files in the wiki can also be considered.

The second one needs some low-level runtime integration, where the web-server is able to spawn custom process. A Prolog interpreter would be run, with Prolog-related wiki contents provided.

It should be possible to access and analyze the contents of the wiki from within the Prolog code. A special library of predicates that access the wiki, using wiki-specific references should be provided.

Another issue are the actual use scenarios for the system. Several possibilities have to be taken under consideration:

- enhance wiki with some *meta-knowledge*,
- describe the *meaning* of wiki contents with Prolog,
- provide knowledge *processing* for the wiki.

Common vocabulary, predicate names, their arities, and meaning should be defined as well. It would give a common meaning to the knowledge gathered in the wiki forming an ontology. Applying OWL or RDFS at this point should be evaluated. The proposed enhancement addresses issues presented in Sect. 4.

8 System Implementation

A preliminary implementation of concepts described in the previous section has been carried out within two students projects in the Knowledge Engineering Methods class of 2008 at AGH UST and a master's thesis research, see [9]. The prototype is based on the DokuWiki system. The implementation uses a dedicated Prolog plugin for DokuWiki. The plugin meets requirements 1. and 2. (see Sect. 7), being able to interpret Prolog code embedded into wiki, as well as spawning the Prolog interpreter.

The Prolog code is included using a simple construct:

```
wikitext

<prolog>
color(red).
</prolog>

wikitext
```

This instructs the wiki, that the contents should be parsed by the Prolog plugin.

In order to run the code, thus spawn the Prolog interpreter, a Prolog goal should be explicitly stated with the plugin invocation, e.g.:

wikitext

```
<prolog goal="color(X),write(X),nl,fail.">
color(red).
color(green).
color(blue).
</prolog>
```

wikitext

Input output issues are resolved as follows. Knowledge gathered in the wiki is treated as input, there is no explicit user input of any kind (no forms, input fields etc.). This knowledge is subject to processing in the way specified above. Prolog standard output is rendered in the page in the place where the plugin invocation is present. Thus, the above example is seen as a wiki page as follows:

wikitext

```
red
green
blue
```

wikitext

since `write/1` generates variable value and `nl/0` a new line on standard output. In this very case `fail/0` forces backtracking, which makes `X` to be resolved with `red`, `green`, `blue` values in turn.

The interpreter can access selected areas of the wiki, using an explicit scope specification, such as:

```
<prolog goal="color(X),write(X),nl,fail." scope="pl:miw">
<prolog goal="female(X),write(X),nl,fail." scope="pl:miw:proj.*08">
```

In the first case the contents of the whole wiki namespace¹ is parsed (all documents within `pl:miw` namespace). In the second case a number of pages matching the given regular expression are parsed (which are pages within `pl:miw` namespace which names start with `proj.` and end with `08`). The contents of all the pages are concatenated into a single Prolog file and then analyzed by the interpreter.

Another solution allows for using Prolog files stored in wiki, or outside it, accessible with an URL. This is especially useful in case of larger files e.g.:

```
<prolog goal="female(X),write(X),nl,fail" file="pl:miw:test.pl">
<prolog goal="female(X),write(X),nl,fail" url="http://sth.org/test.pl">
```

The current implementation is a proof-of-concept prototype. It does have some limitations, including:

¹ The concept of a namespace is typical to DokuWiki, however it is also present in other wiki systems, particular syntax indicating a namespace differs between systems.

- scope specification – it is not entirely compatible with the default DokuWiki namespace handling,
- full input/output from the Prolog program – currently it is limited to explicit output handling encoded in the `goal` specification for the plugin,
- flexible goal specification – in general one can use the `:-` expression anywhere in the Prolog file, to provide a goal, with the current version of the plugin this not always works as expected,
- caching – this poses some important problems related to both functionality as well as efficiency; in some cases it is desirable to override or disable the DokuWiki rendering cache to get results generated by the Prolog engine; on the other hand a separate caching mechanism for the Prolog code itself is provided, however, it does have certain limitations,
- unidirectional interface – currently it is only possible to invoke Prolog interpreter from the wiki; there is no way to actually query the wiki from Prolog,
- there is no syntax checking of Prolog code – it is possible to enter syntactically incorrect code without any notification about it,
- there is no editing support such as code completion.

See <https://ai.ia.agh.edu.pl/wiki/prolog:prologwiki> for project progress as well as testing area. There is also an ongoing master's thesis research carried out in this domain.

9 Use Example

The examples discussed here present some basic use for the plugin. They are also available online, see <https://ai.ia.agh.edu.pl/wiki/prolog:prologwiki>.

```

===== Smith Family =====
There is Kate and Liz obviously girls, and a couple of boys: Tom and Bob.
Kate is Bob's mom, while Tom is his dad.
Tom is also Liz's dad.
<prolog scope="family">
female(person(kate,smith)).
female(person(liz,smith)).
male(person(tom,smith)).
male(person(bob,smith)).
parent(person(kate,smith),person(bob,smith)).
parent(person(tom,smith),person(bob,smith)).
parent(person(tom,smith),person(liz,smith)).
</prolog>

```

Figure 1. `smith_family`: Smith family description

Expressing family relationships is a good example for using knowledge wiki. An example description of Smith Family as `smith_family` page, is given in Fig. 1.

It defines both a description of the family (clear wikitext) and corresponding, formal definition with use of `female/1`, `male/1` and `parent/2` predicates. Furthermore there is a scope defined, which interprets knowledge gathered in `family` page, see Fig. 2. These are additional, common rules allowing to infer family relationships such as maternity or paternity.

```
=====  
Family Definitions =====  
=====  
Father and Mother =====  
A parent and a female is somebody's mother.  
A parent and a male is somebody's father.  
<prolog>  
mother(X,Y) :- parent(X,Y),female(X).  
father(X,Y) :- parent(X,Y),male(X).  
</prolog>
```

Figure 2. `family`: Common definitions regarding families

```
=====  
Families =====  
Knowledge about all families is gathered in this namespace.  
<prolog scope="*_family">  
</prolog>
```

Figure 3. `families`: Gathering knowledge about all families

There could be another page (`families`) which gathers knowledge about all families present in the wiki (within current namespace), see Fig. 3.

10 Perspectives for Future Work

This paper presents an original idea of implementing a semantic knowledge wiki using Prolog for knowledge representation and processing. In the paper a proof-of-concept prototype implementation for the DokuWiki system is described. Since the current prototype has number of limitations, there is an ongoing effort to extend this idea.

Embedding predicate logic in terms of Prolog code into a wiki system delivers a powerful tool for expressing and processing knowledge. However, there are some constraints needed to make knowledge exchangeable and reusable. A similar approach can be observed in the AceWiki [10] project (<http://attempto.ifi.uzh.ch/acewiki>). It provides a semantic wiki that uses a controlled natural language called ACE. Compared to most other semantic wikis it does not use RDF or OWL directly, the semantics is contained directly in the wiki text and not in form of annotations.

A vocabulary should be defined which establishes a relationship between predicates and their meaning. Knowing a list, probably structured, of predicates and their meanings, gives a basis for using, reusing and extending them. Similarly, there should be a clear mechanism for defining such relationships. Existing technologies such as ontologies should be considered, as well as these close to the Prolog language, such as self-documenting help mechanism. There are three directions visible now: an OWL or RDF based ontology, a wiki-based one, or a purely Prolog-based one.

Current version of the prototype does not introduce Prolog debugging features. It is not possible to check the syntax of the Prolog code, or assist the user in entering it. In the future both debugging and syntax highlighting features are planned. And extended interface that uses automatic hinting could also be provided.

There are obviously some performance issues regarding knowledge processing within a wiki system based on Prolog code interpretation. Extracting knowledge from many pages and processing it could be a time consuming operation. Some smart caching techniques should be used then. A research regarding this issue has already been started. It is based on the caching mechanism present in the DokuWiki system.

As it is described in Sect. 8 a goal for the inference engine could be specified at the `prolog` tag. This approach seems to be a little redundant since there is already a Prolog built in mechanism to state a goal, integrating it with a knowledge base, with use of `:-`, see Sect.6. Application of either one and their semantical differences will be researched.

Using scope attribute can lead to some ambiguity. If there are many `prolog` elements within a page with different scopes, a global scope of the page is a sum of all the scopes. Alternatively there could be no such a concept as the page global scope. In this case scopes will be applied on element basis. Alternatively a scope could be defined with use of a special predicate instead of an attribute.

The proposed Prolog-based knowledge wiki is not compatible with the semantic annotation mechanism found in semantic wikis. These include the use of technologies such as OWL or RDF. There should be some bridging interface allowing for such cooperation and knowledge exchange established.

One of the promising applications of the proposed technology is to provide a learning environment for students taking Artificial Intelligence courses. The knowledge wiki gives a chance to learn Prolog without using other technologies than a web browser. Furthermore, even complicated knowledge repositories with distributed facts and rules spanning over many wiki pages, with multiple users and cooperative user interaction, can be build this way. It suits very well demonstration purposes as gathered knowledge can be put to work immediately.

References

1. Schaffert, S.: Ikewiki: A semantic wiki for collaborative knowledge management. In: WETICE '06: Proceedings of the 15th IEEE International Workshops on Enabling

- Technologies: Infrastructure for Collaborative Enterprises, Washington, DC, USA, IEEE Computer Society (2006) 388–396
2. Auer, S., Dietzold, S., Riechert, T.: Ontowiki - a tool for social, semantic collaboration. In Cruz, I.F., Decker, S., Allemang, D., Preist, C., Schwabe, D., Mika, P., Uschold, M., Aroyo, L., eds.: International Semantic Web Conference. Volume 4273 of Lecture Notes in Computer Science., Springer (2006) 736–749
 3. Baumeister, J., Reutelshoefer, J., Puppe, F.: Knowwe: community-based knowledge capture with knowledge wikis. In: K-CAP '07: Proceedings of the 4th international conference on Knowledge capture, New York, NY, USA, ACM (2007) 189–190
 4. Reutelshoefer, J., Baumeister, J., Puppe, F.: Ad-hoc knowledge engineering with semantic knowledge wikis. In: Proc. of SemWiki 2008 - The Wiki Way of Semantics, Workshop co-located with the 5th European Semantic Web Conference, Tenerife, Spain (2008)
 5. Baumeister, J.; Puppe, F.: Web-based knowledge engineering using knowledge wikis. In: Proc. of the AAAI 2008 Spring Symposium on "Symbiotic Relationships between Semantic Web and Knowledge Engineering", Stanford University, USA (2008) 1–13
 6. Bratko, I.: Prolog Programming for Artificial Intelligence. 3rd edn. Addison Wesley (2000)
 7. Covington, M.A., Nute, D., Vellino, A.: Prolog programming in depth. Prentice-Hall (1996)
 8. Nalepa, G.J., Wojnicki, I.: Concept of an interactive web portal for teaching prolog. In: FLAIRS2008. (2008) accepted.
 9. Kotra, M.: Dokuwiki plugins for graphviz and prolog. Knowledge Engineering Project (MIW), G. J. Nalepa supervisor (2008)
 10. Kuhn, T.: AceWiki: Collaborative Ontology Management in Controlled Natural Language. In: Proceedings of the 3rd Semantic Wiki Workshop, CEUR Workshop Proceedings (2008)

Learning to Cope with Critical Situations - An Agent based Approach

Régis Newo and Klaus-Dieter Althoff

University of Hildesheim, Institute of Computer Sciences,
Laboratory of Intelligent Information Systems
Email: newo|althoff@iis.uni-hildesheim.de

Abstract. How does someone react when he faces a critical situation in his life? In this paper we present an initial initial implementation architecture based on a simulation model described in [10]. In our model we mainly consider the interactions between a person concerned and factors like his environment and his own abilities. Using the empolis information access suite, we currently implement our model by means of a multiagent system approach, realized by distributed knowledge-based systems with a specific focus on case-based reasoning technology.

1 Introduction

In our everyday life, we consistently face situations which pose more or less immense challenges. Examples can be the breakup with a partner, the loss of a job, an illness or even the death of a relative. As different as those challenges can be, the reactions of the persons who are facing the same kind of challenges can be very different as well. The problem consists in finding out, how someone reacts when he/she faces up a given challenge. The problem being a psychological one, there have been many research groups in psychology working in that direction, beginning in the early 1980s. They developed psychological models and paradigms in order to represent and analyse people's behaviours as well as theories, software-based models, and simulation approaches.

In this paper, we present an agent-based approach for the representation and simulation of human behaviours in critical situations. For this purpose we developed - in cooperation with Werner Greve (Institute of Psychology, University of Hildesheim¹) - the SIMOCOSTS (SIMulation MODEL for COPing STRategy Selection) model. In the SIMOCOSTS project we are actually aiming at a threefold goal, namely (1) developing a research software tool for supporting psychologists, who are working on cognitive modelling and learning as roughly described above, in their research work, (2) realizing what we call "collaborative multi-expert-systems" (CoMES; see below), and (3) instantiating the SEASALT software architecture we developed in our research lab as a first step towards realizing CoMES. In this paper, we elaborate on how we currently intend to implement our simulation while focussing on the representation of the needed

¹ http://www.uni-hildesheim.de/psychologie/mitglieder/werner_greve.htm

knowledge.

In the next section, we will shortly introduce CoMES and SEASALT and discuss related work. We describe the SIMOCOSTS model, its functionality, the developed knowledge representation and processing in Section 3, and the status of its implementation in Section 4. Finally in Section 5 we give a short outlook on relevant future work.

2 Background and related work

In this section we shortly explain the underlying CoMES approach and its first instantiation via the SEASALT architecture. Related work from the areas of cognitive architectures, coping processes, and other related psychological areas can be found in [10] and [9].

2.1 Collaborative Multi-Expert-Systems

Collaborative Multi-Expert-Systems (CoMES, see also [2]) denote a new research approach that is both, a continuation of the well-known expert system approach and a research direction based on the ideas of case factory and knowledge-line [3]. In the Knowledge-line concept we systematically apply the software product-line approach [11] from software engineering to the knowledge of knowledge-based systems. This enables the necessary "knowledge level modularization" for building potential variants in the sense of software product-lines. The modularization can be achieved by making use of multi-agent systems [6, 12] as a basic approach for knowledge-based systems. An intelligent agent - as a first approximation - is implemented as a case-based reasoning (CBR) system [1], which, besides case-specific knowledge, can also include other kinds of knowledge. Each CBR agent is embedded in a case factory [3] that is responsible for all necessary knowledge processes like knowledge inflow, knowledge outflow as well as knowledge analysis.

While many early (and also some current) expert systems had the problem of acquiring and maintaining their knowledge, the underlying idea in CoMES is to "develop CoMES where knowledge is produced". Another idea is to keep the resulting learning scenarios/tasks as simple as possible, thus having more agents and having each one learning in a rather simple way.

2.2 Sharing experience using an agent based system architecture layout

A first step towards realizing the CoMES approach is the SEASALT (Sharing Experience using an Agent based System Architecture Layout) architecture. The architecture can be vertically split in two parts as can be seen in Figure 1. On the left hand side the knowledge provision and on the right hand side the knowledge acquisition. For the current stage of the SIMOCOSTS project we focus on the knowledge provision part only (a more detailed description of

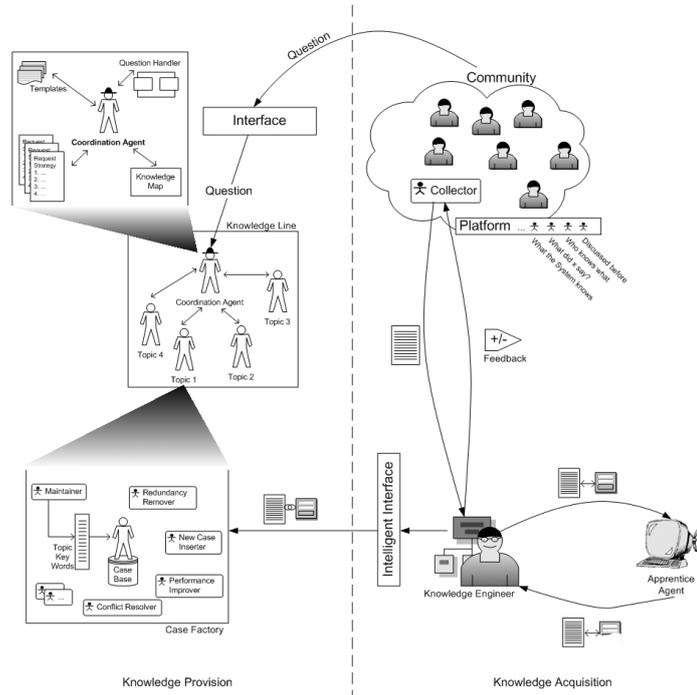


Fig. 1. The SEASALT Architecture

SEASALT is given in [4]). If a user enters a question using the Interface, it passes the question on to the Coordination Agent. The Coordination Agent analyzes the question, looks up the matching Topic Agent(s) and sends its requests to them. A response based on the existing case base is created by each Topic Agent and passed back to the Coordination Agent. Finally, the response of the Topic Agents is used by the Coordination Agent to compile an answer.

3 A Simulation Model for Coping Strategies

SIMOCOSTS (Simulation Model for Coping Strategy Selection) is the underlying model for our simulation. The model is based on the psychological theories developed by Brandstädter and Greve [5]. One main difference between our simulation approach and other ones consists in the fact that all the other view the respective persons as agents. But we intend to represent a person with many agents while following the holonian concept [8] in order to have a detailed and agent-based representation of each individual. A detailed picture and description of the model can be found in [9] and [10]

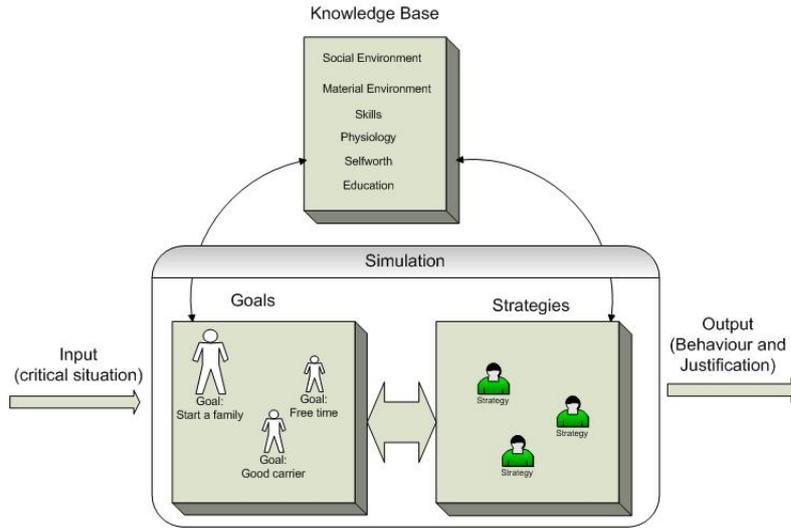


Fig. 2. The Implementation Architecture of SIMOCOSTS

4 An Implementation Architecture for the Simulation of Coping Strategies

We developed the model mentioned in the previous Section with a main focus on the processes needed for the simulation. The main drawback of that model is that it is not suitable for an (initial) implementation. We thus present in this Section the implementation architecture of our simulation tool. We want to start with a rather simple architecture which will be later expanded, because of the complexity of the task. The main idea of our architecture is based on the fact that each person has some goals that he wants to achieve (see [10]). In our scenario, a critical situation occurs when there exist some facts that prevent the person from reaching those goals.

As we know, human acts (especially while loosing problems) is mostly based on past experiences. For our purpose, the achievement of goals as well as the general knowledge that will be used for the achievement of the goals will be based on past experiences. That is why we will make use of the case-base reasoning technology in our implementation. Furthermore, we will implement the goals by using the so called practical reasoning agents paradigm [13], which is based on the Belief-Desire-Intention (BDI) principle.

Our architecture (see Figure 2) consists of the three main following parts.

Knowledge Base The knowledge base consists of all the general knowledge, that can be helpful while loosing the problem. That knowledge include skills, material and/or social environment, etc. We plan to use many differents case

bases for the different case bases for the distinct parts of the general knowledge needed (e.g. skills).

The Strategies In our architecture, the strategies represent the actions (in analogy to BDI agents) that can be used for the computation of the plan in the means-ends reasoning stage. These actions mostly have an impact on the knowledge base defined earlier (e.g. the acquisition of a new skill) as well as on the internal goals (i.e. adaptation of the goals). We plan to implement those strategies as rules in a case-base reasoning system.

The (internals) Goals The initial goals of the person are the initial beliefs of the agents which are used for the computation of the intentions when a critical situation occurs. Each agent is responsible for analyzing if its goals are still reachable (i.e. there is no critical situation). , we will implement the goals as cases in a case-base reasoning system.

When a situation is judged as critical, each affected goal try to find out how it can be achieved. The achievement is done by using the strategies, which are based on the general knowledge of the person. Actually we started to build, using CBR, a knowledge base needed for a specific example (i.e. breakup of a partner). This has to be very sound in order to have a plausible simulation. We plan to use the Information Access Suite of empolis [7] GmbH for the realization of our architecture, because it is a powerful tool which gives us the possibility to handle case bases as well as rules.

4.1 Classification of the Architecture

Our Architecture follows the principle of the CoMES approach introduced in Section 2.1 and leans on the SEASALT architecture which we presented in Section 2.2. We have a knowledge line in our implementation architecture which contains the three parts presented above. In fact, the knowledge line in our architecture can be seen as all the informations needed to represent a person. We thereby achieve the reusability which is important while developing a knowledge line in terms of CoMES.

As for the similarities with the SEASALT architecture, we will also have a community which will consist of experts in the area of psychology. The goals can be seen as the topic agents in SEASALT and we also a knowledge engineer whose task will be the acquisition of the information needed for a person. We also have a distributed architecture because it is based on the CoMES approach.

5 Outlook and Conclusion

In this paper, we presented an architecture for the implementation of the simulation of coping processes. After the introduction of the CoMES approach and the SEASALT architecture, we presented our an implementation architecture

based on the SIMOCOSTS model. Our implementation will be based on two main technologies, namely case base reasoning and multi-agent systems, while following the CoMES approach.

Further work include an accurate specification of the knowledge base an its implementation as well as the implementation of strategies and goals for given examples.

References

1. Klaus-Dieter Althoff. Case-based reasoning. In S.K. Chang, editor, *Handbook on Software Engineering and Knowledge Engineering. Vol.1, World Scientific*, pages 549–587. 2001.
2. Klaus-Dieter Althoff, Kerstin Bach, Jan-Oliver Deutsch, Alexandre Hanft, Jens Mänz, Thomas Müller, Régis Newo, Meike Reichle, Martin Schaaf, and Karl-Heinz Weis. Collaborative Multi-Expert-Systems – Realizing Knowlegde-Product-Lines with Case Factories and Distributed Learning Systems. In J. Baumeister and D. Seipel, editors, *Accepted for Proc. 3rd Workshop on Knowledge Engineering and Software Engineering (KESE 2007), Osnabrück, Germany*, Berlin, Heidelberg, Paris, 2007. Springer Verlag.
3. Klaus-Dieter Althoff, Alexandre Hanft, and Martin Schaaf. Case factory – maintaining experience to learn. In Mehmet H. Göker, Thomas Roth-Berghofer, and H. Altay Güvenir, editors, *Proc. 8th European Conference on Case-Based Reasoning (ECCBR'06), Ölüdeniz/Fethiye, Turkey*, volume 4106 of *Lecture Notes in Computer Science*, pages 429–442, Berlin, Heidelberg, Paris, 2006. Springer Verlag.
4. Kerstin Bach, Meike Reichle, and Klaus-Dieter Althoff. A Domain Independant System Architecture for Sharing Experience. In Alexander Hinneburg, editor, *Proceedings of the LWA 2007: Lernen - Wissen - Adaptivität*, pages 296–303, September 2007.
5. J. Brandtstädter and W. Greve. The aging self: Stabilizing and protective processes. *Developmental Review*, 14:52–80, 1994.
6. Hans-Dieter Burkhard. Software-agenten. In Günther Görz, Claus-Rainer Rollinger, and Josef Schneeberger, editors, *Handbuch der Künstlichen Intelligenz, 4. Auflage*, pages 943–1020. Oldenbourg, 2003.
7. empolis GmbH. Technisches white paper e:information access suite. Technical report, empolis GmbH, September 2005.
8. S. Glückselig. Holonische Multiagentensimulation. Master’s thesis, Universität Würzburg, 2005.
9. Thomas Müller. Simulation kognitiver Prozesse mit Multiagentensystemen. Master’s thesis, Universität Hildesheim, 2006.
10. Régis Newo, Thomas Müller, Klaus-Dieter Althoff, and Werner Greve. Learning to Cope with Critical Situations - A Simulation Model of Cognitive Processes using Multiagent Systems. In Alexander Hinneburg, editor, *Proceedings of the LWA 2007: Lernen - Wissen - Adaptivität*, pages 159–164, September 2007.
11. Frank van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering*. Springer, Berlin, Heidelberg, Paris, 2007.
12. Gerhard Weiß, editor. *Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence*. The MIT Press, 1999.
13. Michael Wooldridge. *An Introduction to MultiAgent Systems*. John Wiley & Sons, June 2002.

On Restaurants and Requirements: How Requirements Engineering may be Facilitated by Scripts

Christoph Peylo

Deutsche Telekom Laboratories, Berlin
christoph.peylo@telekom.com

Abstract. Requirements engineering is a central part of software projects. It is assumed that two third of all errors in software projects are caused by forgotten requirements or mutual misunderstandings in the requirement gathering process. Due to the inherent structure of project planning and the project management process, it is very unlikely that this problem will be solved unless the process itself is changed or we develop tools that possess some intelligence to facilitate the assessment of requirements. In this paper a position for the latter approach is formulated. It is argued that it is feasible to establish a domain ontology based on meta information and explanations that are represented as scripts. It is shown that this ontology has to be constructed in a dynamic way to reflect the dynamics of the requirements engineering process. Finally, it is sketched how use cases and test cases can be derived from this ontology.

1 The Difficulty of Creating Novel Things in Time and Budget

A project is a temporary endeavour undertaken to create a unique product, service or result [1]. Consequently, a project has a limited time frame and creates unique deliverables like services, products or results. The success of a project depends on the meeting of the outcome with its expected result. Thus, the deliverables of a project have to fit in their intended niche.

Accordingly, identifying the project requirements is of uttermost importance for any project. The project requirements describe the characteristics, i. e. conditions or capabilities, that must be met by the deliverables [1]. Further steps of project management such as establishing objectives, balancing the demands for scope, time, cost and quality, etc., have the clear and comprehensive definition of requirements as a prerequisite.

Hence, the identification of requirements (requirements engineering) is critical for the success or failure of a project. It is assumed that two thirds of all errors in system development are caused by forgotten requirements or misunderstandings.

1.1 Project Requirements and Project Risks

The understanding of project requirements must be reached (according to the classical project management approach) in the very early stages of the project (initiation stage), in order to formulate the project scope statement.

The project scope statement describes the project's deliverables and the work that is to be accomplished to create them. Frequently, it is quite a coarse grained level of detail in which the conditions and capabilities of the intended system are described at that time. Nevertheless, this will build the basis for further planning during the next stages.

From a business point of view, this is a quite disadvantageous setting: the gathering and analysis of the key project deliverables and the decision of how to supply them is in a project phase before there is a signed contract. Consequently, the time and work in this phase is not paid by the intended customer-to-be. It is quite common in large projects to get compensation for feasibility studies, but this does not apply to all projects.

In summary, if this stage is not performed well, it is unlikely that the project will be successful in meeting the expectations of the stakeholders.

2 On the Difficulty to Represent Understanding with Imprecise Formalisms

To understand the project requirements it is important to understand the background, i.e. the business processes from which the need of a new product or service has arisen. This helps to understand the stakeholder's needs, wants and expectations and what has to be achieved to satisfy a contract.

Alas, the setting of such a task is often quite complex. The general background is often quite specialized and not easy to understand, business processes are sometimes ill defined. The terminology used by the stakeholders may differ from the vocabulary of the requirements engineer or, even worse some terms may be used with a slightly different semantics (which usually becomes apparent later in the project). Last but not least, some terms may turn out to be not terms at all but business processes (cf. [2]).

Thus, there are numerous and well known reasons (cf. [3]) why requirements are left out or not fully understood:

- Business processes are ill defined. A business process may consist of several sub-processes which may be too trivial for the stakeholders to mention.
- Terms are used with (slightly) different semantics.
- Business processes of a new (innovative) setting are ill defined.
- Novel business processes may interfere with existing business processes.
- There are contradictory processes involved.

It would be necessary to describe the system and its background in a comprehensive and almost complete way to eliminate all sources of those mistakes. There are hardly any projects where there is time and budget for such a comprehensive

approach. It is important to remember that in the initiation phase of a project a contract is not signed, i. e. there is no or insufficient compensation for an in-depth approach available.

Consequently, it is quite common to concentrate on the representation of the functional requirements with use cases. A use case describes the interaction between a system and a request that originates from outside of that system. Use cases represent that interaction as a sequence of single steps and events to achieve a specific goal. There are several representation schemes (most common: the UML use case diagram in various versions) to graphically express this kind of interaction. The meaning (or semantics) of the use case is not represented by the well defined building blocks of the formalism [4], [5], but shall constitute itself (helped by various annotations) in the mind of the reader. This approach is quite common but prone to misunderstandings.

Admittedly, those representation formalisms have a certain beauty: they represent complex interactions in a compact way that may be perceived quickly (at least in comparison to lengthy (and often tiresome to read) definitions in natural language). Due to their seeming clarity and formality they are often over-estimated. Nevertheless, they are deceptive with respect to their precision and expressiveness. Their main limitations are:

1. Weak and not well defined semantics of relations.¹
2. The expressiveness of graphical representation schemes is limited per se to a fragment of first order logic (existential quantified, conjunctive connected). Trying to extend the symbology by annotations (to cover modal or second order constructs) will increase the confusion, not the expressiveness.
3. Use cases represent the interaction in the communication between user and system. Commonly, they refer to sub processes and documents that are interchanged during those process steps without explaining the content in full detail. Thus, generally, it is not possible to decide by the study of a use case whether the process flow may lead to the desired result (i. e. the system output may be achieved, given the set of input).

After this synopsis of the requirements gathering process and the difficulties that exist in avoiding misunderstandings it is concluded that either this process should be upvalued considerably (in terms of time and money), or tools for facilitating this process on a more semantic level should be applied.²

In the remaining part of this paper, an approach is sketched how existing AI concepts could be deployed for this purpose.

¹ This is no new insight, as shown by Woods [6].

² There are several approaches that try to support the requirements engineering process in linking use cases to contextual scenarios (e. g. [7] and [8]). But the representation of scenarios is done in natural language and suffers from the known problems connected with that approach (cf. sec. 2 and sec. 4).

3 Contributions from AI

How can AI facilitate the requirements engineering process, and, more specifically, how can AI contribute to avoid misunderstandings? From an AI perspective, the problem is situated in the context of formalizing domain knowledge and to explain and to communicate this knowledge (cf. [9], [10]).

3.1 On Explanations and Understanding

Explanation can be regarded as the process by which we make sense of the world [11]. Thus, we construct structures from which knowledge can be derived at a later stage. Explanations seem to be linked to the process where knowledge structures are constructed or communicated. Thus, explanations connect phenomenon in a systematic way to make outcomes predictable. Whereas explanations reflect the process of understanding, knowledge seems to be more about the management of realization. Thus, knowledge can be understood as a tertiary relation (someone assigns someone knowledge about something).³

At a very basic level explanation equals understanding: We believe that we understand why a person is doing what he is doing if we can point to a script that he or she is following [11]. A script is a structured representation describing a stereotyped sequence of events in a particular context and forms a very basic knowledge structure [11]. In that sense the famous restaurant script [13] was used to understand the basic interactions in a restaurant. This script-type of understanding is equivalent to *making sense* and is to be distinguished from the deeper (*cognitive*) understanding, of course. For the purpose of this paper *making sense* will do. Thus, scripts as condensed or compiled explanations may be considered as suitable building blocks for a system with a shallow degree of self-awareness.⁴

3.2 The Structure of a Script

As stated above, a script models a flow of action in a specific setting. To understand this setting, i. e. to interact in a limited communication [11], it is necessary to tag structural information to its building blocks to facilitate the semantic processing in a computer system. The components of scripts are

- Entry conditions that must be met before the script may be started.⁵
- Results or conditions that are true once the script has terminated. Thus, a script has a specific setting as an entry condition and after the script terminated, the setting (the state of affairs) is different from the initial setting.⁶

³ This holds true especially in educational contexts, see [12].

⁴ A script resembles goals and scenarios in the COSMOD-RE approach of Pohl [14]. But this approach is a methodology (cf. [15]) which does not result in a system as proposed here.

⁵ In the famous restaurant script these include a restaurant that is open and a customer that is hungry.

⁶ In the restaurant script effects of the script are that the customer is not hungry and has less money.

- Roles as placeholders for actors or objects in actions that the individual participants perform.
- Scenes that reflect temporary aspects of a script. A scene works like a script in a script. It encapsulates operations that change the state of affairs.
- The entities involved as objects or passive parts in that script.
- A set of well defined actions. Actions are distinguished by the arity of the relation (e.g. transitive verbs are a binary relation, double-transitive verbs a tertiary relation), and a type restriction with respect to roles and entities.

Those components offer the tools, by which a *lightweight* understanding may be modeled. Logically, entities may be modeled as predicates, actions may be represented as relations on roles. Roles are variables (with type restrictions) for entities. A state of affairs may be represented as a list of predicates that hold in that moment.

3.3 Example: the Restaurant Script

The classic example of Schank's theory is the restaurant script. The script theory is closely related to Schank's concept of *conceptual dependencies* [13]. According to that concept, the meaning of natural language sentences should be expressed by using conceptual primitives. In the example given below the conceptual dependencies are marked using uppercase letters and a typewriter font. The meaning of these primitives is as follows:

PTRANS: Transfer of the physical location of an object (i.e. go).
MBUILD: Building new information of old information (i.e. decide).
MTRANS: Transfer of mental information (i.e. tell).
ATRANS: Transfer of an abstract relationship (i.e. give).
MOVE: Movement of a body part by its owner.
ATTEND: Focusing of a sense organ toward a stimulus (e.g. listen).

The subject of the sentences is represented by **S** which is a role and can be instantiated by any agent. The script consists out of four scenes:

Scene 1: Entering: S PTRANS S into restaurant, S ATTEND eyes to tables, S MBUILD where to sit, S PTRANS S to table, S MOVE S to sitting position.
Scene 2: Ordering: S PTRANS menu to S (menu already on table), S MBUILD choice of food, S MTRANS signal to waiter, waiter PTRANS to table, S MTRANS 'I want food' to waiter, waiter PTRANS to cook.
Scene 3: Eating: Cook ATRANS food to waiter, waiter PTRANS food to S, S INGEST food.
Scene 4: Exiting: waiter MOVE write check, waiter PTRANS to S, waiter ATRANS check to S, S ATRANS money to waiter, S PTRANS out of restaurant.

It is not compulsory to adopt the concept of conceptual dependencies to utilize scripts. Nevertheless, it is necessary to define entities, roles and operations (i.e. actions) in a way, that offers some generality and transferability. Thus, a set of universals with predefined semantics and support for roles seems to be helpful.⁷

⁷ Further work will include an analysis how ongoing efforts on semantic modeling could be integrated in this approach (cf. [16]).

3.4 Semantic Expressiveness

As sketched above, a script transforms one state of affairs into another state. States may be modeled adequately with a fragment of first order logic (existential quantified, conjunctive connected predicates). Accordingly, entities, e. g. objects or actors, that are referred to in a script may be formalized as well by a set of attributes, i. e. as predicates. Generally, first order logic is not sufficient to model the dynamic interdependencies and actions in a domain, due to the necessity of modal, temporal or second order (quantification about predicates) constructs. These language constructs have to be provided by modeling the actions and roles accordingly. Thus, type restrictions and quantifications on predicates or attributes have to be considered in the process of defining and implementing roles.

Consequently, the static aspects (situations as constellations of entities at a given point in time) are modeled with a fragment of first order logic. Actions, as well as operations on entities, permit more advanced constructs like quantification on predicates and additional qualifiers. This augments the expressiveness of the whole formalism considerably. A script forms a context in which the semantics for at least one - and to avoid misunderstandings: exactly one - valid assignment and interpretation is provided.

This approach is computationally feasible, since the domain of the variables of the predicates are restricted in most application scenarios to reasonably sized sets of possible instantiations.

3.5 Scripts and Ontologies

Generally, a software system is intended to be representationally and inferentially adequate with respect to its application area. Thus, the entities in the software system and their real world counterparts shall be describable by the same attributes. Inferences over attributes and entities in the system shall hold in reality and vice versa. Thus, the conceptualization of the application domain in the software system shall model those entities, relationships and processes that are essential for achieving the intended level of adequacy. Such a conceptualization may be referred to as an ontology [17]. In this setting the role of an ontology is twofold. It shall represent the body of knowledge from which the deliverables of the system shall be derived and it shall provide the vocabulary and the rules from which the interactions with the system may be described (cf. [9]).

The ontology has to be dynamic: the project goals may be subject to change and therefore the underlying ontology respectively. Since a project is an unique endeavor we can not take some ontology from the shelf, but it is more likely that each project (even if located roughly in the same domain) will need its own ontology.

Such an ontology will be referred to as an *agreed* ontology to express that it shall represent the common understanding of the domain by all stakeholders of the project. The term *agreed* implies a certain dynamics as well in the process

of defining and refining the ontology. It reflects the processes as mutual understanding as the project group grows and implies that the formalism should be mighty enough to tackle well known problems with respect to knowledge bases (frame problem, non-monotone logic, etc.).

3.6 The Building Blocks of an Agreed Ontology

Accordingly, there have to be building mechanisms for both: scripts as constituents of an ontology and the ontology itself. Given the inherent structure of a script as outlined above it lies at hand that scripts can be defined by a context-free grammar. Basically, a context-free grammar has four components (cf. [18]):

- A set of terminal symbols. These are the elementary symbols of the language defined by this grammar.
- A set of nonterminal symbols or syntactic variables.
- A set of rules, where each rule consists of a nonterminal (head) and a sequence of terminals or nonterminals (body), by which the head may be replaced.
- A designation of one of the nonterminals as start symbol.

Applied to this context, it is evident that scenes, roles and actions form the nonterminal symbols of this grammar. The terminal symbols are either domains of the syntactic variables, such as instantiations of roles (i. e. a specific user or a specific entity) or outcomes of a script, i. e. a state of affairs.

This shall be illustrated with a scenario where a device fails, and calls for a technician. This scenario is taken from a setting that has been accomplished by the Deutsche Telekom Laboratories [19]. It is situated in a context where machine-to-machine techniques are deployed to automate facility management processes. This scenario is built up from several scenes. The scenes of the scenario have to be applied in a distinct order, thus the scenes are numbered.

1. A device fails. A notification is sent calling for the technician. His credentials are activated, so that he may enter the room where the device is located. A process is triggered which waits for the technician. If the technician has not arrived during an interval, the call is sent again.
2. The technician arrives at the building. The technician has to authorize himself with his credentials to be able to enter the room where the device is located. This will trigger another event. This event includes a success parameter, stating whether the door opened or not. This can generate an alarm, should the technician enter the wrong credentials.
3. During the repair process the device has to be queried several times. Since the technician is authorized, the conditions for a repair process are met and no further call is sent.
4. Once the device works again and the technician is finished the authorization lifespan will be ended. Again, an alarm is triggered if the technician fails to authenticate himself when leaving the location. The credentials are deactivated to ensure that the technician may enter the location only in the course of a repairing process.

The scenario is comprised of several actions that result in specific situations, i.e. events. The scenario is represented with a grammar in Backus-Naur form as given below. The terminal symbols are enclosed with quotes. Head and tails of the rules are separated by a ' ::= ', ',' is used as concatenation and ';' as termination symbol.

```

1. accessAndServiceControl ::= queryStateOfDevice;
2. queryStateOfDevice ::= device, deviceWorks;
3. queryStateOfDevice ::= device, deviceFails, isAuthorized,
   queryStateOfDevice;
4. queryStateOfDevice ::= device, deviceFails, callTechnician,
   waitingForTechnician;
5. device ::= "specificDevice";
6. deviceWorks ::= "deviceWorks";
7. deviceFails ::= "deviceFails";
8. callTechnician ::= technician, notifyTechnician;
9. technician ::= "specificTechnician";
10. notifyTechnician ::= "technicianNotified";
11. setCREDENTIALS ::= "techniciansCREDENTIALSActivated";
12. unsetCREDENTIALS ::= "techniciansCREDENTIALSDeactivated";
13. waitingForTechnician ::= openDoor;
14. waitingForTechnician ::= callTechnician, waitingForTechnician;
15. openDoor ::= arrivesTechnician, setCREDENTIALS, authenticate,
   isAuthorized;
16. soundAlarm ::= setCREDENTIALS, authenticate, isNotAuthorized;
17. arrivesTechnician = "technicianHasArrived";
18. authenticate ::= CREDENTIALS, isAuthorized;
19. authenticate ::= invalidInput, isNotAuthorized;
20. CREDENTIALS ::= "CREDENTIALS";
21. invalidInput ::= "invalidInput";
22. isAuthorized ::= "isAuthorized";
23. isNotAuthorized ::= "isNotAuthorized";
24. repairDevice ::= queryStateOfDevice, leaveRoom;
25. leaveRoom ::= authenticate, unsetCREDENTIALS;

```

The terminal symbols in this example are placeholders for specific instantiations of roles, e. g. a specific technician, location or device, or situations. A situation s is a configuration (cf. sec. 3.4) which holds true for all state of affairs Δ in the system at a given point t in time.⁸ Thus: $\Delta_t \models s \wedge \Delta_t \not\models \neg s$. It surely is a challenge to model the action and role part in a way that supports explanations and allows quantifications. Nevertheless, although the exact definition and modeling of actions and roles may be demanding in specific cases, it is considered that this does not present an obstacle in principle to this approach. Since there are several approaches documented and available, where this problem has been solved (c.f. [11]).

⁸ For example, Δ_t may be the set of all relations in a database system at a specific point of time t .

3.7 Constructing Agreed Ontologies

Scripts form the nonterminal vocabulary of an agreed ontology, the terminals are representations of outcomes of scripts. The challenge to define the rules for the ontology is quite demanding, since they model the order and interdependencies of scripts and the dynamics of the ontology depends on them.

The dynamics is achieved by adding new scripts to the ontology, removing and modifying existing scripts or changing the order of the scripts. This shall be sketched by extending the service scenario given above with further use cases from that domain.⁹

Access and Service Control: Maintenance personnel are given key (e. g. RFID tags, access cards) for accessing facilities and identification at devices to be maintained. Tags store employee credentials, doors to be used, work orders as well as operations carried out. The usage of doors is monitored and alerts are generated if needed.

Inventory Management: Every asset may have one or more unique identifier. This provides knowledge of the connected devices, their functionalities, and attributes. Automatic inventory of assets using fixed and handheld readers helps locating displaced and mobile assets. Absence of a reading event can be used to detect stolen equipment.

Predictive Maintenance: Continuous monitoring of operational (e.g. load) and non-operational (e.g. temperature) parameters using sensors to predict breakdowns. Estimate individual maintenance intervals for different equipments. Using maintenance history (data logging) to analyze tradeoffs between cost to maintain old equipment and investment in new equipment.

Remote Control: Remotely monitor and query about the status of individual persons (in terms of location) and devices. Devices shall be reconfigured remotely.

The script for access and service control was explained in detail in sec. 3.6. The other use cases may be represented as scripts in analogous way.¹⁰ A simple example grammar that models an ontology for this facility management setting is given below:¹¹

1. `FacilityManagementOntology ::= InventoryManagement, PredictiveMaintenance, RemoteControl, AccessAndServiceControl;`
2. `InventoryManagement ::= establishedAutoID;`
3. `PredictiveMaintenance ::= establishedPMPProcesses, remoteControl;`
4. `AccessAndServiceControl ::= establishedAASProcesses, remoteControl;`
5. `RemoteControl ::= "establishedRemoteControl";`

⁹ Additionally, it might be necessary to ensure global consistency of the outcomes of the different scripts. The approach of assumption based truth maintenance systems (ATMS) [20] can be deployed to solve this problem. A script of this approach roughly plays the role of an assumption in de Kleer's concept.

¹⁰ In this context use cases are subsumed by scripts.

¹¹ The grammar is represented in BNF, terminal symbols are enclosed with quotes.

It is obvious that the scenarios are not independent from each other but imply an order. Thus, to enable predictive maintenance remote monitoring has to be established. The granularity can be increased, by splitting the scenario of access and service control in two scenarios: an access control and a service control scenario. This leads to a change in the definition of the inventory management process:

1. `AccessAndServiceControl ::= remoteControl, AccessControl;`
2. `AccessControl ::= "establishedACProcesses";`
3. `InventoryManagement ::= AccessControl, establishedAutoId;`

It is possible to change a sequence or to edit the starting conditions or the outcome of a script. Thus, by manipulating the grammar the ontology of the domain of interest can be managed. In addition an ATMS could administer the consistency of the general state of affairs of the model of the application domain. Since each script affects the state of affairs of the whole system and the changes that have been applied by the operations of a script are recorded, each state of affairs can be tracked down to the scripts involved. Thus, it may be easily discovered, if starting conditions of a script never appear or if scripts lead to inconsistencies.

Since scripts not only model the interaction flow but are more detailed with respect to the semantics of the process and the outcome, scripts may even act as blue print for test cases. Consequently, this approach enables to transfer the methodology of test driven development to the requirement engineering process. As production code in this setting has to pass the predefined test cases, new requirements would have to be formulated as scripts and checked against the so far agreed ontology. Consequently, it can be decided for each requirement (that is well-formed in terms of the ontology's grammar) whether it may be derived, is subsumed, leads to contradictions or augments the set of requirements gathered so far.

4 Conclusion

In this paper a position was formulated that points out some general deficits in requirements engineering. It was argued that the tools to represent functional requirements of a non-trivial software system are commonly restricted in their semantic expressiveness and thus inept to establish mutual understanding concerning those processes. Misunderstandings will easily arise due to the ill defined semantics of the representation formalism: each stakeholder will underlay his or her individual semantic reference system for understanding. To describe the system's functional requirements in a comprehensive way in natural language is possible in theory but not feasible either. It is known from experience that large volumes are scarcely read by their target audience.

Thus, it is argued that this problem can be solved by an agreed ontology that models the understanding of the target system in a way that explanations on functional requirements can be given. This ontology should be implemented in a system that forms the semantic grounding of all assumptions about the

domain of interest and the interactions within the future software system. Misunderstandings and contradictions can be managed due to its semantic-enabled constituents, i. e. scripts, and the internal management of the system.

5 Consequences and Further Work

Although the argumentation in this paper did not provide detailed examples, it should be comprehensible, that this approach is technically feasible and will hold true. Future work will provide comprehensive examples and flesh out the approach.

Nevertheless, applying this approach to software projects will result in a major change in the administrative setting of a project and communication between the stakeholders that will hinder its acceptance. The biggest issue in that respect is that presently, the documentation of all relevant organisational stipulations is essentially paper-based. Utilizing this approach would mean to transfer an essential part of the project documentation, i. e. the written and signed requirements specification, to a different medium. The legal issues are no hindrance, since a system that implements this approach could be serialized and signed with the certificates of the stakeholders. Nevertheless, major shifts in administrative procedures are not done lightly. Thus, future work will have to prove that the benefit of this approach to software engineering will outweigh the inconvenience of changing an administrative process.

References

1. Project Management Institute, ed.: A guide to the project management body of knowledge: PMBOK guide. 3 edn. Project Management Institute, Inc. (2004)
2. Knauss, E.: Einsatz computergestützter Kritiken für Anforderungen. *Softwaretechnik-Trends* **27** (2007)
3. Wiegers, K.: *Software Requirements*. Microsoft Press (2005)
4. Jeckle, M., Rupp, C., Zengler, B., Queins, S., Hahn, J.: Uml 2.0 - Neue Möglichkeiten und alte Probleme. *Informatik Spektrum* **27** (2004) 323 – 332
5. Nalepa, G., Wojnicki, I.: Using UML for Knowledge Engineering - A Critical Overview. In Baumeister, J., Seipel, D., eds.: 3rd Workshop on Knowledge Engineering and Software Engineering (KESE 2007) at the 30th Annual German Conference on Artificial Intelligence. (2007) 37 – 47
6. Woods, W.: What's in a Link: Foundations for Semantic Networks. In Borow, D., Collins, A., eds.: *Representation and Understanding*. Academic-Press, New York (1975) 36–81
7. Sutcliffe, A.: Scenario-based requirements analysis. *Requirements Engineering Journal* **3** (1998) 48 – 65
8. Allmann, C.: Situations- und szenariobasierte Entwicklung von Anforderungen in der technischen Entwicklung. *Softwaretechnik-Trends* **28** (2008)
9. Walton, D.: Can Argumentation Help AI to Understand Explanation. *Künstliche Intelligenz* **2** (2008) 8 – 11
10. Richter, M.: Logik versus Approximation. *Künstliche Intelligenz* **4** (2004) 62 – 64

11. Schank, R., Kaas, A., Riesbeck, C.: Inside case-based Explanation. Lawrence Erlbaum Associates, Inc. (1994)
12. Peylo, C.: Wissen und Wissensvermittlung im Kontext von internetbasierten intelligenten Lehr- und Lernumgebungen. Volume 257 of Dissertationen zur künstlichen Intelligenz. Akad. Verl.- Ges. Aka, Berlin (2002)
13. Schank, R., Abelson, R.: Scripts, Plans, Goals and Understanding. Lawrence Erlbaum Associates, Hillsdale, New Jersey (1977)
14. Pohl, K., Sikora, E.: The Co-Development of System Requirements and Functional Architecture. In Krogstie, J., Opdahl, A., Brinkkemper, S., eds.: Conceptual Modelling in Information Systems Engineering. Springer, Berlin, Heidelberg, New York (2007)
15. Bramsiepe, N., Sikora, E., K.Pohl: Ableitung von Systemfunktionen aus Zielen und Szenarien. Softwaretechnik-Trends **28** (2008)
16. Colomb, R.: Ontology and the Semantic Web. IOS Press, Amsterdam (2007)
17. Guarino, N.: Formal Ontology and Information Systems. In Guarino, N., ed.: Formal Ontology in Information Systems. Proceedings of the First International Conference, June 6-8, Trento, Italy, Amsterdam, Berlin, Oxford, Tokyo, Washington, IOS Press (1998) 3–19
18. Aho, A., Lam, M., Sethi, R., Ullman, J.: Compilers, Principles, Techniques, & Tools. Addison-Wesley, Reading, Massachusetts (2007)
19. Krishnamurthy, S., Anson, O., Sapir, L., Glezer, C., Rois, M., Shub, H., Schlöder, K.: Automation of Facility Management Processes using Machine-to-Machine Technologies. In: The Internet of Things. Volume 4952 of LNCS. Springer, Berlin, Heidelberg, New York (2008) 68 – 86
20. de Kleer, J.: An assumption based truth maintenance system. Artificial Intelligence (1986)

Improving Result Adaptation through 2-step Retrieval

Meike Reichle and Kerstin Bach

Intelligent Information Systems Lab
University of Hildesheim
Marienburger Platz 22, 31141 Hildesheim, Germany
{reichle|bach}@iis.uni-hildesheim.de,

Abstract. In this paper we present the retrieval and adaptation mechanisms used in our information system on travel medicine, docQuery. The retrieval method's main feature is an overall improved accuracy of retrieval results' similarities through a more diverse distribution of similarities over the retrieved result sets. Its underlying idea is the execution of several consecutive retrievals on one case base, where attributes from the cases resulting from the first query are used to refine a subsequent query in order to yield better results than the first retrieval. The refined result sets narrow down the search space for cases to be used in result adaptation, which improves adaptation quality. The mechanisms are implemented in the docQuery information system on travel medicine.

1 Introduction

Intelligent information systems provide a technology for covering even complex topics in a comprehensive but flexible way. Realising such systems requires high quality data sources, knowledge models, and maintenance techniques. To achieve this, knowledge has to be acquired, analysed, stored, and retrieved, which challenges a knowledge-based system and is crucial for its continuous existence over a longer period of time. Case-Based Reasoning (CBR) is a methodology that has proven most effective for knowledge storage, retrieval and adaptation in all kinds of intelligent information systems [1].

In this paper we present the 2-step retrieval mechanism, a retrieval mechanism for CBR systems that works in an iterative way, executing two consecutive retrieval steps on the same case base, using information gained from the results of the first retrieval step in order to refine the second one. This consecutive retrieval strategy leads to an overall improved accuracy of retrieval results' similarities through a more diverse distribution of similarities over the retrieved result set.

Our first application of the 2-step retrieval mechanism is the docQuery project [2], an intelligent information system on travel medicine that is being developed in a joint project by the Intelligent Information Systems Lab and mediScon worldwide. docQuery is built using the SEASALT architecture [3], which is a first instantiation of the CoMES approach [4].

The paper is structured as follows: Section 2 presents the docQuery project, the application domain travel medicine and its particular challenges with regard to knowledge-based systems. Section 3 presents the actual 2-step retrieval algorithm, with an illustrated example. Section 4 finally presents an evaluation of the 2-step retrieval algorithm based on its application in the docQuery project, describing the evaluation's setup in subsection 4.1 and its results in subsection 4.2 followed by a description of related approaches in section 5. The paper closes with a conclusion and an outlook on future work in section 6.

2 Travel Medicine as an Application Domain: the docQuery System

Travel medicine is an interdisciplinary speciality concerned with the prevention, management and research of health problems associated with travel and covers all medical aspects a traveller has to take care of before, during and after a journey. For that reason it covers many medical areas and combines them. Furthermore, information about the destination, the activities planned and additional conditions have to be considered when giving medical advice to a traveller. Travel medicine starts when a person moves from one place to another by any kind of transportation and stops after returning home healthy. In case a traveller gets sick after a journey a travel medicine consultation might also be required.

The research project presented in this paper is supported by mediScon worldwide, a Germany based company with a team of physicians specialized on travel medicine and TEMOS¹, a tele-medical project of the Institute of Aerospace Medicine at the German Aerospace Center DLR². Together we will develop *docQuery*, an intelligent information system on travel medicine which provides relevant information for each traveller for their individual journey. First of all we will focus on prevention work, followed by information provision during a journey and information for diseased returnees.

Since common sources on the World Wide Web are not authorized by physicians and/or experts, we aim at providing reliable information for everybody. In preparation for a healthy journey it is important to get a high quality and reliable answer on travel medicine issues which both laymen and experts should be able to use. Based on the SEASALT architecture [3], we propose building a web community which gives information to travellers and physicians (non-experts in the field of travel medicine) by experts on travel medicine. docQuery will provide an opportunity to share information and ensure a high information quality because it is maintained by experts. Furthermore it will rise to the challenge of advancing the community alongside their users. Travellers and experts can visit the website to get the detailed information they need for their journey. A traveller will give docQuery the key data on their journey (like travel period,

¹ TEMOS means TElemedicine for a MObile Society, see <http://www.temos-network.org>

² <http://www.dlr.de/me/>

destination, age(s) of traveller(s), activities, etc.) and docQuery will prepare an information leaflet the traveller can take to his or her general practitioner to discuss the planned journey. The leaflet will contain all the information needed to be prepared and provide detailed information if it is required. In the event that docQuery cannot answer the traveller's question, the request will be sent to experts who will answer it. The computation of the answer follows the steps a physician would take during a consultation. Since travel medicine touches on different topics such as geographic information, diseases, medicaments, activities etc. We developed a modularised knowledge base, with a case base for each respective topic. These case bases are queried and their results are then combined, observing the constraints given by the user and domain itself (e. g. medical preconditions or medicines that cannot be taken in combination).

Modularising case bases into subdomains instead of simply partitioning them into smaller ones with the same case format, has several advantages. Firstly the individual case bases are easier to maintain with regard to the correctness of their contents, since they represent a more simple knowledge domain. Also, breaking up the rather complex domain of travel medical advisories into more simple subdomains that are then recombined as needed, gives the whole information system more flexibility. Providing the appropriate combination rules exist, the contents of the individual case bases can also be combined into cases that have not yet been presented to the system, as long as they adhere to the respective combination rules.

Further, not all knowledge domains that are included in an information system on travel medicine require the same type of maintenance and are subject to the same amount of change over time. While it is for instance no problem to keep a rather simple domain such as countries and regions as minimal and consistent as possible, the domain of travel related diseases is better served by including as many cases as possible, even if some of them are very similar. By splitting the knowledge domain into these smaller subdomains, each of them can be maintained in a way that is best suited for the respective subdomain.

Since the topic of this paper is the retrieval on one individual case base, we only gave a short overview of the docQuery System and its modularised case bases in this section. More on modularised case bases, their maintenance, and the combination of their results can be found in [5].

3 2-Step Retrieval

When dealing with a topic like travel medicine we cannot assume that all users ask complete and/or correct questions and like Weibelzahl [6] we enrich the user's query enhancing it with additional information from the case base.

Our initial retrieval is based on the geographic position of a country and because of the fact that the earth is divided in a manageable amount of countries, which are completely covered in our case base's similarity measure in the form of a geographic taxonomy, we can rely that every requested country can be retrieved. However, we cannot be sure that we will have (complete) information on

that country. Also, due to the nature of our domain, travel medicine, geographic proximity is not sufficient to find feasible adaptation candidates to complete the retrieved country's information – also occurring diseases have to be noticed. In our retrieval mechanism we thus start by requesting the destination country, this step is followed by an enhanced query including additional information about the initial country's diseases. The second retrieval's results with the highest similarity will be the adaptation candidates we take into account. In the current approach we randomly pick one of the cases with the highest similarity as adaptation source. In the future we will add maintenance information to the cases to be able to compute the most updated or most recently maintained adaptation candidate.

The approach presented here concentrates on interdependent attributes that are not completely given for every single case. We will show how we can narrow the result set by retrieving reliable information snippets in order to adapt them to create a (complete) response.

Assuming that we have a traveller planning a journey, the retrieval will start based on the destination region. We know that our model contains all countries of the world, so the retrieval algorithm will be able to find the appropriate destination. But due to many changes of disease outbreaks we have to provide up to date country and regional information. Therefore we do not only retrieve the country we have searched for, we also include in our result set countries with a similar *structure* considering travel medical aspects. To realize this we also use information about vaccinations that can be divided in three categories:

1. **Obligatory Vaccinations:** Those vaccinations are required in order to be allowed enter a country.
2. **Standard Vaccinations:** Those vaccinations are required if one is travelling to a certain country - although they are not a regulation.
3. **Risk Vaccinations:** Those vaccinations are required for people with an enfeebled immune system such as pregnant women, children, elderly people, or those who suffer from different kinds of (chronic) diseases and require a higher protection provided by a vaccination.

Additionally our system will give information about diseases that can be contracted during a journey. According to [7] those can be divided in the following categories of diseases: vectors (In medicine a vector is a carrier of infections, diseases, etc. because it carries for example the parasitic agent i.e. in malaria a mosquito serves as the vector), person-to-person contact, ingestion of food and water, bites and stings, and water/environmental contact.

Currently we do have information about vaccination advices, but we do not have complete information about diseases contracted during a journey, because they rely until a certain point on up-to-date information. Nevertheless we will provide this kind of information to the users of docQuery and since we do have similarity models for each type of disease we will adapt the information from similar countries.

In order to ensure that our system adapts correct data we will use the 2-Step-Retrieval to reduce the amount of cases we can adapt from. In the first

step we will only do the retrieval based on our geographic taxonomy, then we narrow the set of retrieved cases by adding vaccination information to a second query. The taxonomy includes 228 countries and islands arranged by continents, subcontinents (e.g. Western Europe), regions (e.g. Iberian Peninsula) followed by the country. A generalization step leads to a value of 0.5 and specialization to a value of 0.8.

When performing a standard 1-step retrieval on Laos, a whole of 10 countries in it's geographic proximity have a similarity of 40%. When performing a 2-step retrieval, the distribution of similarities is much more diverse, as illustrated in figure 1.

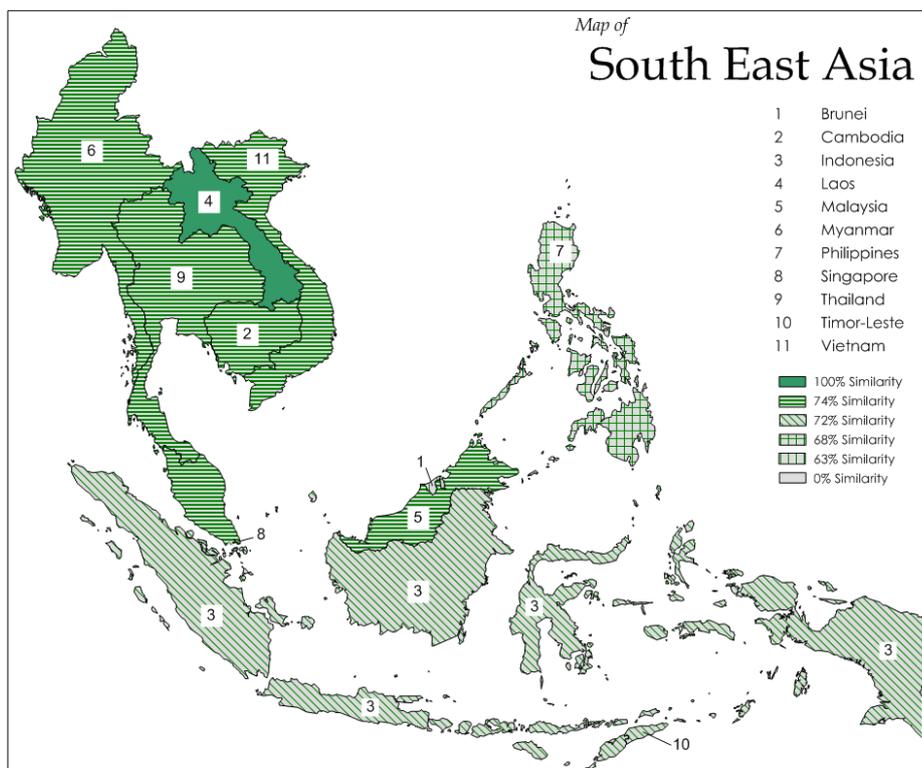


Fig. 1. 2-Step-Retrieval for the example of Laos

Laos does not yet have complete information on the diseases contracted there, so these attributes have to be filled using adaptation from another, similar case. To complete the disease information we now have to choose one country to adapt from. Using 1-step retrieval there are 10 adaptation candidates to chose from,

some of which, such as e. g. the Philippines are in fact quite different from Laos with regard to travel medicine.

To ensure that the cases we adapt from are more similar to the requested country, we now also consider vaccination information in our request, using it in a second retrieval step as illustrated in figure 2. Since we need information on countries with a similar *disease structure* in order to be able to find a country profile with an appropriate amount of information, even if the destination originally given by the user does not offer those, we use 2-step retrieval. An example query (again using the country, this time plus the vaccination information of the originally retrieved country) for the second retrieval step would be: "*Laos, Yellow fever, Diphtheria, Hepatitis A, Measles, Tetanus, Cholera, Hepatitis B, Japanese Encephalitis, Rabies, Typhoid fever*".

After the second retrieval step, the countries situated around Laos now have more differentiated similarities and offer a higher amount of information considering their *disease structure*. As can be seen in figure 1, this time only the countries near Laos as well as Malaysia are returned with the same similarity to Laos, reducing the number of adaptation candidates to 5. If we are taking one of those countries into account the likelihood of retrieving a valid result set will be highly increased.

4 Evaluation: 2-Step-Retrieval

Following the example given in section 3 we will now present the evaluation of our approach in the travel medicine domain. Therefore we will present how the 2-Step-Retrieval affects the retrieved result sets and we illustrate its advantage in comparison to a straightforward 1-step retrieval approach. In the architecture of docQuery, the case bases *Destination*, *Associated Information*, and *Activity* are adequate to do 2-step retrieval, since our other case bases contain health critical information that require a more strict retrieval.

4.1 Experimental Setup

The *Destination* case base covers country characteristics that are used to prepare an information leaflet for a traveller. It contains vaccination requirements and vaccination-preventable infectious diseases, pre-travel information on different kinds of diseases that might occur in a certain country or region, as well as hygiene and prevention advise.

The experimental data contain a case base covering all countries in the world and the vaccination information abroad we have to consider preparing information leaflets for travellers. To carry out the experiment we took a controlled sample of 18 countries of East and South East Asia, representative with respect to country borders, coasts, islands and climatic conditions and manually filled in the data on transmittable diseases, so that we have 18 cases with complete information. The sample comprises all countries of East and South East Asia that

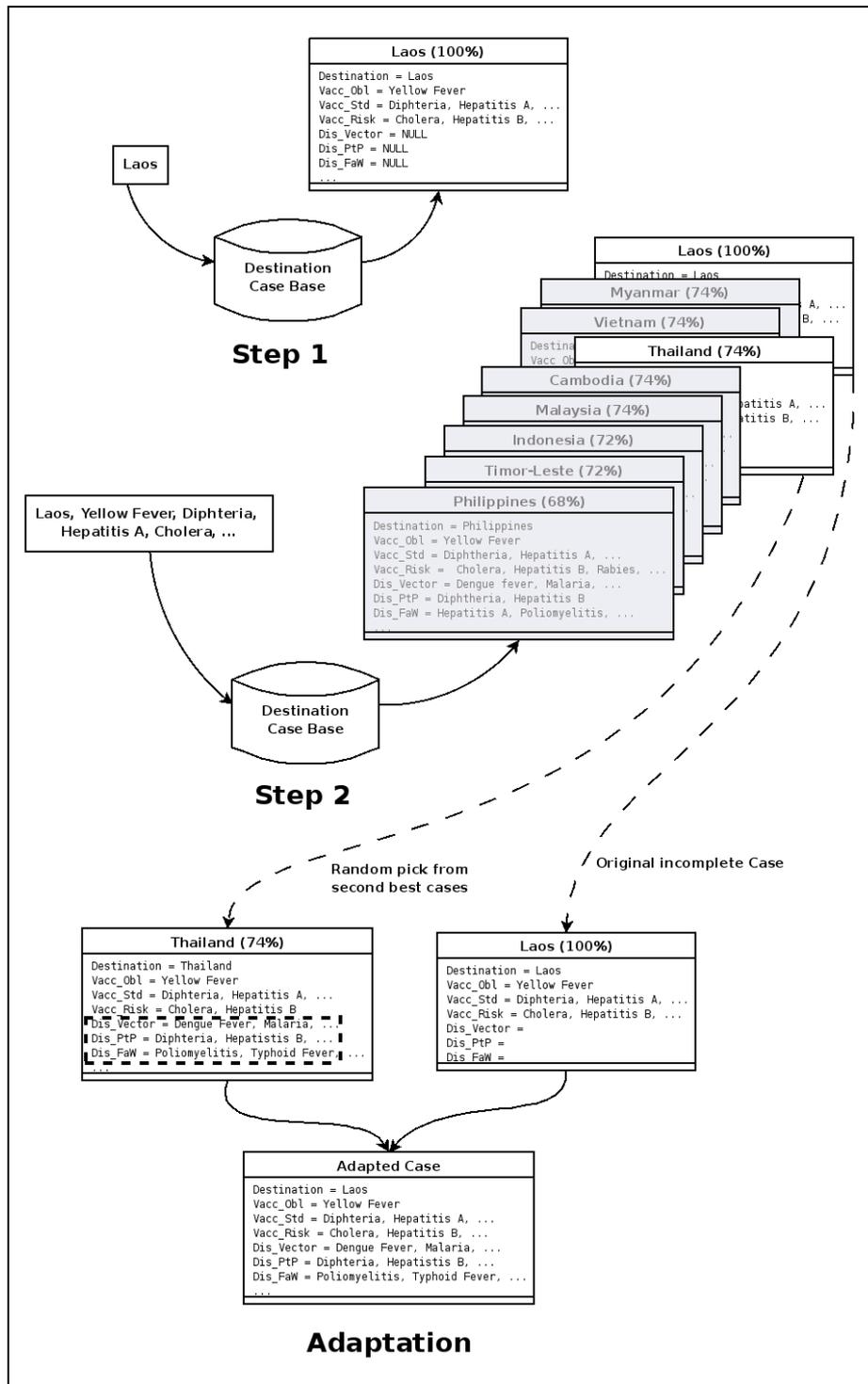


Fig. 2. 2-step retrieval and adaptation using Laos as the query and Thailand as the randomly picked adaptation candidate.

ensures that we are able to find neighbouring countries, non-neighbouring countries, and countries associated to different nodes (e.g. China which is situated in Eastern Asia and Thailand which is in South East Asia).

We carried out our evaluation as a leave-one-out experiment. For each country in the case base we did the following steps:

1. Remove diseases information from country.
2. Do a one-step retrieval using the country name.
3. Do a two-step retrieval using the country name and vaccination information as additional information.
4. Do an adaptation with each respective result set.
5. Compare the set of diseases obtained from the two respective adaptation candidates to the original set of diseases.

For the experiment we use the following weights:

$$sim = 6 \times [Region] + 4 \times [Vacc_Risk] + 3 \times [Vacc_Std] + 2 \times [Vacc_Obl] \quad (1)$$

In our similarity measure "*Region*" is weighted times 6 because it is the most diverse and reliable fact, and because we expect that travellers know their destination, but not the diseases. "*Risk people vaccination*" information are weighted times 4 because vaccination advices depend on each traveller's profile and in particular his or her disease history in combination with chronic illness(es). Also, this attribute differentiates countries from each other. "*Standard vaccination*" advices are weighted times 3 because they describe the disease structure from the medical point of view and allow a general classification. "*Obligatory vaccination*" is weighted times 2 because obligatory vaccination requirements depend on the geographic region as well as on official orders.

For each country we did 5 runs³ and compared the result of the 1-step-retrieval and the 2-step-retrieval to the expected result. First we compared the number of adaptation candidates and then the resulting adaptation quality, checking if all expected disease were found, if one or more diseases were missed (*false negatives*), or if incorrect extra diseases were added to the case (*false positives*).

4.2 Results of the Experiment

Figure 3 shows how the number of adaptation dropped significantly in 90% of our test cases. These numbers illustrate that result sets are indeed more diverse when enriching the queries with extra information. Moreover not only the number of adaptation candidates change, also the adaptation candidates differ between 1-step and 2-step retrieval. For example the retrieved countries for Japan: In the 1-step-retrieval North Korea, Mongolia, Taiwan, China, as well as Macau are returned, but the adaptation candidates for the 2-step retrieval do not include Mongolia, but Hong Kong, because of the fact that the disease structure of Hong

³ We chose to perform several runs, since the randomising element in the final choice of the adaptation candidate can yield different results for the same query.

Kong is much more similar to Japan than Mongolia. In all test cases at least one country has been dropped out of the adaptation candidates, but on average 3.3 countries were not taken into account in the 2-step retrieval. In 39% of the cases an adaptation candidate has been add.

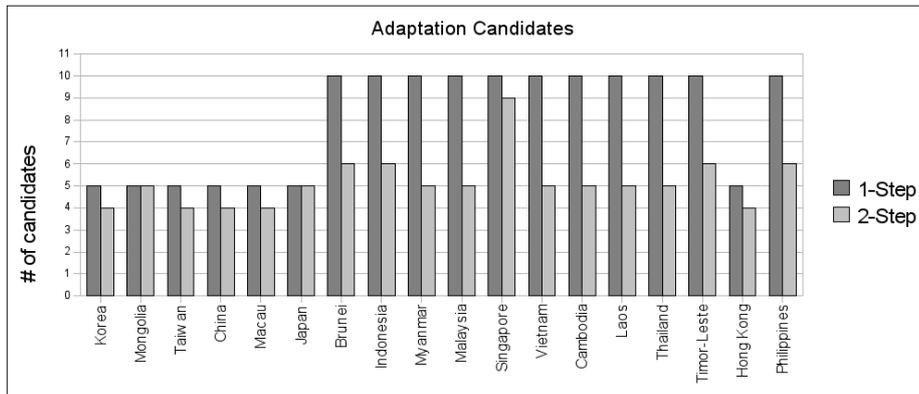


Fig. 3. Comparison of the number of adaptation candidates in 1- and 2-step retrieval

In the next step we investigated whether the adaptation candidates remaining after the 2-step retrieval were in fact the better ones, that is, if their adaptation results were better than the adaptation results from the candidates resulting from one-step retrieval. The results of the adaptations can be seen in Figure 4.

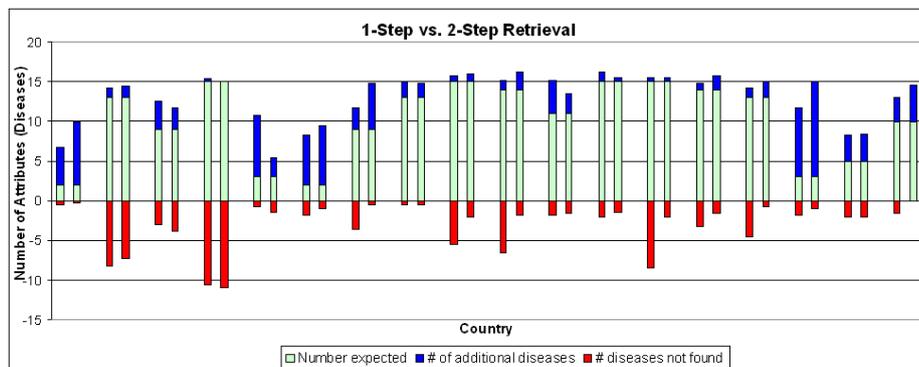


Fig. 4. Result Comparison of the 1-Step vs. the 2-step Retrieval

Each column pair represents the aggregated results of one country – the first column shows the adaptation results of the 1-step-retrieval and the second of the

2-step-retrieval. The y-axis shows the number of correct diseases (*true positives*, positive scale, light-coloured) and extra (*false positives*, positive scale, dark-coloured) diseases found as well as missed diseases (*false negatives*, negative scale, dark-coloured). The result sets used for the evaluation result from requesting the following countries: Korea, Mongolia, Taiwan, China, Macau, Japan, Brunei, Indonesia, Myanmar, Malaysia, Singapore, Vietnam, Cambodia, Laos, Thailand, Timor-Leste, Hong Kong, and the Philippines.

In total we did 90 single-case requests and after the 1-step-retrieval 62% of the adapted cases contained all of the expected diseases. Applying the 2-step retrieval to the same cases 76% of the adapted cases contained all expected diseases. Although both retrieval variants also return false positives in most of the tests, the solutions of the 2-step retrieval are generally more reliable, especially according to false negatives. The 2-step retrieval performed significantly better than the 1-step retrieval with regard to false negatives as can be seen in Figure 5.

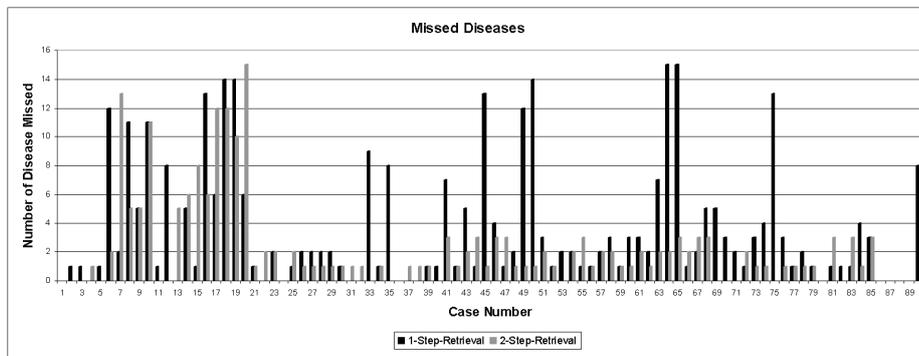


Fig. 5. Unaggregated comparison of missed diseases (false negatives).

In summary the experiment shows that the 2-step retrieval provides more robust results and especially in the field of travel medicine a query can be significantly enhanced by adding more information on the destination, because, for example, disease and vaccination advice can only be provided by an expert, not by the traveller. Furthermore the information stored in the case bases can be used to create more refined queries.

5 Related Work

The idea of using and combining information from different cases has also been discussed in [8], in which Redmond describes how snippets of different cases are combined to receive a solution for a given problem. In comparison to our approach, Redmond uses similar case representations from which he extracts

parts of cases in order to combine them, but in our approach we take the whole retrieved case as a snippet of our solution. Nevertheless those two approaches have in common that each snippet has to match the other snippets and limits solutions that go along with it.

A similar approach has been presented in [9, 10] in which the incremental CBR (I-CBR) mechanism for diagnosis has been introduced. The I-CBR separates information in between "free" and "expensive" features and starts the first retrieval steps based on the free features before the user is asked to give information about expensive features to narrow the set of information. In comparison with this approach we have a different point of view. Our system already holds the user's information and we do not necessarily narrow the result set, but we use the 2-step retrieval to tighten the set of candidates we derive information from to adapt single information. Another approach on how I-CBR can influence the result sets has been presented in [11], but in comparison to our approach Jurisica et. al. did not receive additional information from exiting case, they used query series and user interaction instead.

In [6, 12] Weibelzahl uses a travel domain consisting of two case bases with different knowledge models. The first case base, called customer case base, holds information on the customers' needs and desires which are mapped to attributes describing products provided in the second case base. In the first step the query containing the user's expectation on their vacation is analysed to set relevant attributes creating a request which can be sent to the product case base regarding the users' expectations. The second request contains especially those product attributes the user would not request on their own, but help to find an appropriate solution in the product case base.

6 Conclusion and Outlook

In this paper we have presented a retrieval mechanism that enhances a query with information in order to receive a more diverse result set. Using 2-step retrieval provides us with robust results to dispatch the subsequent retrieval and result combination. The 2-step retrieval algorithm presented in this paper exemplifies how the retrieval strategy can be implemented in a CBR system. Further on we use this approach to combine and adapt parts of cases and attributes of different case bases, because we expect that our information obtained of the travel medicine community will be incomplete. Also we suppose that taking more attributes into account might help the algorithm to receive even more diverse result sets.

As a next step we will enable our system to combine the retrieval results of cases retrieved of modularised heterogeneous case bases in order to create a whole individual information leaflet for travellers containing information on activities, diseases, medication, etc. Hence, we will implement a multi-agent system centred around a coordination agent (or broker agent) combining retrieval results and ensuring complete information regarding given constraints.

Another aspect of our future work is generalising the 2-step retrieval algorithm and evaluating whether the algorithm can be applied to other case bases and domains as well, or if this only works for our specific domain. Also, we have to figure out if the algorithm of using retrieval results for refining queries can be applied to other case bases in the travel medicine application domain as well as in other application domains.

Integrating the 2-step retrieval algorithm in SEASALT puts forward our idea of using knowledge lines for building CoMES upon existing knowledge sources.

References

1. Bergmann, R., Althoff, K.D., Breen, S., Göker, M.H., Manago, M., Traphöner, R., Wess, S.: Selected Applications of the Structural Case-Based Reasoning Approach. In: *Developing Industrial Case-Based Reasoning Applications: The INRECA-Methodology*. Volume 1612 of *Lecture Notes in Computer Science*. Springer (2003) 35–70
2. Bach, K.: docquery - a medical information system for travellers. Internal project report (September 2007)
3. Bach, K., Reichle, M., Althoff, K.D.: A domain independent system architecture for sharing experience. In: *Proceedings of LWA 2007, Workshop Wissens- und Erfahrungsmanagement*. (September 2007) 296–303
4. Althoff, K.D., Bach, K., Deutsch, J.O., Hanft, A., Mänz, J., Müller, T., Newo, R., Reichle, M., Schaaf, M., Weis, K.H.: Collaborative multi-expert-systems – realizing knowledge-product-lines with case factories and distributed learning systems. In Baumeister, J., Seipel, D., eds.: *Workshop Proceedings on the 3rd Workshop on Knowledge Engineering and Software Engineering (KESE 2007)*, Osnabrück (September 2007)
5. Althoff, K.D., Reichle, M., Bach, K., Hanft, A., Newo, R.: Agent based maintenance for modularised case bases in collaborative multi-expert systems. In: *Proceedings of AI2007, 12th UK Workshop on Case-Based Reasoning*. (December 2007) 7–18
6. Weibelzahl, S.: Conception, implementation, and evaluation of a case based learning system for sales support in the internet. Master’s thesis, Universität Trier (1999)
7. The International Society of Travel Medicine: The body of knowledge for the practice of travel medicine (2003)
8. Redmond, M.: Distributed cases for case-based reasoning: Facilitating use of multiple cases. In: *AAAI*. (1990) 304–309
9. Cunningham, P., Bonzano, A., Smyth, B.: An incremental case retrieval mechanism for diagnosis (1995)
10. Cunningham, P., Smyth, B., Bonzano, A.: An incremental retrieval mechanism for casebased electronic fault diagnosis (1998)
11. Jurisica, I., Glasgow, J., Mylopoulos, J.: Incremental iterative retrieval and browsing for efficient conversational cbr systems. *Applied Intelligence* **12**(3) (2000) 251–268
12. Weibelzahl, S., Weber, G.: Benutzermodellierung von Kundenwünschen durch Fall-basiertes Schliessen. In Jörding, T., ed.: *Adaptivität und Benutzermodellierung in interaktiven Softwaresystemen, ABIS-99*, Magdeburg (1999) 295–300

Translating Cooperative Strategies for Robot Behavior^{*}

Florian Ruh and Frieder Stolzenburg

Hochschule Harz, Automation and Computer Sciences Department, D-38855 Wernigerode
{fruh, fstolzenburg}@hs-harz.de

Abstract. This paper presents a method for engineering and programming multi-robot systems, based on a combination of statecharts and hybrid automata, which are well-known in the fields of software engineering and artificial intelligence. This formal specification method allows graphical presentation of the whole multiagent system behavior. In addition, these specifications can be directly executed on mobile robots. We describe the transformation process from the specification to executable code, after introducing the necessary definitions. A translator that automatically converts hybrid hierarchical statecharts into simple flat hybrid automata (i.e. without hierarchies) has been implemented. The respective tool allows the text-based input of hybrid hierarchical automata specifications of multiagent system with synchronization. The translation into flat automata is performed by means of different plug-ins, leading e.g. to executable code for Sony Aibo robot dogs. The plug-in just mentioned has been successfully applied in the RoboCup four-legged league.

Key words: agent-oriented software engineering; multiagent systems; RoboCup; tools for intelligent systems.

1 Introduction

Robotic soccer provides many research challenges and one of them is behavior control including the subjects of team play, cooperation and flexible, quick reaction. A soccer team can be designed as a homogeneous multiagent system. Since the behavior of multiagent systems and agents alone can be understood as driven by external events and internal states, an efficient way to model such systems are state transition diagrams, which are well-established in software engineering. They are graphical representations of finite state machines with hierarchically structured states and transitions which lead from one state to another depending on the input or events. Outputs or actions can be done during transitions or in states. State transition diagrams have been applied successfully for multiagent systems in general and in the RoboCup, a simulation of (human) soccer with real or simulated robots (see e.g. [2, 5, 19]).

However, state transition diagrams do not properly cover all aspects of multiagent systems. Therefore, hybrid hierarchical automata (HHA) with timed synchronization have been developed to take continuous processes in the environment into account [7]. Moreover, they can consider time as an additional factor for synchronization processes. This formalism can help to model situations when two or more agents have to deal with one resource. In the domain of soccer e.g., the agents have to consent that exactly one player goes to the ball.

^{*} This paper emerged from the master thesis of the first author [16].

Therefore, as a running example, we consider a scenario influenced by the UEFA champions league competition 2006/2007: the *Makaay move* (see Fig. 1). Let there be one player of type *A* and two players of type *B* in the offensive team. Player *A* performs the kick-off, while the players of type *B* are waiting in different sectors on the pitch, which is divided into sectors (cf. [6]). Player *A* chooses a direction for passing (right or left midfield, sectors 3 or 5), then kicks off and passes to one player of type *B* in the destination sector. Player *A* runs to sector 1 (middle offense) where *B* has passed the ball to. Finally, player *A* tries to shoot to the goal directly. If it fails, *A* tries it again. Meanwhile, *B* goes to the ball if it is nearer to it. *B* then passes to *A* in sector 1 again.

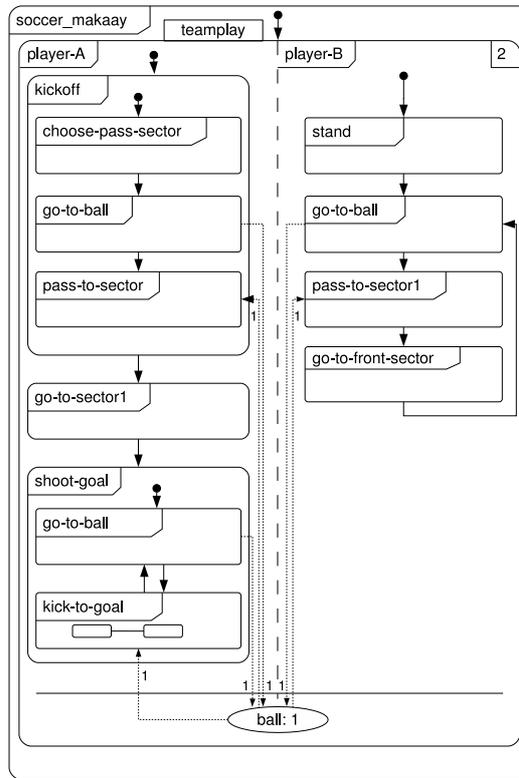


Fig. 1. Statechart for Makaay move.

In the sequel, Sect. 2 covers the formal specification of hybrid statecharts. Corresponding description and target languages are defined and compared in Sect. 3. With these foundations, we can create a concept for the translation process. Sect. 4 then deals with the design of the application and shows an example of use. Finally, we discuss related works in Sect. 5 and conclude with Sect. 6.

2 Hybrid Statecharts

2.1 States and Transitions

In a realistic physical environment, it is inevitable to consider continuous actions in addition to discrete changes. Hybrid automata extend regular state transition diagrams with methods that deal with those continuous actions. To understand the characteristics, we will introduce several definitions for hierarchical hybrid automata with timed synchronization now [7, 8] – called HHA.

Definition 1 (basic components). *The basic components of a state machine are the following disjoint sets:*

S: a finite set of states, partitioned into three disjoint sets: S_{simple} , S_{comp} , and S_{conc} — called simple, composite and concurrent states, containing one designated start state $s_0 \in S_{comp} \cup S_{conc}$;

X: a finite set of variables, partitioned into two disjoint sets: X_{real} and X_{int} — the continuous/real-numbered and the integral/integer variables, respectively; for each $x \in X$ we introduce the variables x' for the conclusions of a discrete change;

T: a finite set of transitions with $T \subseteq S \times S$.

Definition 2 (state hierarchy). Each state s is associated with zero, one or more initial states $\alpha(s)$: a simple state has zero, a composite state exactly one, and a concurrent state more than one initial state. In the latter case, the initial states are called regions. Moreover, each state $s \in S \setminus \{s_0\}$ is associated to exactly one superior state $\beta(s)$. Therefore, it must hold $\beta(s) \in S_{conc} \cup S_{comp}$. A concurrent state must not directly contain other concurrent ones. Furthermore, it is assumed that all transitions $s_1 T s_2 \in T$ keep to the hierarchy, i. e. $\beta(s_1) = \beta(s_2)$. Furthermore, we write $\alpha^n(s)$ or $\beta^n(s)$ for the n -fold application of α or β to s , in particular, $\alpha^0(s) = \beta^0(s) = s$. Variables $x \in X$ may be declared locally in a certain state $\gamma(x) \in S$. A variable $x \in X$ is valid in all states $s \in S$ with $\beta^n(s) = \gamma(x)$ for some $n \geq 0$, unless another variable with the same name overwrites it locally.

As said earlier, Fig. 1 depicts the statechart for our running soccer example. Here, states are named after their affiliation to the players or the actions which are being done at that moment. The states *soccer_makay* (which is the start state s_0 here), *kickoff*, *go-to-ball*, *player-A* and *player-B* are composite states; *teampay* is a concurrent state while all others are simple states. The oval symbol *ball* is a synchronization point and will be discussed in Sect. 2.2.

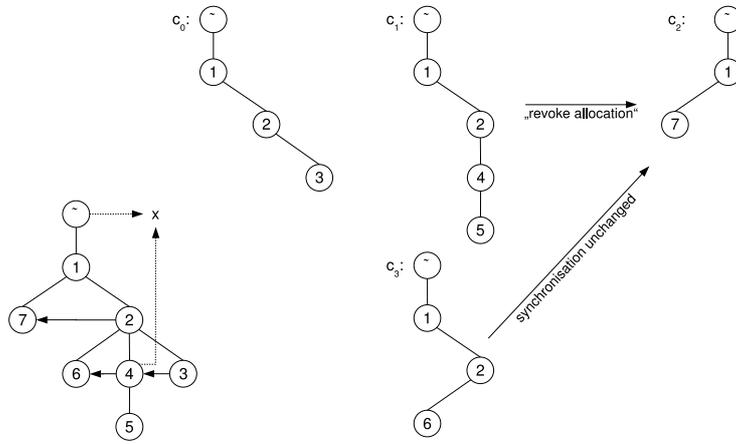
Definition 3 (jump and state conditions). For each transition, there exists a jump condition. This is a predicate with free variables from the valid variables of $X \cup X'$. Additionally, each state $s \in S$ contains a state condition which describes continuous changes in s . It is a predicate with free variables from $X \cup \{t\}$.

Events are well-known in UML statecharts [12] and hybrid automata [8]. They can easily be expressed by (binary) integer variables in our formalism. Therefore, we do not introduce them explicitly in our definitions. But in contrast to simple hybrid automata, we introduce hierarchies. Fig. 2(a) shows an example state tree, which is induced by the β -function. Here, R is the root of the tree, and e.g. state 1 can be reached from 5, i.e. $1 = \beta^3(5)$. Note that the value of β^n is always uniquely determined due to the tree-like (and not graph-like) structure of the state hierarchy. Furthermore, let $\alpha^3(R) = 3$. A configuration (defined next) is the subset of the active states in the state tree.

Definition 4 (configuration and completion). A configuration c is a rooted tree of states with the root node as the topmost initial state of the overall state machine. Whenever a state s is an immediate predecessor of s' in c , it must hold $\beta(s') = s$. A configuration must be completed by applying the following procedure recursively as long as possible to leaf nodes: if there is a leaf node in c labeled with a state s , then introduce all $\alpha(s)$ as immediate successors of s .

The semantics of our automata can now be defined by alternating sequences of discrete and continuous steps. Following the synchrony hypothesis, we assume that discrete state changes (via transitions whose annotated jump condition holds in the current situation) happen in zero time, while continuous steps (within one state) may last some time. Due to the lack of space, for details on the semantics of HHA, the reader is referred to [15].

Fig. 2(b) demonstrates the relationship between state trees and configurations. It depicts several configurations that are created from the state tree in Fig. 2(a). A configuration itself can be connected to another one. The original transition t , which was used for the completion of s_2 in c_2 , is used in a discrete step while its origin is changed from s_1 to c_1 and its target from s_2 to c_2 .



(a) State tree with synchronization point x . (b) Configurations with synchronization problem.

Fig. 2. State tree and configurations of an automaton.

2.2 Synchronization

Synchronization is significant for modeling multiagent systems. Usually, a system deals with limited resources. The interaction with them can take part in several states. Especially when reacting to events from the environment, the reaction process takes some time $\tau > 0$. For this, a *synchronization* takes care of the common resources defined at a *synchronization point*. While synchronization is associated with transitions, implemented via labels in original hybrid automata [8], synchronization is associated with states in HHA, i.e. actions which last some certain time. In contrast to this, the synchrony hypothesis states (for discrete steps), that a system is infinitely fast and therefore can react immediately within zero seconds, i.e., a transition takes zero time.

Definition 5. A *synchronization point* is identified by a variable $x \in X_{\text{sync}} \subseteq X$ with a maximum capacity $C(x) > 0$. Each state connected to the synchronization point is

classified by one of the following relations, $R_+ \subseteq S \times X_{sync}$ or $R_- \subseteq X_{sync} \times S$. If a state increases the capacity, it will be classified by R_+ and otherwise by R_- , if it decreases it (or resets the resource). In general, each connection in $R_+ \cup R_-$ is annotated with a number m with $0 < m \leq C(x)$ which identifies the volume to be increased or decreased from the synchronization point, respectively.

Synchronization may take some time, since they are connected to (continuous) states and not to discrete transitions. Thus, the synchronization process can theoretically be interfered by other actions or concurrent states which also try to share the same synchronization point. To avoid side effects that may lead to inconsistency or even system failure, the process is separated into allocation and (future) occupation of resources. For this, the allocation variables x_+ and x_- register the request for occupation or release for each synchronization point x . Therefore, x_+ and x_- must be added to X .

In this case (synchronization point x and connected state s), $s_1 T s_2$ is called *incoming transition* for s iff $\alpha^n(s_2) = s$ for some $n \geq 0$, *initializing transition* iff it is an incoming one with $\alpha^n(s) = \gamma(x)$, *outgoing transition* iff $s_1 = \beta^n(s)$ for some $n \geq 0$ where s_1 occurs in the current configuration and x is valid in s , *successful outgoing transition* iff it is an outgoing transition with $s_1 = s$ and *failed outgoing transition* iff it is not a successful outgoing transition. Note that outgoing transitions cannot be characterized statically but only dynamically by investigating the configuration trees. This is an important issue for the revoking of the allocation (see Sect. 4.2). At a synchronization point x , additional constraints must be defined which affect the transitions that are incident with all states s connected to x . Due to the lack of space, for details on the synchronization concept, the reader is referred to [7].

The synchronization point *ball* in Fig. 1 has a capacity of 1. Thus, it can be interpreted as a Boolean value as there is only one ball in a soccer match. Both states *go-to-ball* occupy the synchronization point *ball*. Hence, they belong to the relation R_+ . The states *kick-to-goal* and *pass-to-sector* release it and therefore belong to R_- .

The example in Fig. 2(a) also makes use of a synchronization point. As seen in the tree, the state R introduces the synchronization point x while 4 is somehow using it here. The definition is marked with the dashed arrow pointing at x . However, for some multiagent systems, the synchronization must be converted into ordinary variables if a target platform does not provide synchronization interfaces.

3 Specification Languages

After having defined basic concepts, let us now consider concrete languages for programming multiagent systems with HHA. Therefore, we will discuss two languages briefly in the sequel: HAL and XABSL.

The project goals of HAL [3] were the definition of an ASCII-formatted specification language for hybrid automata with timed-synchronization and, furthermore, its transformation into an input format for model checkers such as *HyTech* [8]. HAL is at the same time the name of the project and the name for the specification language (*Hybrid Automaton Language*). This corresponds to the definitions introduced in the previous section. A HAL specification is usually written into an ASCII formatted file.

According to the syntax, it consists of a global frame which must be a composite automaton. It may include several other automata following the rules of hybrid automata with timed synchronization. Even though the terms of inheritance, polymorphism are not defined in HAL syntax, modularization is actually known. The namespace of two parallel automata cannot collide while subsequent automata can access variables of their superiors. An example is shown in the listing (Fig. 3).

```

composite makaay {
  start( teamplay );
  concurrent teamplay {
    syncpoint( ball , 1 );
    region player_A {
      cardinality := 1;
      start( kickoff );
      composite kickoff {
        start( choose_pass_sector );
        var pass_sector := 0;
        var random_05_3_5 = 0;
        simple choose_pass_sector {
          flow := pass_sector ~ = random_05_3_5;
          invariant := pass_sector != 3 & pass_sector != 5;
          trans := ( go_to_ball , pass_sector == 3 |
            pass_sector == 5 );
        } % choose_pass_sector
        simple go_to_ball {
          sync( ball , 1 );
          flow := go_to_ball_without_turning_maxspeed120;
          invariant := ball_seen_distance >= 70;
          trans := ( pass_to_sector , ball_seen_distance < 70
            );
        } % go_to_ball
        % (...)
        invariant := ball_sector == 4;
        trans := ( go_to_sector1 , ball_sector != 4 );
      } % kickoff
      % (..)
    } % player_A
    % (...)
  } % teamplay
} % makaay

```

Fig. 3. HAL specification.

Another successful approach of modeling agent behavior is *XABSL* (Extensible Agent Behavior Specification Language) [11]. It was developed and integrated into the code basis of the *GermanTeam*, several times world and German champion in the

RoboCup four-legged league, as a language for behavior engineering. The specifications can be transformed automatically into intermediate code which has to be interpreted on the target platform by the *XabslEngine*. The XABSL package also provides functionalities for visualization, debugging and documentation. The *option* division in XABSL specifications includes a global symbol file to get access to the environment. It consists of one initial and several other states with their own decision trees. The *action* division specifies all assignments that are executed there. A subsequent option call is also possible.

4 The Translator Tool

The HAL converter provides a window-based flattening mechanism for state machine specifications, a batch mode for quick processing, and re-usability. Additionally, there should be a graphical editor to easily create source code from hybrid statecharts. Already created files (or files that are created manually) are allowed to be used as an input for the application. Hence, a lexer and a parser provide the conformity with the HAL syntax. With this design, it is possible to create a hybrid automaton which can be used later as input for the flattening algorithm (see below). The translator from HAL to XABSL shall cover all features of synchronized hybrid state machines that can be transferred to XABSL.

4.1 Flattening Algorithm

For the translation process, there is no simple one-to-one structural mapping between HHA and XABSL. As XABSL and also standard verification tools often are not able to cope with hierarchies, it is required to flatten the automaton, i.e., all states except the initial one are transformed into simple ones. As the translator shall be feasible of creating processable output for those tools, this gives us another reason to flatten the hierarchical structure. Though this transformation may lead to state explosion, it could be avoided, nevertheless, if hierarchical configurations could be processed as directly as in some logic-based implementations [7, 15].

In the implementation, configurations are used to clarify which agent currently is in which state. The flattening algorithm processes an input state tree and converts it to a set of configurations. In particular, the output can be used to simplify the agent's behavior structure and to gain performance due to less complexity. For this, the algorithm is divided into four major parts.

1. **Copy regions**

Expand the regions in the tree according to their cardinality c (given in the upper right corner of a region). Modify each region to a composite state, copy it c -times and replace the original with the copies.

2. **Globalize variables and constants**

Each state may introduce variables and constants. Each local definition must be globalized as it will be used in the configuration flows and transitions later on. The global definitions must be uniquely named to avoid namespace collisions.

3. Convert synchronizations

If a state uses a synchronization point to interact with other states, these synchronizations must be resolved. Due to their complexity, a relatively extensive inspection is required which is explained in detail in Sect. 2.2 and 4.2. Although the resulting additions to transition guards reduce readability, the even more complex process of inter-state synchronization could be eliminated. A practical approach for the detection of the correct place to revoke an allocation is given below.

4. Create configurations

Each state tree possesses an initial configuration c_0 . This contains all the initial states that can be reached in the tree beginning at the root. According to the completion algorithm (Def. 4), the configurations are created recursively beginning at c_0 . Already existing configurations will be recognized and used if transitions lead to them. These newly created transitions form the discrete steps of the system.

The synchronization conversion in the third step is a rather complex process. At first, all synchronization points in the automaton are collected. After this, the automaton will be traversed, and the occupation, the release, as well as the allocation, and its revoke are added for each synchronization found in the state tree. The synchronization point x itself, its maximum capacity $C(x)$, and its allocation variables x_+ and x_- are converted into global variables. For each transition type, different expressions must be added to the guards and the discrete expressions. However, a flag x_f for each synchronization point x is introduced indicating its current status. If x is occupied then $x_f := -1$. If x is allocated but not occupied yet then $x_f := 1$. Otherwise, $x_f := 0$. For all not initializing incoming transitions, $x_f := 1$ will be added to their discrete expressions, $x_f := 0$ will be added for all initializing incoming transitions, $x_f := -1$ will be added for all successful outgoing transitions. For each not successful outgoing transition, it must be checked if x is already allocated but not occupied by this synchronization. Therefore, the transition must be duplicated. The comparison $x_f = 1$ is added to the guard of the first transition, $x_f \neq 1$ is added to the second one. The revocation of the allocation is added only to the discrete expression of the first one. Finally, all synchronization points can be erased as they are now properly converted into ordinary variables.

4.2 Allocation in Synchronizations

During the development of the theoretical model of hybrid automata with timed synchronization, a problem concerning not successfully outgoing transitions occurs. The correct situation has to be found, when the allocation shall be revoked, since it must actually be done only once per occupation. For this purpose, some definitions have to be introduced.

Let $\delta(s)$ return all variables used in the state s but not defined there. Furthermore, we introduce a mapping ζ which returns all state successors of s that use x and are part of the configuration c :

$$\zeta(s, x, c) = \{s_i \mid \beta^n(s_i) = s \wedge x \in \delta(s_i) \wedge s_i \in S(c), n > 0\}$$

Fig. 2(a) depicts a simple example for that synchronization problem. The dashed arrows indicate the definition and the usage of the synchronization point x . The state tree

shows – among others – a transition from state 2 to 7. Fig. 2(b) shows the appropriate configurations with c_0 being the initial one. In this case, the synchronization point x is defined in the state R while only 4 is using x . In fact, R must be a concurrent state as it defines a synchronization point. Though concurrent states usually have two or more regions, this example reduces complexity and actually uses only one.

Let us now have a closer look on what is happening in c_1 when the process has activated state 5. State 4 is also active as it is the immediate predecessor of state 5 in the tree. The transition from 2 to 7 is a not successful outgoing transition for 4 as $2 = \beta^n(4)$ with $n = 1 > 0$.

Now, to collect all states that may have allocated x before the transition t induces a discrete step to c_2 , the mapping ζ can be applied. Here, ζ is used with the parameter 2 as this state is the origin of the transition. In the configuration c_1 this is a set containing one single state: $\zeta(2, x, c_1) = \{4\}$. That statement confirms that (only) 4 has allocated the synchronization point x in this situation. There has no occupation been done yet. So for the transition t , the allocation has to be revoked and further actions can be done during the process. On the contrary, the configuration c_3 does not have an active allocation or occupation since $\zeta(2, x, c_3) = \emptyset$. Therefore, the synchronization point remains unchanged for the transition t .

4.3 User Interface

From a shell the user can start the Java application in console or window mode with several mandatory and optional parameters. As shown in Fig. 4, all configurations can be set intuitively in the window mode. The required input file can either be typed into the text-field on the top of the main content pane or it can be chosen by using a file dialog window. The temporary and the target output file are named accordingly. However, they can be defined individually, too.

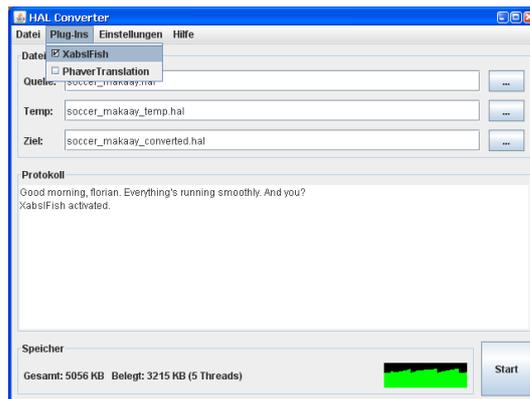


Fig. 4. HAL screen-shot before starting the translation.

4.4 XabslFish

The XabslFish plug-in defines constraints which check the input automaton for the proper structure. For this, the flattener algorithm must have processed the automaton. The regions must be copied according to their capacities, the variables must be globalized to provide an efficient handling of the XABSL symbol file and the synchronizations must be converted into ordinary variables. The variables must not be renamed during their globalization as they will later be translated by use of the configuration file.

The conversion of the flattened automaton starts with reading the appropriate configuration file. This is used to transform HAL variables to expressions, to basic behavior

calls or for their declaration in the output symbol file. In the second step of the translation, the symbol file is generated. A Boolean flag indicates if any symbol has to be written at all. In case that there is no symbol to write, the file will not be created. In consequence of that it can be decided if the future options will include this file or not. The third step is the major part of the translation. Here, the automaton tree is traversed and each node will be converted into an option. The subsequent automata become accessible via internal states while initial subsequent states keep their status. Each transition from a successor to another automaton is implemented into the state decision tree where discrete expressions are converted into actions in the target state. If all successor nodes are completed the own flow expressions of the current automaton will be converted into actions. This algorithm is processed recursively for each automaton.

4.5 An Example of Use

Let us now come back to our running example (Fig. 1). For purpose of clarity, the transition labels with the jump conditions and the discrete expressions are omitted as well as the flow expressions and invariants in the states. However, within this example there are several main features of synchronized hybrid automata covered. The soccer team has at least three players: one of type *A*, two of type *B*. Furthermore, there is exactly one ball on the field which can be interpreted as a resource with the maximum capacity 1. Due to the lack of space, for further details on the implementation, the reader is referred to [16].

5 Related Work

There are many related works on the specification of multiagent systems and also on software engineering of multiagent systems (see e.g. [2]), including Agent UML [13], where UML statecharts for modeling agent behavior are also considered, but not in the main focus of interest, however. We will therefore only briefly discuss some work on multi-robot coordination architecture, coordination mechanisms, and on formal specification of multi-robot systems.

As proposed by [17], multiagent systems have to deal with allocation and synchronization of tasks and concurrent subtasks. There are market-based approaches which support the coordination of the robot teams while each robot is paid revenue for each accomplished subtask and otherwise incurs cost for allocating team resources. ALLIANCE [14] is the name of an architecture for fault tolerant multi-robot cooperation. With this, it is possible to create multiagent systems that can deal with failures and uncertainties in the selection and execution of actions and dynamically changing environment.

In [10], coordination mechanisms are introduced in a concrete multi-robot architecture. The scenario description language Q in [9] concentrates on social agents that interact with humans. However, these articles deal with teamwork behaviors and interaction rather than translating a formalism to a specific hardware platform. Nevertheless, modeling and implementing multiagent systems is proposed in [2, 5]. Though this is

based on UML statecharts, yet, we use hybrid automata with timed synchronization in addition, in order to construct those systems.

The paper [1] presents a case study in multi-robot coordination, employing linear hybrid automata. By a rectangular approximation of the physical environment, the geometric regions in which a robot reaches a given goal faster with the help of communication, can be computed. [19] employs Petri nets for the specification of multiagent plans. Here, synchronization also can be expressed quite naturally within the Petri net framework. The MABLE language [18] is based on BDI agents, described textually, and a tool for modeling and verifying multiagent systems. However, the focus in the papers just mentioned is on verification or analysis and not on implementation and generating executable code on mobile robots, whereas we apply standard software engineering methods to real robots in this paper.

6 Discussion and Conclusions

XabslFish allows the behavior control of Aibo robots (or any other XABSL-driven robots) to be designed as a multiagent system with formal methods. Therefore, the performance of the robot is enhanced. The safety of a correct behavior control is based upon the fact that the translation from HAL to XABSL can precisely be adjusted manually for each input automaton. The application *XabslFish* supports the modeling of hybrid automata with timed synchronization and translates the specification to an understandable format for the target platform. The soccer domain is used as an example for multiagent systems, which have to act autonomously in a dynamic environment.

XabslFish translates multiagent system specifications from the hybrid automaton language HAL to XABSL. It is designed as a plug-in for the application, the HAL converter, that deals with hybrid automata. Even though it can translate the major part of a state machine automatically, some individual mappings must be defined in a configuration file. In summary, this paper exemplifies that multiagent systems can be specified by formal methods based on standard modeling procedures (namely state machines). It is also demonstrated, how by transformation techniques executable code can be generated, running on a mobile robot (namely the Aibo robot).

Future work will concentrate on an implementation of the formalism with constraint logic programming (CLP), which can be used for both, engineering and analysis of multiagent systems, following the lines of [15]. This will lead to an even more realistic knowledge engineering system.

References

1. R. Alur, J. M. Esposito, M. Kim, V. Kumar, and I. Lee. Formal modeling and analysis of hybrid systems: A case study in multi-robot coordination. In *World Congress on Formal Methods (1)*, pages 212–232, 1999.
2. T. Arai and F. Stolzenburg. Multiagent systems specification by UML statecharts aiming at intelligent manufacturing. In Castelfranchi and Johnson [4], pages 11–18. Volume 1.
3. T. Bernstein, D. Borns, C. Colmsee, K. Czarnotta, H. Germer, N. Nause, M. Pacha, R. Thomas, A. Vellguth, T. Wiebke, and M. Windler. HAL – hybrid automaton language.

- Technical report, Department of Automation and Computer Sciences, Hochschule Harz, 2006. Team project description (in German).
4. C. Castelfranchi and W. L. Johnson, editors. *Proceedings of the 1st International Joint Conference on Autonomous Agents & Multi-Agent Systems*, Bologna, Italy, 2002. ACM Press.
 5. V. T. da Silva, R. Choren, and C. J. P. de Lucena. A UML based approach for modeling and implementing multi-agent systems. *Autonomous Agents and Multiagent Systems*, 2:914–921, 2004.
 6. F. Dylla, A. Ferrein, G. Lakemeyer, J. Murray, O. Obst, T. Röfer, S. Schiffer, F. Stolzenburg, U. Visser, and T. Wagner. Approaching a formal soccer theory from behaviour specifications in robotic soccer. In P. Dabnichcki and A. Baca, editors, *Computers in Sport*, pages 161–185. WIT Press, Southampton, Boston, 2008.
 7. U. Furbach, J. Murray, F. Schmidberger, and F. Stolzenburg. Hybrid multiagent systems with timed synchronization – specification and model checking. In M. Dastani, A. El Fallah Seghrouchni, A. Ricci, and M. Winikoff, editors, *Post-Proceedings of 5th International Workshop on Programming Multi-Agent Systems at 6th International Joint Conference on Autonomous Agents & Multi-Agent Systems*, LNAI 4908, pages 205–220, Honolulu, 2008. Springer, Berlin, Heidelberg, New York.
 8. T. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, NJ, 1996. IEEE Computer Society Press.
 9. T. Ishida and S. Yamane. Introduction to scenario description language q. In *ICKS '07: Proceedings of the Second International Conference on Informatics Research for Development of Knowledge Society Infrastructure*, pages 137–144, Washington, DC, USA, 2007. IEEE Computer Society.
 10. G. A. Kaminka and I. Frenkel. Integration of coordination mechanisms in the BITE multi-robot architecture. In *ICRA-07*, 2007.
 11. M. Löttsch, M. Jüngel, M. Risler, and T. Krause. XABSL: The Extensible Agent Behavior Specification Language. URI: <http://www2.informatik.hu-berlin.de/ki/XABSL/>, 2006.
 12. Object Management Group, Inc. *UML Version 2.1.2 (Infrastructure and Superstructure)*, November 2007.
 13. J. Odell, H. V. D. Parunak, and B. Bauer. Extending UML for agents. In G. Wagner, Y. Lesperance, and E. Yu, editors, *Proceedings of the Agent-Oriented Information Systems Workshop at 17th National Conference on Artificial Intelligence*, pages 3–17, 2000.
 14. L. E. Parker. Alliance: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automaton*, 1998.
 15. C. Reinl, F. Ruh, F. Stolzenburg, and O. von Stryk. Multi-robot systems optimization and analysis using MILP and CLP. In P. U. Lima, N. Vlassis, M. Spaan, and F. S. Melo, editors, *Workshop 1: Formal Models and Methods for Multi-Robot Systems at 7th International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 11–16, Estoril, Portugal, 2008. International Foundation for Autonomous Agents and Multi-Agent Systems.
 16. F. Ruh. A translator for cooperative strategies of mobile agents for four-legged robots. Master thesis, Fachbereich Automatisierung und Informatik, Hochschule Harz, 2007.
 17. A. T. Stentz, M. B. Dias, R. M. Zlot, and N. Kalra. Market-based approaches for coordination of multi-robot teams at different granularities of interaction. In *Proceedings of the ANS 10th International Conference on Robotics and Remote Systems for Hazardous Environments*, March 2004.
 18. M. Wooldridge, M. Fisher, M.-P. Huget, and S. Parsons. Model checking multi-agent systems with MABLE. In Castelfranchi and Johnson [4], pages 952–959. Volume 2.
 19. V. A. Ziparo and L. Iocchi. Petri net plans. In *Proceedings of the Fourth International Workshop on Modelling of Objects, Components and Agents, MOCA'06*, pages 267–289, 2006.

Author Index

Althoff, Klaus-Dieter	55
Atzmueller, Martin	19
Bach, Kerstin	73
Diaconescu, Ion-Mircea	1
Dietrich, Jens	1
Giurca, Adrian	1, 7
Kluegl, Peter	19
Kluza, Krzysztof	31
Nalepa, Grzegorz	31, 43
Newo, Régis	55
Pascalau, Emilian	7
Peylo, Christoph	61
Puppe, Frank	19
Reichle, Meike	73
Ruh, Florian	85
Stolzenburg, Frieder	85
Wagner, Gerd	1
Wojnicki, Igor	43