



INTERNATIONAL  
HELLENIC  
UNIVERSITY

# **A Natural Language User Interface for a Semantic Web Agent**

**Samaras Dimitrios**

SID: 3301110010

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

*Master of Science (MSc) in Information and Communication Systems*

OCTOBER 2012

THESSALONIKI – GREECE



INTERNATIONAL  
HELLENIC  
UNIVERSITY

# A Natural Language User Interface for a Semantic Web Agent

**Samaras Dimitrios**

SID: 3301110010

Supervisor:

*Dr. Nick Bassiliades*

Supervising Committee Members:

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

*Master of Science (MSc) in Information and Communication Systems*

OCTOBER 2012

THESSALONIKI – GREECE

# Abstract

This dissertation was written as a part of the MSc in ICT Systems at the International Hellenic University.

One explanation as to why the Semantic Web has not quite caught on yet is that the barrier to entry is too high. During this dissertation the background and implementation of the “ACE Interface External Web Agent”, an open source semi-autonomous web platform serving as an intermediate with the RuleML Rule Responder System is described. Everyone, specialists in knowledge management to non-experts, can access it and pose queries the Rule Responder System. This is accomplished by automatically translating natural language queries written in Attempto Controlled English into the Reaction Rule Markup Language which is the knowledge representation format. Furthermore, the responses are provided to the user in a friendly and easy to understand way via simple web pages.

At this point I would like to thank *Dr. Nick Bassiliades*, academic coordinator in the School of Science and Technology at the International Hellenic University, for the trust he showed in me and the assignment of this dissertation project, the guidance and the assisting bibliography. I would also like to thank *Dr. Efstratios Kontopoulos*, academic assistant in the School of Science and Technology at the International Hellenic University, for his unselfish and unfailing support and understanding as my dissertation advisor. Last but not least I would like to thank my fellow classmates for their input and most of all for their moral support during the period in which this master’s thesis was conducted.

Samaras Dimitrios

October 15<sup>th</sup>, 2012



# Contents

<b>ABSTRACT .....</b>	<b>III</b>
<b>CONTENTS .....</b>	<b>V</b>
<b>1 INTRODUCTION.....</b>	<b>9</b>
<b>2 LITERATURE REVIEW .....</b>	<b>13</b>
2.1 CONTROLLED NATURAL LANGUAGES .....	13
2.1.1 <i>General-purpose CNLs</i> .....	14
2.1.2 <i>Business-purpose CNLs</i> .....	14
2.2 ATTEMPTO CONTROLLED ENGLISH (ACE) .....	15
2.2.1 <i>Introduction to Attempto Controlled English</i> .....	15
2.2.2 <i>ACE Syntax</i> .....	16
2.2.3 <i>ACE Tools</i> .....	20
2.2.4 <i>ACE to First-Order Logic</i> .....	27
2.3 PROCESSABLE ENGLISH (PENG) .....	27
2.3.1 <i>The PENG system</i> .....	28
2.3.2 <i>PENG Text Editor</i> .....	29
2.4 COMPUTER PROCESSABLE LANGUAGE (CPL) .....	30
2.4.1 <i>Interpreting CPL</i> .....	31
2.4.2 <i>CPL- Lite</i> .....	31
2.5 SEMANTICS OF BUSINESS VOCABULARY AND BUSINESS RULES (SBVR) .....	32
2.5.1 <i>SBVR Business Vocabulary</i> .....	33
<b>3 ADDRESSING THE PROBLEM.....</b>	<b>37</b>
3.1 THE PROBLEM.....	37
3.2 IMPLEMENTATION CHOICES .....	38
3.2.1 <i>Programming Language Selection</i> .....	38
3.2.2 <i>Selection of the Input and Communication Languages</i> .....	39
3.3 SEMANTIC WEB RULES AND RULEML.....	39
3.3.1 <i>RuleML</i> .....	41

3.3.2	<i>The RuleML Family of Sublanguages</i> .....	42
3.3.3	<i>Reaction RuleML</i> .....	43
3.4	THE RULE RESPONDER SYSTEM.....	49
3.4.1	<i>Interchange of Knowledge between Agents</i> .....	50
3.4.2	<i>Query Delegation among the Agents</i> .....	53
3.5	RELATED WORK.....	54
<b>4</b>	<b>CONTRIBUTION</b> .....	<b>57</b>
4.1	THE ACE INTERFACE EXTERNAL WEB AGENT .....	57
4.2	SYSTEM ARCHITECTURE.....	58
4.3	IMPLEMENTATION .....	58
4.4	USER INTERFACE .....	59
4.4.1	<i>Additional Content</i> .....	65
4.5	TRANSLATION PROCEDURE .....	68
4.6	CLASSES AND METHODS .....	69
4.6.1	<i>Parsing the input</i> .....	69
4.6.2	<i>Creating the Reaction RuleML request</i> .....	72
4.6.3	<i>Transmitting the request</i> .....	75
4.6.4	<i>Collecting and Storing the Response</i> .....	75
4.6.5	<i>Displaying the Response to the User</i> .....	76
4.7	AGENT INSTALLATION MANUAL.....	78
<b>5</b>	<b>CONCLUSIONS</b> .....	<b>79</b>
5.1	PROBLEMS MET DURING THE IMPLEMENTATION .....	80
5.1.1	<i>XPath Navigation Incompatibility</i> .....	80
5.1.2	<i>SymposiumPlanner-2012 Unavailability</i> .....	81
5.1.3	<i>Prolog Engine Necessity</i> .....	81
5.2	FUTURE IMPROVEMENTS .....	82
5.2.1	<i>Text analysis and Interpretation</i> .....	82
5.2.2	<i>Collaboration with ACE tools</i> .....	83
5.3	BUSINESS APPLICATIONS .....	83
	<b>BIBLIOGRAPHY</b> .....	<b>85</b>

# Figures and Schemas

Figure 1.1.1: Example response through the “ACE Interface External Web Agent” .....	11
Figure 2.2.1: Attempto APE (ACE Parser) Webclient snapshot.....	21
Figure 2.2.2: Consistency checking with RACE.....	22
Figure 2.2.3: Theorem proving with RACE .....	23
Figure 2.2.4: Query answering with RACE .....	24
Figure 2.2.5: Attempo demo AceWiki .....	25
Schema 1.3.1: Architecture of the PENG system.....	28
Figure 2.5.1: Simple View of SBVR model .....	33
Figure 2.5.2: Business Vocabulary development .....	33
Figure 3.3.1: The Semantic Web layers .....	40
Figure 3.3.2: The modularization of RuleML .....	42
Figure 3.3.3: Reaction RuleML applications.....	44
Figure 3.3.4: Reaction RuleML sublanguages .....	44
Figure 3.4.1: Animated representation of the External Agent interoperation with Rule Responder .....	50
Figure 3.4.2: Query Delegation with the use of a role assignment matrix for the RuleML symposium .....	54
Figure 4.2.1: Schematic representation of the systems architecture. ....	58
Figure 4.3.1: Schematic representation of the “ACE Interface External Web Agent” Implementation. ....	59
Figure 4.4.1: Semantic Web Agent front-end, Main page .....	61
Figure 4.4.2: Malformed or no input response .....	62
Figure 4.4.3: ACE Interface External Web Agent Query analysis result a. ....	63
Figure 4.4.4: ACE Interface External Web Agent Query analysis result b. ....	63
Figure 4.4.5: ACE Interface External Web Agent Query transformed to Reaction RuleML.....	64
Figure 4.4.6: ACE Interface External Web Agent Response page.....	65
Figure 4.4.7: About.html Snapshot.....	66
Figure 4.4.8: Cookbook.html Snapshot .....	67

Figure 4.5.1: Schematic representations of the three phases of ACE to Reaction RuleML and Reaction RuleML to HTML translation. ....	69
--	----



# 1 Introduction

The World Wide Web (WWW) or simply “Web” is following an exponential growth that has resulted in its transformation into a huge repository of information spread in billions of pages and shared among nearly 2.3 billion internet users [Internet World Stats, 2012]. Despite the WWW usage increase around the globe, the human ability to extract and use the information remains limited.

The evolutionary step of the Semantic Web over the current Web is to enable users to find, share, and combine information easily. It provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries [W3C Semantic Web Activity, 2011]. As it was originally envisioned, it enables machines to "understand" and respond to complex human requests based on their meaning. Such an "understanding" requires that the relevant information sources are semantically structured. The term was rounded up by Tim Berners-Lee, director of the *World Wide Web Consortium (W3C)*, who oversees the development of the proposed Semantic Web standards and defines the Semantic Web as "*a web of data that can be processed directly and indirectly by machines.*" [Berners-Lee, T. et al., 2001]. It has been described in a variety of terms: as a utopic vision, as the Web of Data, as a revolutionary change in the daily use of the Internet. Most of all the Semantic Web has inspired and urged many individuals into creating a variety of semantic technologies and applications.

As Feigenbaum L. [Feigenbaum L, 2007] mentions over the Semantic Web: "*While its critics have questioned the feasibility, proponents argue that applications in industry, biology and human sciences research have already proven the validity of the original concept. Scholars have explored the social potential of the Semantic Web in the business and health sectors, and for social networking.*"

To this end, various semantic web agents have been implemented in order to navigate and manipulate the structured data of the Semantic Web. Despite that, all these structured and machine readable data provided by the agents, due to the nature of computer

science, are not easily comprehended by humans and the rule and knowledge representation languages currently employed require a vast amount of familiarization time.

A key factor for spreading the Semantic Web usage among the internet users is to lower the entry barrier by offering a more user-friendly format. While the intuitiveness and expressiveness of natural language (e.g. English, Greek, etc.) makes it a logical candidate, it is inherently ambiguous and has therefore largely been ignored for formal applications. Fortunately, recent work has shown that principles from computational linguistics can be applied to translate between language and machine-interpretable logic. This is accomplished by using a subset of language that remains completely natural. One such controlled language is Attempto Controlled English (ACE), whose development over the past ten years has also included its own Attempto Parsing Engine (APE) and Reasoner (RACE).

The focus of this work is using ACE as the means to pose formal rules to the Organizational Agents of structured domains in the Semantic Web via a newly-developed Semantic Web Agent the “ACE Interface External Web Agent”. Specifically, queries written in ACE are taken as input, parsed and then mapped into formal representation in the XML-based, Rule Markup Language (Reaction RuleML). RuleML is the product of a long-standing effort toward standardizing rule markup on the Web and is compatible with other Semantic Web languages. Especially with the use of Reaction RuleML a variety of reaction rules serving in various domains are covered.

The outcome of this dissertation is the “ACE Interface External Web Agent” which allows non-programmers to pose their queries to the Rule Responder Organizational Agent for the RuleML Symposium 2012 and retrieve information about the symposium chairs of the RuleML symposia. As a trivial example, users may write:

What is the `r:contact-information` of the `n:general-chair-of-RuleML-2012-ECAI`?

Instead of something formal, like the following (the corresponding Reaction RuleML):

```
<?xml version="1.0" encoding="UTF-8"?>

<RuleML xmlns="http://www.ruleml.org/0.91/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ruleml.org/0.91/xsd"
xmlns:ruleml2012="http://ibis.in.tum.de/projects/paw#">
```

```

<Message directive="query-sync" mode="outbound">
  <oid>
    <Ind>SymposiumPlannerSystem</Ind>
  </oid>
  <protocol>
    <Ind>esb</Ind>
  </protocol>
  <sender>
    <Ind>User</Ind>
  </sender>
  <content>
    <Atom>
      <Rel>getContact</Rel>
      <Ind>ruleml2012ATecai_GeneralChair</Ind>
      <Var>Contact</Var>
    </Atom>
  </content>
</Message>
</RuleML>

```

Furthermore, instead of receiving a response in the same XML-format, the response is presented to the user in a graphical and easy to comprehend way, as shown in *figure 1.1.1*:

Fun	Ind
name	Grigoris Antoniou
email	antoniouATdomainname
phone	302810391624
title	Dr
role	General Chair
affiliation	University of Huddersfield
name	Guido Governatori
email	guidoDOTgovernatoriATnictaDOTcomDOTau
phone	610733652907
title	Dr
role	General Chair
affiliation	NICTA

**Figure 1.1.1: Example response through the “ACE Interface External Web Agent”**

During the chapters that follow, the background knowledge that covers the implementation of the project, the design and implementation as well the projects' future development topics are covered.

In the 2<sup>nd</sup> chapter of this dissertation "Literature Review", the background and the state of the art in the field of controlled natural languages (CNLs) are described. The use of a controlled natural language, ACE for this project, is the first step that makes the Semantic Web Agent that is created accessible by non-experts. In the 3<sup>rd</sup> chapter, "Addressing the Problem", the problem that initiated this master thesis is explained. Additionally, since Rule Responder is the basic infrastructure that the ACE Interface External Web Agent is collaborating with, a brief description is given over its architecture and its knowledge interchange format (Reaction RuleML). The 4<sup>th</sup> chapter, "Contribution", focuses on the development of the "ACE Interface External Web Agent". The system architecture is described along with the web pages that the user interacts with and their functionality. Getting into more detail, the translation process from ACE to Reaction RuleML and from Reaction RuleML to the HTML final response is explained in steps (design) as well as in the programming language with code fragments (implementation). Finally in the 5<sup>th</sup> chapter, "Conclusions", final conclusions are drawn over the project implementation and improvements left for future implementation are collocated. What is more, additional applications of the implementation are presented beyond the boundaries of the academia.

## 2 Literature Review

In this chapter, the background and the state of the art in the fields of controlled natural languages (CNLs) are described. CNLs in general are discussed first: An overview of the main subtypes of CNLs is featured in Section 2.1. In sections 2.2 to 2.5, a closer look is taken at the language Attempto Controlled English (ACE) which is the language to be used throughout this master thesis, and three more CNLs are described: Processable English (PENG), Computer Processable Language (CPL) and Semantics of Business Vocabulary and Business Rules (SBVR).

### 2.1 Controlled Natural Languages

*Controlled natural languages (CNLs)* are subsets of a full natural language (any human speaking language) with explicit restrictions on grammar, lexicon, and style following strict rules. They have an unambiguous syntax and clean semantics leading to an error reduction on the represented knowledge [Wyner A., 2009]. CNLs have been deployed to the writing of specifications serving the knowledge representation and translation tasks in a two-fold way, meaning that they are comprehensive and usable by both human users and computers alike.

All CNLs are either directly or indirectly related to formal logic. Further advantages that these languages offer are: the ease of learning, as they bare strong resemblance to natural languages, along with the ease of use, as they don't require any specialized type of knowledge.

General purpose (see Section 2.1.1) as well as business purpose (see Section 2.1.2) machine-oriented CNLs have been developed to facilitate the translation of technical documents and assist in the knowledge extraction and processing by machines. Additionally, non – specialist users are supported by intelligent tools and agents developed for writing specification texts for knowledge representation in a familiar notation without the need to formally encode the information [Schwitter, R., 2004].

### 2.1.1 General-purpose CNLs

General-purpose CNLs have been developed to be exploited in various domains without a specific application scenario in mind. The general-purpose CNLs discussed in this chapter are ACE (2.2), PENG (2.3) and CPL (2.4).

*Attempto Controlled English (ACE)* is a general-purpose controlled English language that can be defined in a concrete and declarative way [Fuchs, N. E., 2012]. ACE comes with a parser –ACE parser–that translates ACE text into a logic-based representation as well as editing tools, anaphorically: ACE Editor, APE (ACE parser), RACE (ACE Reasoner), ACE View, AceRules and AceWiki that facilitate its use. It is the language mainly used throughout this dissertation and for this reason; it will be described more thoroughly in the next section (2.2). Also ACE has been applied in the area of the Semantic Web [Berners-Lee, T. et al., 2001] and can be mapped into OWL and SWRL.

*Processable English (PENG)* is a CNL bearing strong resemblance to ACE but covers a smaller subset of natural English. It is designed for an incremental parsing approach and was one of the first languages used within a special editor with predictive features to tell the user how a partial sentence can be continued [Schwitter, R., 2004].

*Computer Processable Language (CPL)* is a controlled English language developed at Boeing research facilities. Instead of applying a small set of strict interpretation rules, the CPL interpreter resolves various types of ambiguities. Apart from being the more liberal controlled language, CPL probably has the most sophisticated implementation of the error messages approach [Kuhn, T., 2009].

### 2.1.2 Business-purpose CNLs

CNLs for business purpose allow companies to express business rules about their application area in a structured and coherent way. The end users are business people with no particular background in knowledge representation and logic and CNLS can offer them an intuitive representation for knowledge management. The business CNL that will be briefly discussed is *Semantics of Business Vocabulary and Business Rules (SBVR)* (2.5).

SBVR Structured English is defined informally by sets of guidelines based on experiences of best practice in rule systems. It is not strictly formal language, and therefore does not come with a parser as those implemented in general-purpose CNLs [Chapin, D., 2010].

## 2.2 Attempto Controlled English (ACE)

### 2.2.1 Introduction to Attempto Controlled English

*Attempto Controlled English (ACE)* is a controlled natural language specifically designed for requirements specifications and knowledge representation. It is a precisely defined subset of English with a restricted grammar in the form of a small set of construction and interpretation rules that can automatically and unambiguously be translated into full first-order logic, making ACE readable both by humans and machines. Although all ACE sentences are syntactically acceptable English, not all English sentences are allowed in ACE, meaning that while it seems completely natural, is in fact a language that has to be learned [Fuchs, N. E. et. al., 2005]. In brief, domain specialists and un-familiarized users alike are provided with a controlled language to express and process knowledge in a familiar way (English language subset) and to combine this with the rigor of formal specification languages.

ACE has been under development at the University of Zurich since 1995. In 2004, ACE was adopted as the controlled natural language of the EU Network of Excellence *REWERSE (Reasoning on the Web with Rules and Semantics)* [Bry, F., 2008]. In 2008, ACE version 6.0 was announced [Fuchs, N. E. et. al., 2008], currently in version 6.6. Apart from its syntax, various tools have been developed for the implementation and control of the language, which are briefly presented in subsequent subsections.

Among the fields it applies, ACE has been exploited in order to improve the usability of policy languages such as the Protune policy language. Also a translator that converts ACE texts into the Rule Markup Language-RuleML (TRANSLATOR) has been implemented [Hirtle, D. Z., 2006]. Another translator has been developed that translates a subset of ACE into the REWERSE Rule Markup Language-R2ML [Lukichev, G. W., 2007]. Additionally, ACE has been used as a front-end for the Process Query Language (PQL) that allows users to query MIT's Process Handbook. It has been

shown that queries expressed in ACE and automatically translated into PQL provide a more user-friendly interface to the Process Handbook [Juri Luca De Coi, et al., 2009]. The full list of ACE application domains includes software and hardware specifications, agent control, legal and medical regulations, and ontology construction.

### 2.2.2 ACE Syntax

ACE syntax consists of vocabulary as well as grammar. The vocabulary contains a full-form common lexicon of close to 100,000 entries, function words, predefined fixed phrases (e.g. ‘it is necessary’, ‘there is’) and content words (nouns, proper names, verbs, adjectives and adverbs). The ACE grammar, on the other hand, defines and constrains the form and the meaning of ACE text and sentences. It comprises of sets of construction rules (syntax) and sets of interpretation rules (semantics).

The syntactic rules and examples presented later on are adopted from: [Kuhn, T., 2011; Fuchs et.al., 2008; Fuchs, N. E. et. al., 2005; Juri Luca De Coi, et al., 2009].

#### 2.2.2.1 ACE Vocabulary

The set of content words is infinitely large and dynamic with a full-form common lexicon of close to 100,000 entries. Users can also import domain specific lexica of content words or missing content words can be temporarily introduced with their respective word class (e.g. *A a:trusted man a:deliberately v:backs-up the n:web-page of the n:external-agent-web-service*). Additionally, with the use of ACE parser, unknown content words can be added to the correct word class based on its content.

As predefined function words: determiners, quantifiers, prepositions, coordinators, negation words, pronouns, query words and numbers are used. Also, predefined fixed phrases such as “there is /there are...such that” and “it is false that” add to the set of limited and unchanging function words.

#### 2.2.2.2 ACE Grammar

The construction rules define admissible sentence structures for ACE and comprise of words, phrases, declarative sentences forming ACE text.



ACE interpretation rules specify the correct meaning of an ACE sentence out of the conceivable ones. This is due to ambiguity that full English language has compared to ACE when interpreted.

For complete ACE 6.6 interpretation rules refer to [Kuhn, T., 2011]

*ACE texts* consist of a sequence of anaphorically interrelated simple, ‘there is/are’ composite and imperative sentences.

*Simple ACE sentences* follow the structure:

*Subject + verb + complement + adjuncts*

e.g.: “A customer inserts two cards manually in the morning.”

In Simple sentences format, subject and verb are present. *Complements* (direct and indirect objects) accompany *transitive verbs* (“insert something”) and *ditransitive verbs* (“give something to somebody”). *Adjuncts* (adverbs and prepositional phrases) that modify the verb are optional.

An alternative for simple sentences is:

*‘There is/there are’ + noun phrase*

e.g.: “There is a customer.”

Adjectives, adverbs possessive nouns and prepositional phrases, or variables as appositions are added to specify nouns and specify the situation in detail.

e.g.: “A customer inserts some cards manually.” (Adverb ‘some’ adds detail to verb ‘insert’).

e.g.: “A customer inserts some cards into an ATM.” (prepositional phrase added).

Furthermore *relative clauses* enhance the noun modifications (‘that’).

e.g.: “A customer who is trusted inserts two cards that he owns.”

*Composite ACE sentences* are created by coordination, subordination, quantification and negation of simple ACE sentences

- Coordination by “and” and ‘or’ is achieved between sentences, phrases and relative clauses.

e.g.: “A customer inserts a card and an automated teller checks the code.”

e.g.: “A customer owns a card that is invalid or that is damaged.”

- Subordination is constructed via if-then sentences, modality and sentence subordination.

If-then sentences define hypothesis and necessary conditions.

e.g.: “If a card is valid then the customer inserts it”

Modality gives the user the ability to express possibility and necessity.

e.g.: “A trusted customer can insert a card.”

With sentence subordination sentences are used as objects

e.g.: “It is not provable that a customer inserts his own card”

- Through quantification we refer to the whole or a specific object of the class. It is achieved by indefinite determinants.

e.g.: “Every customer inserts a card.”

e.g.: “There is a card that every customer inserts”

- Negation is used to express that something is not the case.

e.g.: “A customer does not insert a card.”

e.g.: “No customer inserts more than 2 cards.” (negation for all objects using ‘no’).

e.g.: “It is false that a customer inserts a card.” (Sentence negation used to negate the whole statement).

e.g.: “It is not provable that a customer inserts a card.” (Negation as failure).

*Interrogative ACE sentences* are formed as ‘Yes/No –queries’ and ‘Wh- queries’

e.g.: “Does a customer insert a card?”

e.g.: “What does a customer insert?”

Formation of queries for interrogation of text is accomplished with the use of a question mark in the end of the sentence.

*Imperative ACE sentences or commands* are formed with the use of exclamation mark in the end of the sentence.

e.g.: “John and Mary, wait!”

### **2.2.2.3 Constraining Ambiguity in ACE**

The ambiguity of full English is constrained in ACE by two means. Ambiguous constructs are replaced by unambiguous alternatives in their place and the ambiguous constructs that remain are interpreted deterministically on the basis of some interpretation rules. The user either accepts the assigned interpretation or can rephrase the text to get another one.

e.g.: “A customer inserts a card that is valid and opens an account.”

This is interpreted as: There is a customer X1. There is a card X2. The card X2 is valid. The customer X1 inserts the card X2. The customer X1 opens an account.

To express that the card opens the account the ACE text has to be: “A customer inserts a card that is valid and that opens an account.”

### **2.2.2.4 Anaphoric References in ACE**

ACE copes with anaphoric references with the use of *the definitive article*, *personal pronouns* or via *variables*. All the anaphoric references during the processing of ACE text are replaced by the most recent and most specific accessible noun phrase that agrees in gender and number [Kaljurand, K., 2011b].

e.g.: “A customer enters a card and a code. If the code is valid then the ATM accepts the card.” (Using the definitive article).

e.g.: “A customer enters a red card and a blue card. The card is correct.” (The most recent defines that the blue card is correct).

while

“A customer enters a red card and a blue card. The red card is correct.” (The most specific defines that the red card is correct).

“A customer does not enter a card. The card is correct.” (Cannot refer to a card).

e.g.: “A customer enters a card X and a code Y. If Y is valid then the ATM accepts X.” (Use of variables X,Y)

### 2.2.3 ACE Tools

This section gives a brief overview of the main ACE-related software tools.

#### 2.2.3.1 Attempto Parsing Engine (APE)

*Attempto Parsing Engine (APE)* is a tool that translates ACE text unambiguously into *Discourse Representation Structures (DRSs)*. The basic features of the parser are:

- Technical feedback on the input text,
  - Various logical forms and representations deriving from DRS (First-Order Logic form, DRS in XML, OWL & SWRL),
  - ACE paraphrase of the input, translating DRS into a subset of ACE,
  - Error debugger pinpointing the location and cause of errors,
  - Model solutions to the problem,
  - ACE syntax is implemented in the form of nearly 200 definite clause grammar rules using feature structures,
  - A complete lexicon of ACE is fitted into APE,
  - User defined lexica can be added or replace the lexicon
- [Juri Luca De Coi, et al., 2009].

APE is implemented in SWI-Prolog and released under the LGPL open source license. The distribution also includes among other tools the DRS verbalizer, translator from ACE to OWL/SWRL [Fuchs, N. E. et. al., 2008b]. APE has a command-line client and can be also used from Java, or over HTTP as a REST web service or from its demo client [APE Webservice., 2010 ; APE Webclient Help., 2008].

In *figure 2.2.1* can be seen the APE Webclient and the produced output of input text “If somebody does not belong to the n:organizational-comitee then he is a symposium-attendant” analysis operation.

Hide menuHelp

Show

☐ Input text
☒ Paraphrase
☒ DRS
☐ DRS XML
☐ FOL
☐ TPTP
☐ OWL FSS
☐ OWL XML
☒ Tokens
☐ Syntax

Options

☐ Guess unknown words
☐ Do not use Clex

Lexicon

☐ Reload the lexicon from URL

If somebody does not belong to the n:organizational-committee then he is an n:symposium-attendant.

↑↓

Analyse

overall: 0.856 sec (tokenizer: 0.000 parser: 0.002 refres: 0.000) :: Mon Oct 01 2012 10:43:39 GMT+0300 (Θερινή ώρα GTB)

	Type	Sentence	Problem	Suggestion
warning	anaphor	1	The definite noun phrase 'the organizational-committee' does not have an antecedent and thus is not interpreted as anaphoric reference, but as a new indefinite noun phrase.	If the definite noun phrase 'the organizational-committee' should be an anaphoric reference then you must introduce an appropriate antecedent.

PARAPHRASE

If there is somebody X1 and it is false that X1 belongs to an n:organizational-committee then X1 is a n:symposium-attendant

DRS

[ ]

[A]

object (A,somebody,countable,na,eq,1) -1/3

NOT

[B,C]

object (C,organizational-committee,countable,na,eq,1) -1/11

predicate (B,belong,A) -1/6

modifier\_pp (B,to,C) -1/7

=>

[D,E]

object (D,symposium-attendant,countable,na,eq,1) -1/18

predicate (E,be,A,D) -1/14

TOKENS

[[^,' If',a,'-body',does,not,belong,to,the,n,:, 'organizational-committee',then,he,is,an,n,:, 'symposium-attendant','.' ]]

**Figure 2.2.1: Attempto APE (ACE Parser) Webclient snapshot**

### 2.2.3.2 Attempto Reasoner (RACE)

RACE offers three main functions: (a) Consistency checking and information about the axioms that lead to text inconsistency, (b) Textual entailment and (c) Query answering of ACE text.

RACE checks whether ACE axioms entail or not the ACE theorems/ACE queries and underlines the axioms entailing theorems, respectively queries or the parts of the ACE theorems/ACE queries that could not be entailed. Additionally, consistency checking is applied over ACE axioms. All the first-order logic subset of ACE is covered by the

Reasoner with the exception of imperative sentences; negation as failure and the modal operators “*may*” and “*should*” [Fuchs, N. E., 2011].

The output as well as the input of the Reasoner is in ACE. Also, no previous knowledge of formal logic or theorem proving is required by the user, nor it is required to understand the way Reasoner works or any user control is necessary over the reasoning process [Kaljurand, K., 2011c ; RACE Web Client Help. (n.d.)].

RACE can be called via a SOAP web service and can be easily accessed via a web client [Kaljurand, K., 2011c]. A presentation of the RACE with an example can be seen in the following figures, *figure 2.2.2*, *figure 2.2.3* and *figure 2.2.4*.

In *figure 2.2.2* text input “*Every n:attendant is a person. Every n:committee-member is an n:attendant. p:Dimitris is an n:attendant. p:Strator is a n:committee-member.*” is checked for axioms consistency with the use of RACE. Annotated ACE input is used as annotated ACE format is going to be using throughout the implementation of the project. In the second figure (2.2.3) the theorem “*There is a person.*” is proved according to the predefined and consistent axioms. In the third figure (2.2.4) the query “*Who is an n:attendant?*”, is answered based on the axioms and the answer process is depicted in the steps.

The screenshot displays the RACE web client interface. At the top, there are two buttons: "Show Parameters" and "Show Help". Below these is a large text area labeled "Axioms" containing the input text: "Every n:attendant is a person. Every n:committee-member is an n:attendant. p:Dimitris is an n:attendant. p:Strator is a n:committee-member." Below the text area are three buttons: "Check Consistency" (highlighted in blue), "Prove", and "Answer Query". Below these buttons is a status bar with a "Check Consistency" button. At the bottom, a status bar shows the execution times: "overall time: 0.465 sec; RACE time: 0.005 sec". Below this, a green box displays the "Axioms" and "Parameters" sections, both containing the same input text. At the bottom of the interface, a message states "Axioms are consistent."

**Figure 2.2.2: Consistency checking with RACE**

Theorems

There is a person.

overall time: 0.465 sec; RACE time: 0.007 sec

**Axioms:** Every n:attendant is a person. Every n:commitee-member is an n:attendant.  
p:Dimitris is an n:attendant. p:Strator is a n:commitee-member.

**Theorems:** There is a person.

**Parameters:**

The following minimal subsets of the axioms entail the theorems:

- Subset 1
  - 1: Every n:attendant is a person.
  - 2: Every n:commitee-member is an n:attendant .
  - 4: p:Strator is a n:commitee-member .
- Subset 2
  - 1: Every n:attendant is a person.
  - 3: p:Dimitris is an n:attendant .

**Figure 2.2.3: Theorem proving with RACE**

Check Consistency
Prove
Answer Query

Query

overall time: 0.312 sec; RACE time: 0.008 sec

**Axioms:** Every n:attendant is a person. Every n:committee-member is an n:attendant. p:Dimitris is an n:attendant. p:Strator is a n:committee-member.

**Query:** Who is an n:attendant?

**Parameters:**

The following minimal subsets of the axioms answer the query:

- Subset 1
  - 1: Every n:attendant is a person.
  - 2: Every n:committee-member is an n:attendant .
  - 4: p:Strator is a n:committee-member .
  - Substitution: who = (at least 1) person
- Subset 2
  - 1: Every n:attendant is a person.
  - 3: p:Dimitris is an n:attendant .
  - Substitution: who = (at least 1) person
- Subset 3
  - 2: Every n:committee-member is an n:attendant .
  - 4: p:Strator is a n:committee-member .
  - Substitution: who = (at least 1) attendant
- Subset 4
  - 2: Every n:committee-member is an n:attendant .
  - 4: p:Strator is a n:committee-member .
  - Substitution: who = (at least 1) committee-member
- Subset 5
  - 2: Every n:committee-member is an n:attendant .
  - 4: p:Strator is a n:committee-member .
  - Substitution: who = Strator
- Subset 6
  - 3: p:Dimitris is an n:attendant .
  - Substitution: who = (at least 1) attendant
- Subset 7
  - 3: p:Dimitris is an n:attendant .
  - Substitution: who = Dimitris

**Figure 2.2.4: Query answering with RACE**

### 2.2.3.3 AceView Ontology and Web Editor

AceView is an ontology and rule editor that uses ACE in order to create, view, edit and query OWL ontologies and SWRL rule sets. Snippets (sequences of anaphorically



linked ACE sentences) which form the ACE text are parsed and translated to OWL/SWRL automatically when asserted to the editor [Kaljurand, K., 2011a].

AceView is integrated in the Protégé framework and all of its operations are completed via the Protégé menu, allowing the user to view and edit the ACE text at several levels:

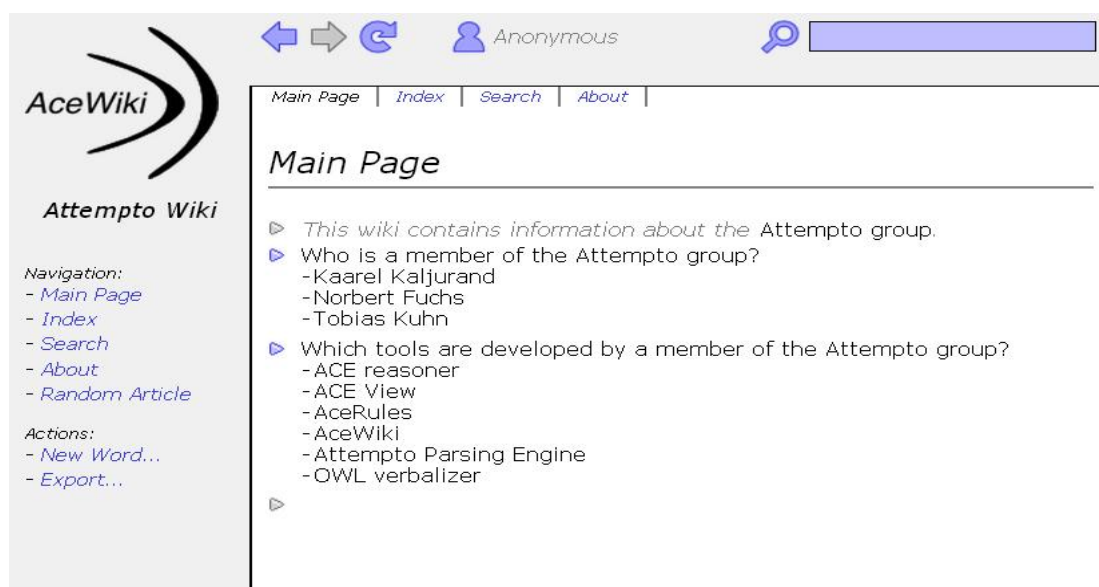
- Word level that enables the user to access individual words.
- Snippet level, allowing access to asserted declarative, asserted interrogative and entailed snippets.

Additionally, vocabulary can be sorted in various ways and complete ACE text can be filtered, sorted and searched through the editor [Kaljurand, K., 2007a]. Furthermore, the user can switch at any moment to one of the other Protégé views for further ontology editing.

#### 2.2.3.4 AceWiki

AceWiki is a semantic wiki that uses ACE to formally represent its content. The expressiveness of formal ACE statements is exploited for the development of a semantic wiki where users can edit the textual content in a collaborative manner over the web while giving information well-defined meaning for computers and people to cooperate [Kuhn, T., (n.d.)].

In figure 2.2.5 the AceWiki, a demonstration purpose Attempto wiki is displayed.



**Figure 2.2.5: Attempo demo AceWiki**

### **2.2.3.5 AceRules**

AceRules is a rule system prototype with a multi-semantics architecture, using ACE both as input and as output language. AceRules is designed for forward-chaining interpreters that calculate the complete answer set [Kuhn, T., 2007]. One of the properties of AceRules is the semantics incorporated by the rule system, namely: courteous logic programs, stable models and stable models with strong negation. Another property is the representation of two types of negation, strong negation and negation as failure. Finally, construction of valid programs is achieved through AceRules by intelligent grouping.

AceRules comes with three interfaces, a web service for integration of AceRules in any program and two more interfaces for human interaction. A technical interface for advanced users and one for the end-users unfamiliar to formal notations [Kuhn, T., 2007].

### **2.2.3.6 OWL verbalizer (ACE to OWL and SWRL)**

The initial intention of ACE is to offer the necessary naturalness in the expression when undertaking knowledge engineering tasks. In order to make ACE interoperable with some of the existing Semantic Web languages several mappings have been developed between ACE and OWL, SWRL and DL-Query [Kaljurand, K., 2007a], [Kaljurand, K., 2007b]. Respectively the inverse mapping has also been implemented, OWL 2 to ACE, verbalizing this way the OWL 2 ontologies as ACE text.

These mappings provide a user friendly front-end syntax editor for OWL 2 and SWRL which makes the differences between OWL 2 and SWRL transparent and provides “linguistically motivated syntactic sugar” [Juri Luca De Coi, et al., 2009].

### **2.2.3.7 ACE Editor**

The ACE Editor demonstrates how editing of ACE texts can be done in a convenient way. The ACE Editor is not a finished tool but rather a general basis to create domain-specific tools on top of it. [Fuchs, N. E., 2006]

#### 2.2.4 ACE to First-Order Logic

ACE text can be translated into *Discourse Representation Structures (DRS)* [Kaljurand, K., 2011b]. DRSs use a syntactic variant of the language of standard first-order logic which is extended by some non-standard structures for modality, sentence subordination, and negation as failure. The characteristics of DRSs are:

- Use a small number of predefined predicates,
- Representation of information derived from words as arguments of the predefined predicates,
- Appearance of eventuality types,
- Use of lattice-theoretic representation of objects that allows the encoding of plurals in first-order language,
- Quantity information is contained [Fuchs et al., 2005].

A more thorough and comprehensive description of DRSs, notation and practical examples is found on [Kaljurand, K., 2011b].

### 2.3 Processable English (PENG)

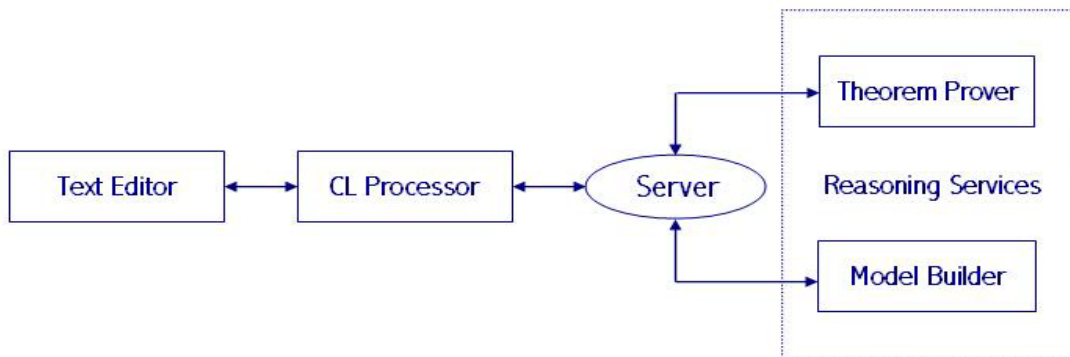
Processable English (PENG) is a machine-oriented controlled natural language, consisting of a restricted controlled grammar and controlled lexicon, designed for non-specialists to write precise specification texts in a seemingly informal notation [Schwitter, R., 2002]. PENG [Schwitter, R., 2009] is similar to ACE but the subset of English covered by the vocabulary is limited, yet fully tractable. The vocabulary and syntactic correctness when writing in PENG is ensured by a intelligent text editor ECOLE with an intelligent feedback mechanism that has been developed in order to assist the creation of well-formed linguistic structures that are translated unambiguously into first-order logic via discourse representation structures (DRSs), a technique similar to the one applied in ACE. On the contrary to ACE that uses the produced discourse representation structure for the resolution anaphoric references, PENG transforms the DRS into a format processable by a top-down chart parser and resolves anaphoric references during the parsing process in the same time a discourse representation structure is constructed. Additionally PENG was the first CNL that was supported by a predictive editor that automatically detects what the user's possible input and au-

to-completes it [Schwitter, R. et al., 2003]. The text created can be instantly validated for its acceptability constraints using a third-party reasoning service, like the OTTER theory prover [Schwitter, R., (n.d.)b]. Additionally, the logic theory resulting can be used for question answering. Texts written in PENG look seemingly informal and are easy to write and understand for humans but have first-order equivalent properties [Schwitter, R., 2004]

PENG can be used for similar applications to ACE language but has recently been mostly used for the construction of an interface for a situation awareness system [Baa-der F., et al., 2009].

### 2.3.1 The PENG system

The PENG system's architecture includes four components. The intelligent text editor ECOLE, the controlled language (CL) processor, a server, and reasoning services (a theorem prover and model builder) as depicted in the schema (2.3.1) below.



***Schema 1.3.1: Architecture of the PENG system [Schwitter, R., 2004].***

The CL processor is communicating with ECOLE through a socket interface. It runs as a client service and connects through a server to the reasoning services which are running separate client processes and are the fourth component of the PENG system. The reasoning services check the acceptability constraints of an inserted text and answer questions about a specified piece of domain improving the knowledge acquisition process.

### 2.3.2 PENG Text Editor

Controlled natural language is the means of communication between the user and the editor, while the editor dynamically enforces the grammatical restrictions of the language over the inserted text and displays the interpretation of a sentence in the form of a paraphrase in a CNL. As text is inserted the allowed syntactic structures that can follow are automatically proposed by the look-ahead function of the editor guiding the writing process. Additionally a spelling checker module is a part of the editor.

The controlled lexicon of the PENG system consists of a *base lexicon* and a *user lexicon*. The base lexicon contains frequent content words (proper nouns, common nouns, verbs, adjectives, and adverbs) and pre-defined function words (determiners, prepositions, coordinators, subordinators, and disambiguation markers) which build the syntax of the CNL.

The user defined lexicon can be extended with domain-specific content words by the author with the use of the pop-up lexical editor interferes when unknown words are inserted [Tilbrook, M. & Schwitter, R., 2006].

Every sentence is checked dynamically for constraints violations so that the user gets immediate feedback from the editor.

#### 2.3.2.1 PENG online

PENG Online is a version of the PENG editor implemented as a Java applet which runs in a web browser and communicates with the server via a socket interface. It can be used for summarizing and augmenting web-pages as well as other documents written in controlled natural language. This authoring tool offers built-in browser functionality for viewing web pages and provides an RSS mode for summarizing the content of these web pages in a CNL. Child processes are created for every java client connecting to the Prolog server to manage multiple user sessions [Tilbrook, M. & Schwitter, R., 2006]. The resulting text from the editor can be stored in an RSS feed format accompanied by the user defined lexicon and later on exported as an XML document [Schwitter, R., (n.d.)b].

### 2.3.2.2 PENG light

PENG translates generated text and posed questions into first-order notations via discourse representation structures for automated reasoning but not backwards into a CNL. In the case of PENG Light the same grammar as PENG is used to translate sentences and questions into a first-order notation (augmented with syntactic information) and this notation is used as a starting point for generating answers to questions in a CNL. PENG Light distinguishes between simple, complex, and interrogative sentences [Schwitter, R., 2009], [Schwitter, R., (n.d.)a].

## 2.4 Computer Processable Language (CPL)

*Computer Processable Language (CPL)* is a controlled language developed at Boeing Research and Technology. Formal knowledge representation (KR) with semantics derives from the interpretation of CPL in order to be used in reasoning and question answering [Clark, P. et al., 2005].

In contrast to ACE which applies to restricted interpretation rules, and in contrast to PENG, which relies on a predictive editor, the CPL interpreter directly resolves various types of ambiguities using heuristic rules for a variety of language processing activities [Peter, C. et al., 2010].

For a brief description of the CPL syntax and grammar: A CPL sentence has the following structure:

*subject + verb + complements + adjuncts*

a structure similar to that of ACE simple sentences. Complements are obligatory while adjuncts are optional modifiers for a sentence to be complete. Basic sentences can be conjoined together using the keyword “AND”. CPL grammar includes handling of prepositional phrases, compound nouns, ordinal modifiers, proper nouns, adjectives, passive and gerundive forms, relative clauses, and limited handling of conjunctive co-ordination. Definite reference substitutes pronouns.

CPL accepts three types of sentences: ground facts which are basic sentences, questions (“wh-” type of questions e.g. “what...”, “who...”), and rules implemented by seven rule templates, for logical implications, preconditions and actions. “Sentences that express states add facts to a situation, and sentences that express actions trigger rules that update the situation, reflecting the changes that the action has on the situa-

tion. The user can ask questions about an emerging situation directly in CPL” [Schwitter R., 2010].

#### 2.4.1 **Interpreting CPL**

As every CPL is interpreted instantly the user gets a continuous feedback in order to correct misinterpretations.

There are three main steps for the interpretation of CPL sentences to logic. First of all the sentences are parsed with the use of *SAPIR*, a bottom-up, broad coverage chart parser, using attachment patterns for common word preference, which are stored in a manually constructed database.

Second step, is the generation of an intermediate simplified “logical form” (LF). The outcome is a normalized tree structure with logic-type elements, generated according to rule sets, containing variables for noun phrases and additional expressions for other sentence elements [Clark, P. et al., 2005].

During the third step, knowledge machine (KM) assertions are generated based on the logical form. KM is a frame-based, well-defined semantics knowledge representation language. The KM interpreter which applies to user defined rules is responsible for reasoning, including reasoning about actions using a situation calculus mechanism [Schwitter, R., 2010].

#### 2.4.2 **CPL- Lite**

CPL-Lite was created to be used by knowledge engineers understanding the ontologies and its vocabulary that needed a controllable and comprehensive way for question posing [Clark, P. et al., 2005].

While CPL relies on heuristics searching for the best interpretation, CPL-Lite is a slimmed down version of CPL that can be interpreted deterministically in a similar manner to PENG (Schwitter R. , Controlled Natural Languages for Knowledge Representation, 2010). These two languages have the same expressiveness but the grammar of CPL-Lite is even more restricted to a set of template sentences and the vocabulary is equally restricted and requires the generation of verbose sentences that map directly to ontology concepts.

## 2.5 Semantics of Business Vocabulary and Business Rules (SBVR)

*SBVR* is a completely declarative meta-model with a solid logical foundation that was developed with the intention to form the basis for formal and detailed natural language providing declarative descriptions of a complex entity's definition and rules, such as a business. Apart from serving as a terminological dictionary it provides the context for sharing these meanings and terms [Chapin, D., 2010].

It was developed in response to Object Management Group's (OMG) request for proposal "Business Semantics of Business Rules". The development team consisted of 17 organizations in 7 countries. SBVR formalizes complex compliance rules, such as operational rules for an enterprise, security policy, standard compliance, or regulatory compliance rules [Hall, J., 2006]. Every rule is typed and respectively annotated by the user with the use of template applied on a text editor. No particular language for vocabulary and rules is standardized for common concepts such as date and time, quantities, currencies and units of measure. Such a case was not the intention of SBVR, yet it is stated in bibliography addressed in this section that the need for a common vocabulary is needed in business use of the language.

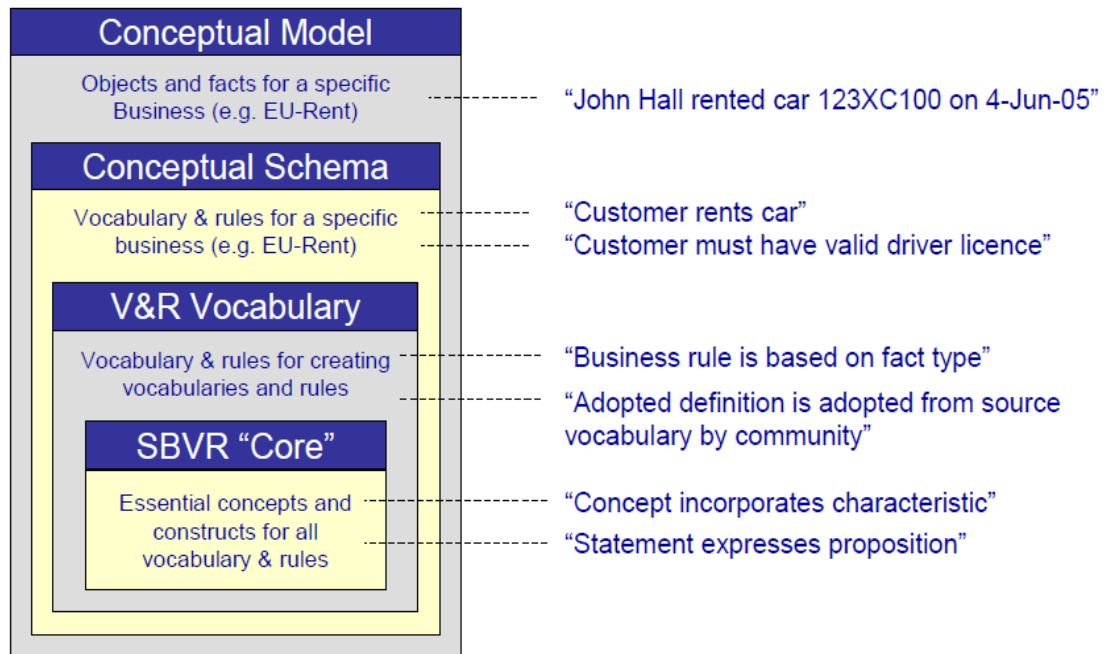
After being adopted by the OMG in 2005 and released in December 2007 (SBVR 1.0), SBVR is an intergraded part of the *Model Driven Architecture (MDA)*. SBVR describes business concepts and requirements without addressing their implementation. Except from applications in commerce, for example manufacturing and finance, SBVR also fits domains such as education, government, medicine and law.

For the vocabulary and rule examples that will follow "Structured English", a restricted form of the natural language is used as defined by the OMG's standard so that consistent interpretation of the vocabulary and rules is enabled, using font styles: nouns are underlined, *verbs* are in italics, literal values and instance names are shown in double underlines, **keywords** are in bold and uninterpreted text is shown in normal font style [Linehan, M. H., 2009].

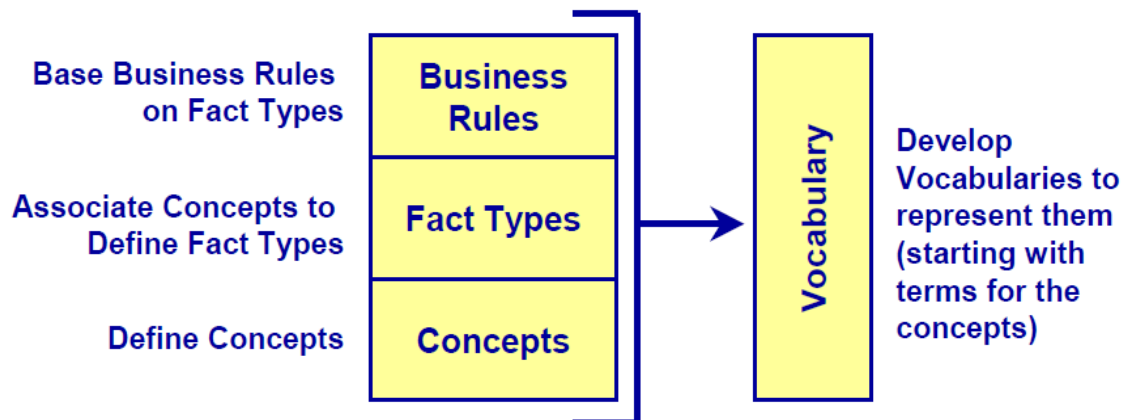
In the following *figure 2.5.1*, it is depicted how vocabulary and rules overlay for the creation of a simple SBVR model. According to [Hall, J., 2006]: SBVR realizes the



business rules ‘Mantra’: “Rules are built on facts. Facts are built on terms.” presented in *figure 2.5.2*.



*Figure 2.5.1: Simple View of SBVR model [Hall, J., 2006]*



*Figure 2.5.2: Business Vocabulary development [Hall, J., 2006]*

### 2.5.1 SBVR Business Vocabulary

The SBVR’s first aspect is the Business Vocabulary. The basic ingredients are the business glossary, the taxonomy, the thesaurus and the natural language ontology.

Business glossary defines noun concepts and definition as well as primary terms. The taxonomy defines the hierarchical relationships (equivalent to UML and OWL classes) and the thesaurus contains multilingual grammar terms such as synonyms, acronyms, abbreviations, etc. Also instances of concepts are defined in the taxonomy e.g. Business Events and Business Entities and verb concepts meaning Business facts and Relations among concepts. Finally, in the ontology, relations among instances of concepts are described, structural business rules as well as definitions, relationships and rules specified in formal logic (equivalent to UML and OWL relationships [Hall, J., 2006]. Noun concepts form class hierarchies via subtype relationships, providing this way the basis for submission reasoning [Linehan, M. H., 2009].

Furthermore, as proposed by [Linehan, M. H., 2009], SBVR vocabularies are “reference ontologies” as their primary intended use of is descriptive and rather than execution. Despite that, SBVR vocabularies can be employed as implementation structures, given the following example:

customer

Definition: one that purchases a commodity or service

Source: Merriam Webster Collegiate Dictionary

customer submits order to company

Definition: **the** customer *transmits* **the** order to **the**company

firm order

Definition: order **that** *is final*

letter of intent

Definition: order **that** *is not final*

Synonym: LOI

Note: A customer may submit an LOI to get a price quote, delivery schedule, or other terms of sale.

order

Definition: customer request for items

order has item

Necessity: **Each** order *has* **at least** one item.

SBVR Operative Business Rules the second aspect of SBVR, the model rules, are based on predicate logic formulas. SBVR provides a means for describing the structure of the meaning of rules expressed in the natural language that business people use. In SBVR this is called “semantic formulation”. Semantic formulations are formulas that make up meaning utilizing the given vocabulary and not statements.

Rules governing the business actions are distinguished from most other rule systems by the use of alethic and deontic modalities from the world of philosophy and logic (Halpin, 2006). Alethic modalities allow structural formulation, e.g. “**It is possible that a car has more than one driver**” while deontic modalities describe behavior, e.g. “**Every rental must be paid in advance**”. Also, SBVR supports exception rules, which serve in authorization or access rules creation, e.g. “**The gate may be opened only for authorized users.**”

Although business rules are expressed in natural language, some rules are often depicted graphically.

Tools such as the ACE Reasoner (RACE) for rule parsing for consistency checking, textual entailment or query answering do not exist. Therefore many user-defined vocabularies and rule combinations lead to impractical and incoherent computations. As predicate logic is the basis for SBVR rules, rule validation and rule conflicts development of such tools is feasible. Furthermore, formal semantics of SBVR allow the rule simulation.



## 3 Addressing the Problem

In this chapter, the idea that initiated this master thesis is explained. The problem statement is analyzed in section 3.1 in more detail. In section 3.2, the implementation choices of the programming language of this project, the knowledge representation and transition languages are justified. Furthermore an introduction into Rules and RuleML is given into section 3.3 and in section 3.4 the Rule Responder System is presented. Lastly, part of the problem definition is the understanding of similar applications, presented in section 3.5, and how the solution that will be developed needs to differentiate from them.

### 3.1 The Problem

The problem that is addressed in this dissertation project consists of creating a simple and easy to use interface for non-expert users to access the semantic web. In the Semantic Web this is the task of External Semantic Web Agents. A semantic web external agent has to provide the ability to access the semantic web in a convenient way in order to provide a low entry level for non-experts. A convenient way can be defined as the way users are already accustomed to search the Web, like key-word searching in search engines.

Natural languages, offer such a possibility to pose structured sentence-like queries that can be processed by machines. Likewise answers can be returned to the users in an easy to process and comprehensive way.

The RuleML symposium has implemented the Rule Responder Semantic Web Agents System that assist in organizing and retrieving information about the symposium chairs of the RuleML symposia, with an External Web Agent called *Symposium Planner*, currently *SymposiumPlanner-2012*. Although this External Web Agent is accessible by anyone who wants to get informed about the symposium, it requires the knowledge of a markup language and most specifically Reaction RuleML. As an alternative the devel-

opers of the system offer the possibility to the users to select predefined queries and also select the organizational or sub-organizational agent the user wants to address to. Although this eliminates the previous restriction it creates a new one, that of a finite number of possible queries.

The RuleML Symposium Planner External Web Agent can be accessed online in the RuleML symposium webpage<sup>1</sup>.

Furthermore, as H. Boley [Boley, H. 2010], one of the main contributors to the Rule Responder System mentions in one of his presentations about the system, the communication improvement between Personal Agents and Humans is addressed as future work for the Rule Responder System. To this end, he mentions the semi-formal interaction with the use of Controlled English in posing questions and receiving answers. Similar extension can apply to other known implementations such as the ones that will be presented in section 3.7.

Also H.Boley addresses the issue of Query Decomposition and proposes the queries being decomposed into sub-queries, delegated to multiple PAs, and partial answers re-integrated and secondly the application in information with data and knowledge integration.

In order to offer a simple and easy to use interface for the Rule Responder Organizational Agent and meet the new requirements of the Rule Responder System the *ACE Interface External Web Agent* is created and thoroughly described in Chapter 4.

## 3.2 Implementation Choices

### 3.2.1 Programming Language Selection

As programming language for the ACE Interface External Web Agent, Java was selected. The choice was encouraged by the fact that the Rule Responder System is implemented in Java and Prova (a combination of Java and Prolog) and the selection of the same language for this project can also assist to future improvement and integration to other applications by any of the involved parties.

---

<sup>1</sup> <http://dbis.informatik.tu-cottbus.de/ruleml2012/ruleresponder.html>.

### 3.2.2 Selection of the Input and Communication Languages

Among the decisions that have to be taken in order to approximate a feasible solution to the implementation of the project, is the selection firstly of the controlled natural language that is going to be used in the front-end of the Interface and secondly the knowledge modeling language used for the agent-to-agent communication. The two languages, which are going to be used for these purposes, are ACE and Reaction RuleML.

ACE as thoroughly discussed earlier in Chapter 2, is a precisely defined subset of English specifically designed for requirements specifications and knowledge representation. As it can be seen in the examples, with a minimum practice one can easily use the language to its full potential, along with the use of the provided tools. Additionally, users can extend its use to private knowledge domains, by defining their own vocabularies and with the use of special annotations.

The XML-based Rule Markup Language (RuleML) is used for representing and exchanging rules over the web that are used for derivation, query, transformation, integrity checking and reactive behavior. The Rule Responder system is using Reaction RuleML as knowledge interchange format. Reaction RuleML allows the expression of rules, via queries, towards the coordinating personal agents that assist to the retrieval of information about the symposium chairs of the RuleML symposia. The responses obtained from the Rule Responder system are in Reaction RuleML as well.

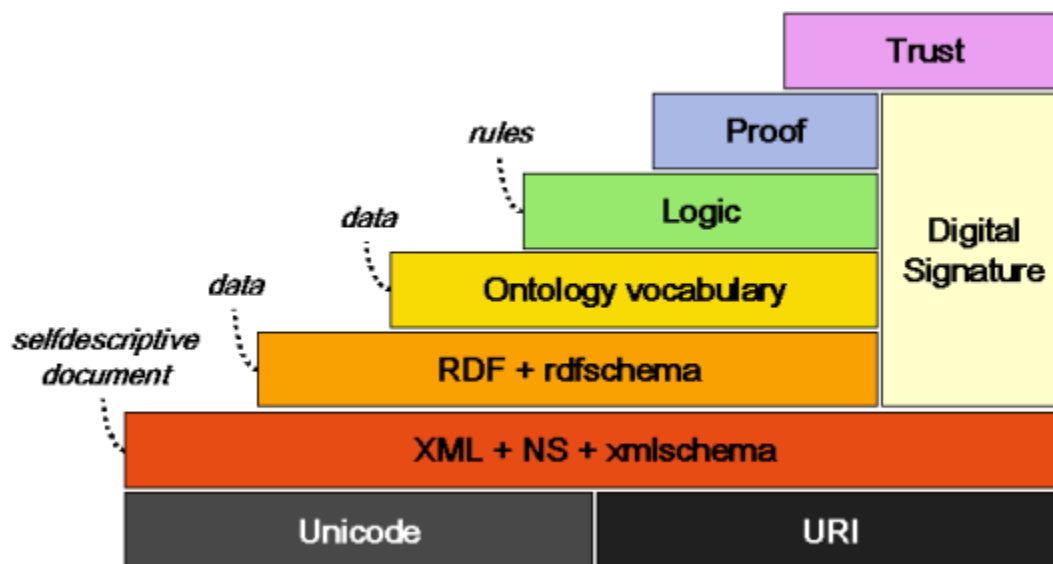
Reaction RuleML covers constructs for (complex) events, actions and states/ fluent/ transition (changeable properties and transitions) definition and processing/ reasoning for different derivation rules, production rules and reaction rules programs.

## 3.3 Semantic Web Rules and RuleML

As mentioned in previous sections rules form a component of vast importance for knowledge representation and management systems and also serve as the basis for the design of the Semantic Web. The Semantic Web aims at improving the current Web, by augmenting its content with semantics and encouraging the cooperation among human users and machines. Since the basic Semantic Web infrastructure is reaching sufficient

maturity, research efforts are shifting towards logic, proof and trust and rule-based systems inevitably concentrate most of the attention. Nevertheless, in order for human users to trust system answers, they have to be presented with adequate explanations that justify the derived results [Kontopoulos E. et. al., 2008]. And, even more importantly, these explanations have to be presented in a user-comprehensible format.

The Semantic Web principles are implemented in the layers of Web technologies and standards. The layers are presented in figure 3.3.1. The Unicode and URI layers ensure that the characters sets used are international and provide common means for identifying the objects in Semantic Web. The XML layer with namespace and schema definitions ensure that XML based standards apply in the Semantic Web. With RDF [RDF] and RDFSchema [RDFS] it is possible to make statements about objects with URI's and define vocabularies that can be referred to by URI's. This is the layer where types to resources and links are applied. The Ontology layer supports the evolution of vocabularies as it can define relations between the different concepts. With the Digital Signature layer for detecting alterations to documents, these are the layers that are currently being standardized in W3C working groups.



**Figure 3.3.1: The Semantic Web layers [Koivunen M.R., Miller E., 2001]**

The top layers: Logic, Proof and Trust, are the topic of all modern researches and simple application demonstrations are being constructed. The Logic layer enables the writing of rules while the Proof layer executes the rules and evaluates together with the



Trust layer mechanism for applications whether to trust the given proof or not [Kivinen M.R., Miller E., 2001].

Logic is currently the target of the majority of the upcoming efforts towards the realization of the Semantic Web vision, namely making the content of the Web accessible not only to humans, as it is today, but to machines as well. The Rule Markup Language (RuleML) is the language enabling the derivation and transmission of rules.

### 3.3.1 RuleML

*RuleML (Rule Markup Language)* is a family of sublanguages implemented via modular XML schemas spanning across all industrially relevant kinds of Web rules. This type of implementation employs the modularity software engineering principle, resulting in increased maintainability (inheritance of rules) and interoperability, accommodating rule sub-communities who are able to specify the best serving language from the family [Hirtle, D. et al., (n.d.)]. Rules in RuleML are written as a combination of natural language and formal notation (a semiformal XML-based markup language). This language permits the Web-based rule modeling, markup, translation, archiving, interchange, execution and publication in XML or even UML, RDF and ASCII for platform independence and interoperability.

Markup standards and initiatives related to RuleML include:

- Mathematical Markup Language (MathML)
- DARPA Agent Markup Language (DAML)
- Predictive Model Markup Language (PMML)
- Attribute Grammars in XML (AG-markup)
- Extensible Stylesheet Language Transformations (XSLT) [RuleML Symposium, 2010].

RuleML is the product of the RuleML Initiative<sup>2</sup>, an international non-profit organization consisting of an open network of individuals and groups from both industry and

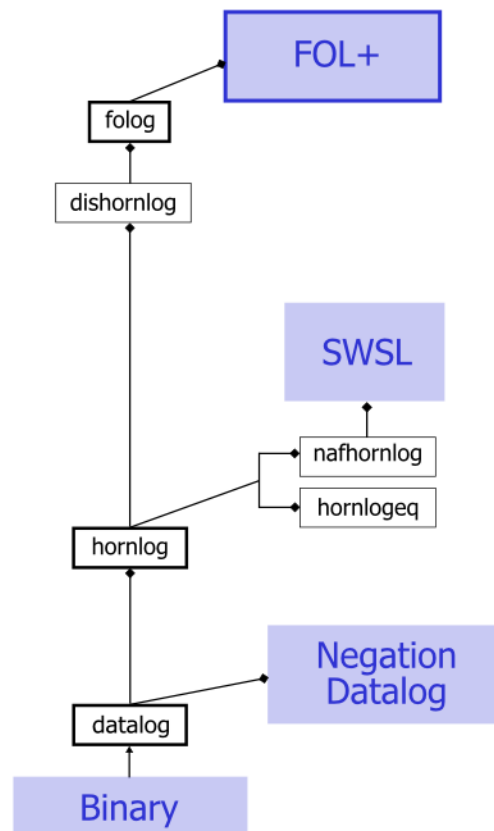
---

<sup>2</sup> [www.ruleml.org](http://www.ruleml.org)

academia that share the interest of the interoperation of Semantic Web rules. RuleML specification, tool, and application development are the tasks of the technical groups of the Initiative. The Initiative has been collaborating with OASIS on Legal XML, Policy RuleML, LegalRuleML, and related efforts since 2004. Further on, the initiative has been interacting with the developers of ISO Common Logic (CL), which became an International Standard in October 2007. RuleML is also a member of OMG, contributing to its Semantics of Business Vocabulary and Business Rules (SBVR), and to its Production Rule Representation (PRR). Moreover, participants of the RuleML Initiative have been supporting the development of the W3C Rule Interchange Format (RIF) [RuleML Symposium, 2010].

### 3.3.2 The RuleML Family of Sublanguages

Each sublanguage, represented as a rectangle in the following figure (3.3.2), corresponds to a well-established rule system (e.g. datalog and hornlog), allowing users to pick and choose to a sublanguage, according to their respective.

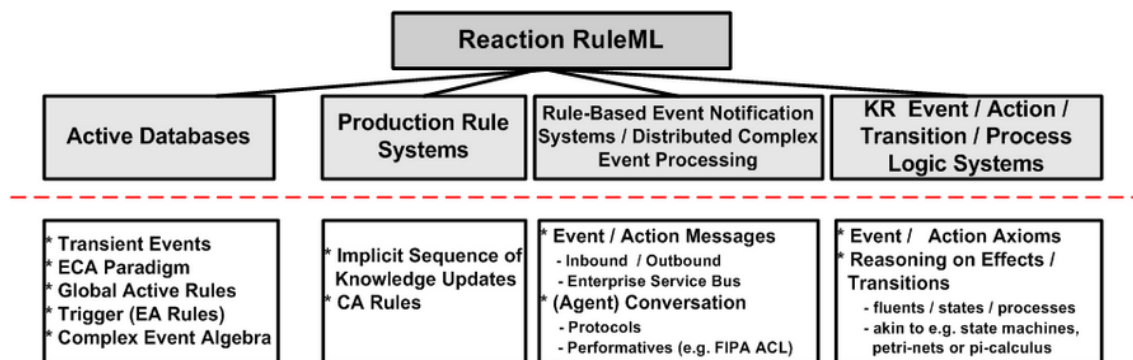


*Figure 3.3.2: The modularization of RuleML [Hirtle, D. et al., (n.d.)]*

The most expressive "class" (i.e. sublanguage) is shown at the top and generality decreases in top-down order. As in the UML, a diamond-headed arrow indicates an aggregation association (e.g. "datalog *is part of* hornlog" and "cterm *is part of* hornlog") while regular-headed arrows indicate generalization as used for inheritance (e.g. "bindatalog *is a* datalog"). Certain aggregation associations, such as hornlog to nafhornlog and hornlog<sub>eq</sub>, branch and have multiple (in the figure 3.3.1, two) targets. This notation logically places all target nodes on the same (horizontal) level. The ovals in the above model represent elementary modules which act as "private" constituents of the actual sublanguages (which are represented as rectangles). This composition may happen directly, as is most obvious for datalog, or indirectly through subsequent associations. According to these conventions, elementary modules cannot be associated with one another because they are dependent on rectangles [Hirtle, D. et al., (n.d.)].

### 3.3.3 Reaction RuleML

Reaction RuleML belongs to the RuleML family of languages and is a general, practical and user friendly language. It is based on XML, which is the primary means for rules interchange. It embodies different kinds of production, action and reaction rules, composite facts and timely logic rules in the RuleML syntax using a system of step-by-step extensions. Most significantly a variety of reaction rules serving in various domains are covered. Reaction Rules can be defined in combination with other sets of rules like conclusion derivation rules or integrity constraints or even inbound in other rule sets. *Figure 3.3.3* represents schematically the main application domains for Reaction RuleML.

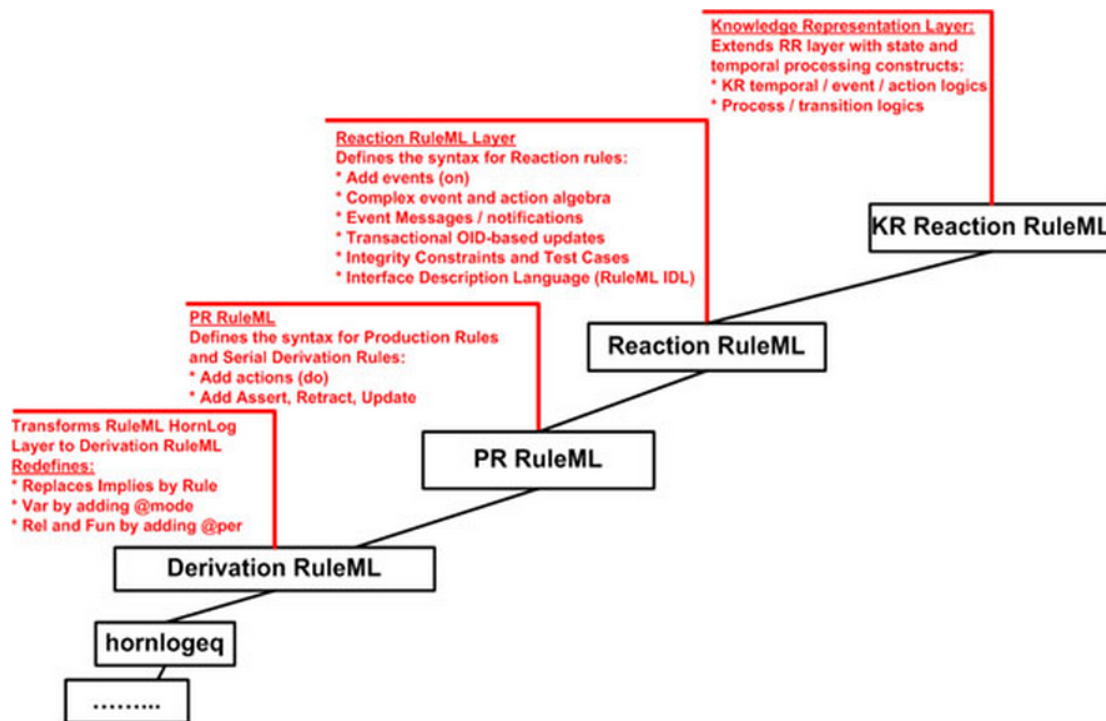


**Figure 3.3.3: Reaction RuleML applications [Paschke, A. et al., 2007a]**

Reaction RuleML covers constructs for (complex) events, actions and states/ fluent/ transition (changeable properties and transitions) definition and processing/ reasoning for different derivation rules, production rules and reaction rules programs.

### 3.3.3.1 Modularization of Reaction RuleML

Reaction RuleML follows the RuleML design patterns and defines new types in single modules which are added in the RuleML family as layers on top of RuleML's hornlog layer. In Version 0.2 the syntax of rules is generalized to a *core rule format* which can be specialized in different language families to different rule types such as *derivation rules*, *production rules*, and *reaction rules* as well as mixed formats. [Paschke, A. et al., 2007a]. Every layer adds different modeling expressiveness to the Reaction RuleML core for the representation of behavioral (re)action and KR event/action logic (figure 3.3.4).



**Figure 3.3.4: Reaction RuleML sublanguages [Paschke, A. et al., 2007a]**

The language fulfils the criteria for good language design as defined by McCarthy and Hayes [McCarthy, J., Hayes, P. J., 1969] such as *minimality*, *symmetry*, *orthogonality* and *adequacy*.

- *Minimality* declares that in Reaction RuleML only a restricted set of language constructs is provided in addition to the existing RuleML constructs.
- *Symmetry* means that the same language constructs always express the same semantics regardless of the context they are used in.
- *Orthogonality* permits every meaningful combination of a language constructs to be applicable.
- *Adequacy* means that the user has adequate resources to express his knowledge over Reaction RuleML.

### 3.3.3.2 Reaction RuleML Production Rules

Production rules are a part of Reaction RuleML and contain actions of update (Assert, Retract) and actions (do). For the production rules definition and example the main sources used are [Paschke, A. et al., 2007b ; Paschke, A. (n.d.)a].

Production rules are defined in an XML schema file and in this file the following elements and characteristics of Reaction RuleML are defined:

- The Rule is redefined with the addition of actions using the words ‘do’ and ‘elseDo’ and meta-conditions using the words ‘after’ and ‘elseAfter’.
- Elementary update actions are defined (Assert, Retract).
- Action ‘Assert’ gets redefined adding the characteristics ‘@safety’ and ‘@all’.
- Action ‘Retract’ gets redefined adding the characteristics ‘@safety’ and ‘@all’.

Additionally, the actions ‘Assert’ and ‘Retract’ can be used in the part of action ‘do’ or in the form of a series of Horn rules in the if-part of a production rule.

In the following PR schema the Rule element is redefined and derivation and production of mixed production rules are added.

```
( oid?, label?, scope?, quantification?,
  (
    (if, ( (do, after?, else?, elseDo?, elseAfter?) |
      (then, else?, elseDo?, elseAfter?) ) )
  )
)
```

)

Rules are redefined so that production rules are in the form: (if, do, after?, else?, elseDo?, elseAfter?).

```
<xs:group name="ProductionRule.content">
  <xs:sequence>
    <xs:group ref="ProductionRule.content"/>
    <xs:element ref="do"/>
    <xs:element ref="after" minOccurs="0"/>
    <xs:element ref="else" minOccurs="0"/>
    <xs:element ref="elseDo" minOccurs="0"/>
    <xs:element ref="elseAfter" minOccurs="0"/>
  </xs:sequence>
</xs:group>
```

Actions ‘elseDo’ and ‘elseAfter’ are added so that derivation rules are in the form: (if, then, else?, elseDo?, elseAfter?).

```
<xs:group name="DerivationRule.content">
  <xs:sequence>
    <xs:group ref="DerivationRule.content"/>
    <xs:element ref="elseDo" minOccurs="0"/>
    <xs:element ref="elseAfter" minOccurs="0"/>
  </xs:sequence>
</xs:group>
```

Additionally, in the PR schema the element ‘Assert’ is redefined and the characteristics ‘@all’ and ‘@safety’ are added for the interchange updates:

```
@safety = normal | transactional; default = normal
@all = yes | no; default = no
```

(oid?, (formula | Rulebase | Atom | Rule | Equivalent | Entails | Forall | Equal | Neg)\*)

```
<xs:attributeGroup name="Assert.attlist">
  <xs:attributeGroup ref="Assert.attlist"/>
  <xs:attributeGroup ref="safety.attrib"/>
  <xs:attributeGroup ref="all.attrib"/>
</xs:attributeGroup>
```

Moreover, in the PR schema the element ‘Retract’ is redefined and the characteristics ‘@all’ and ‘@safety’ are added for the interchange updates:

```
<xs:attributeGroup name="Retract.attlist">
  <xs:attributeGroup ref="Retract.attlist"/>
  <xs:attributeGroup ref="safety.attrib"/>
  <xs:attributeGroup ref="all.attrib"/>
</xs:attributeGroup>
```

The actions ‘Assert’, ‘Retract’, ‘Update’ are added in the body of PR so that it goes to the form: (Atom | And | Or | Equal | Naf | Neg | Assert | Retract | Update)

```
<xs:group name="body.content">
  <xs:choice>
    <xs:group ref="body.content"/>
  </xs:choice>
  <xs:group ref="update_primitives.content" />
</xs:group>
```

The actions ‘Assert’, ‘Retract’, ‘Update’ are added so that the content type of the model is: (Atom | Naf | Neg | And | Or | Equal | Assert | Retract | Update)

```
<xs:group name="formula-and-or.content">
  <xs:choice>
    <xs:group ref="formula-and-or.content"/>
    <xs:group ref="update_primitives.content" />
  </xs:choice>
</xs:group>
```

### 3.3.3.2.1 A derivation rule example

In the following example the rule:

```
(occurs (heartbeat ?Service), ?T) => (assert ( alive ?Service, ?T))
```

is implemented:

Variable begin with “?”. The <Assert> label is used to import actions and conditions in the database. (style="active") is the first assertion rule and the condition marked by the <Atom > label are included in the <if> part. The sentence includes the predicate <Rel> occurs, an expression <Expr>, containing a function <Fun> heartbeat. With a variable <Var> Service and the time period the variable is in (<Var>). All this is translated as: A

‘Service’ in a certain moment ‘T’ has ‘heartbeats’. The actions as mentioned in the previous section are included within the <do> labels. In this rule there is only one action and this is declared within the second <Assert> labels. It declares that the logical assumption written within the <Atom> labels will be included if the condition is true. The <oid > declared the action identifier <Ind>availability. The logical assumption include the predicate <Rel> alive and the variables <Var> Service & T. The action called ‘availability’ inserts to the system knowledge base the logical sentence that a ‘service’ is ‘alive’ at ‘T’.

```

<Assert>
  <Rule style="active">
    <if>
      <Atom>
        <Rel use="value">occurs</Rel>
        <Expr>
          <Fun use="value">heartbeat</Fun>
          <Var>Service</Var>
        </Expr>
        <Var>T</Var>
      </Atom>
    </if>
    <do>
      <Assert>
        <oid><Ind>availability</Ind></oid>
        <Atom>
          <Rel>alive</Rel>
          <Var>Service</Var>
          <Var>T</Var>
        </Atom>
      </Assert>
    </do>
  </Rule>
</Assert>

```



### 3.4 The Rule Responder System

Since Rule Responder is the basic infrastructure that the ACE Interface External Web Agent will collaborate with, a brief description over its architecture and its knowledge interchange format will be given. No further implementation is done over the Rule Responder apart from the connection with the system.

Rule Responder is a multi-agent system for collaborative team and community support on the (Semantic) Web that enables the rule-based collaboration between the distributed human members of such a virtual organization [Paschke, A., Boley H., 2011]. It is implemented and deployed for RuleML symposium since 2008. Persons of an organization are assisted by semi-automated rule-based agents, which use rules, to describe the decision and behavioral logic. The languages of the RuleML family used for rule serialization based on logic and XML are:

- Hornlog RuleML: Reasoning (decision)
- Reaction RuleML: Interaction (behavior)

The Rule Responder is implemented on top of a Mule-based Enterprise Service Bus (ESB) as a Service Oriented Architecture (SOA). *Mule*<sup>3</sup> is used to create communication end points at each personal and organizational agent of Rule Responder. The transport protocols used are Hyper Text Transfer Protocol (HTTP) and Java Message Service (JMS).

The semi-automated agents are distinguished to Personal, Organizational and External Agents. A personal agent (PA) assists a person – or selected group of persons – of an organization, semi-autonomously acting on their behalf. A personal agent works on a profile of FOAF-like (Friend of a Friend) facts plus FOAF-extending rules that encode ‘routine’ knowledge of its human owner(s). Each personal agent is realized in the Rule Responder multi-agent infrastructure with a servlet using one or more rule engine(s).

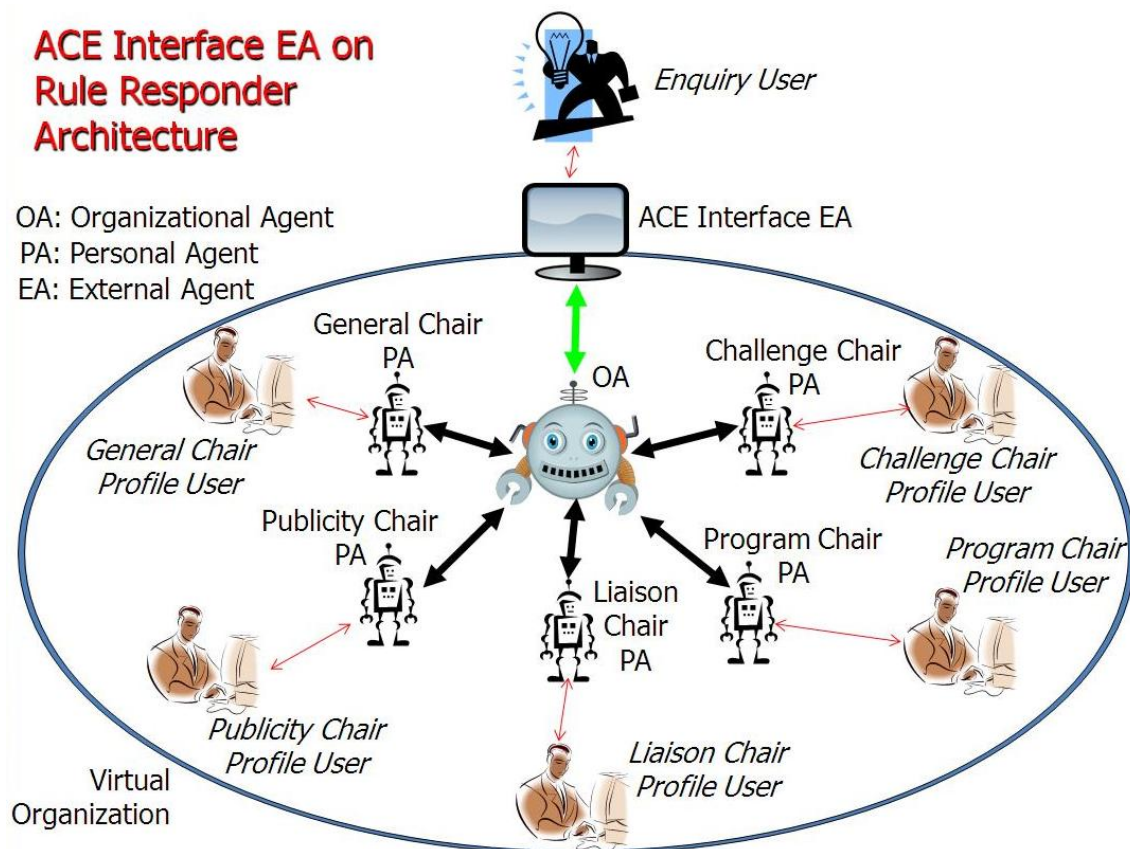
An organizational agent represents goals and strategies shared by each member of the organization. It contains rule sets (that consist of facts) that describe the policies, regulations, opportunities, and expertise of its organization. Every organizational agent is realized with an instance of a rule engine.

---

<sup>3</sup> <http://www.mulesoft.org/>

External agents exchange messages with (the public interface of) organizational agents, sending queries (requests), receiving answers (results), or interchanging complete rule sets. End users, via external agents, employ the Web (HTTP) interface of Rule Responder (currently an API-like browser interface), *figure 3.4.1*.

The rule engines used by the Rule Responder are Prova (Prolog + Java), OO jDREW: (Object Oriented java Deductive Reasoning Engine for the Web) [Boley, H., Craig B. 2008] and DR-Device (Defeasible Logic Reasoner for the Semantic Web), [Kontopoulos E., et.al., 2011]. Rules can be shared among personal agents and rules that apply to all, are lifted a level higher, to the organizational agent level.



**Figure 3.4.1: Animated representation of the External Agent interoperation with Rule Responder. [Boley, H. 2010]**

### 3.4.1 Interchange of Knowledge between Agents

Each rule engine can use its own rule language. On the other hand, agents require an interchange language so they can communicate with each other. For this matter Rule

Responder uses RuleML, most specifically Reaction RuleML as its interchange language. Translations between the interchange language and the personal agent individual languages are done by the personal agents. The Reaction RuleML messages and rule bases between the agents are sent through the ESB as requests and results. Each PA has a knowledge base containing the responsibilities of the position in order to answer queries relevant to the chair's role. An example of a Reaction RuleML message transmitted between agents is given below.

```
<?xml version="1.0" encoding="UTF-8"?>

<RuleML xmlns="http://www.ruleml.org/0.91/xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.ruleml.org/0.91/xsd"
xmlns:ruleml2012="http://ibis.in.tum.de/projects/paw#">
  <Message directive="query-sync" mode="outbound">
    <oid>
      <Ind>SymposiumPlannerSystem</Ind>
    </oid>
    <protocol>
      <Ind>esb</Ind>
    </protocol>
    <sender>
      <Ind>User</Ind>
    </sender>
    <content>
      <Atom>
        <Rel>getContact</Rel>
        <Ind>ruleml2012ATecai_GeneralChair</Ind>
        <Var>Contact</Var>
      </Atom>
    </content>
  </Message>
</RuleML>
```

Through this request message the contact information of the General Chair from the Rule Responder system Super-Organizational Agent. The response is shown below:

```
<Message mode="outbound" directive="answer">
  <oid>
    <Ind>httpEndpoint:143</Ind>
  </oid>
  <protocol>
    <Ind>esb</Ind>
```

```

</protocol>
<sender>
  <Ind>httpEndpoint</Ind>
</sender>
<content>
  <Atom>
    <Rel>getContact</Rel>
    <Ind>ruleml2012ATecai_GeneralChair</Ind>
    <Expr>
      <Fun>person</Fun>
      <Expr>
        <Fun>name</Fun>
        <Ind>Grigoris Antoniou</Ind>
      </Expr>
      <Expr>
        <Fun>email</Fun>
        <Ind>antoniouATdomainname</Ind>
      </Expr>
      <Expr>
        <Fun>phone</Fun>
        <Ind>302810391624</Ind>
      </Expr>
      <Expr>
        <Fun>title</Fun>
        <Ind>Dr</Ind>
      </Expr>
      <Expr>
        <Fun>role</Fun>
        <Ind>General Chair</Ind>
      </Expr>
      <Expr>
        <Fun>affiliation</Fun>
        <Ind>University of Huddersfield</Ind>
      </Expr>
    </Expr>
  </Atom>
</content>
</Message>

<Message mode="outbound" directive="answer">
  <oid>
    <Ind>httpEndpoint:143</Ind>
  </oid>
  <protocol>
    <Ind>esb</Ind>
  </protocol>
  <sender>
    <Ind>httpEndpoint</Ind>
  </sender>
  <content>
    <Atom>
      <Rel>getContact</Rel>
      <Ind>ruleml2012ATecai_GeneralChair</Ind>
      <Expr>
        <Fun>person</Fun>
        <Expr>
          <Fun>name</Fun>
          <Ind>Guido Governatori</Ind>
        </Expr>
        <Expr>
          <Fun>email</Fun>

```

```

        <Ind>guidoDOTgovernatoriATnictaDOTcomDOTau</Ind>
    </Expr>
    <Expr>
        <Fun>phone</Fun>
        <Ind>610733652907</Ind>
    </Expr>
    <Expr>
        <Fun>title</Fun>
        <Ind>Dr</Ind>
    </Expr>
    <Expr>
        <Fun>role</Fun>
        <Ind>General Chair</Ind>
    </Expr>
    <Expr>
        <Fun>affiliation</Fun>
        <Ind>NICTA</Ind>
    </Expr>
</Expr>
</Atom>
</content>
</Message>

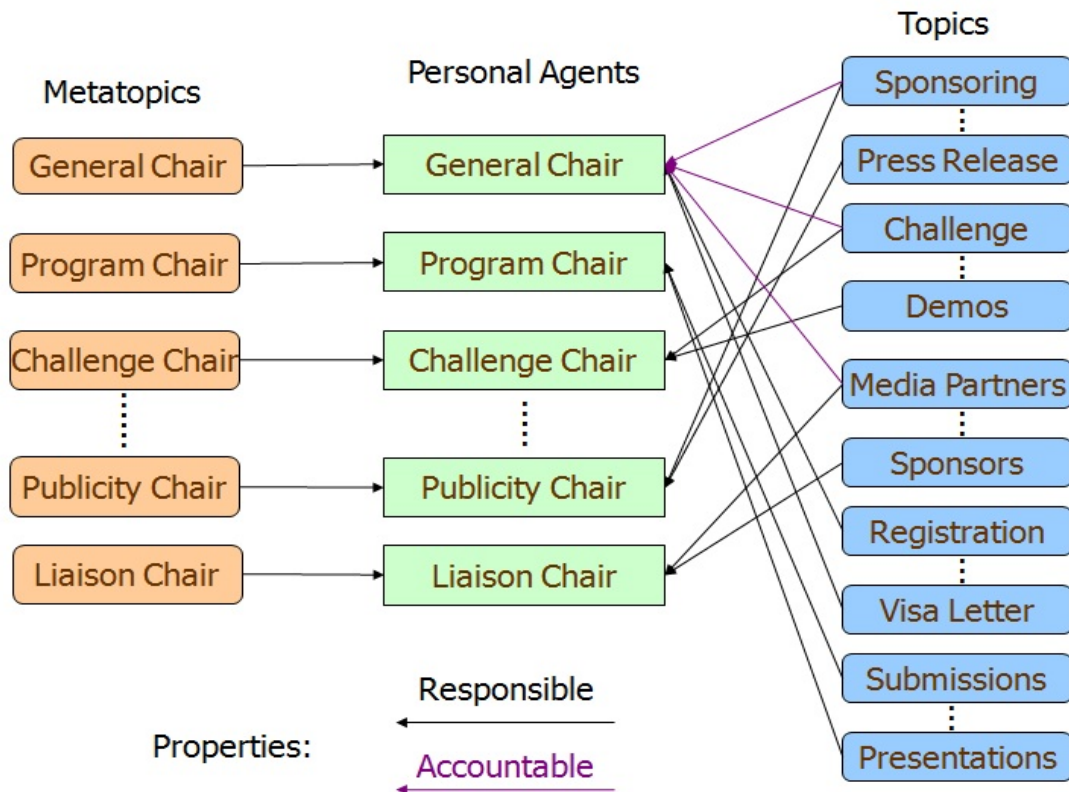
```

The attribute `directive="..."` specifies the pragmatic performance, which is either a request ("answer") or a response ("query-sync").

### 3.4.2 Query Delegation among the Agents

The requests (queries) are delegated to the appropriate personal agents by the organizational agent. A role assignment matrix is responsible for the correct task management. As seen in *figure 3.4.2*, (meta)topics are assigned to personal agents within the virtual organization.

# Role Assignment Ontology



*Figure 3.4.2: Query Delegation with the use of a role assignment matrix for the RuleML symposium [Boley, H. 2010].*

Some queries have more than one answers, as in the example given in section 4.1.1, the personal agent send the answers one at the time, interleaving backtracking and transmission increasing the overall performance of the system. When no more answers are found by the personal agent an end-of-transmission message is sent to the organizational agent and then a complete answer is displayed.

## 3.5 Related Work

The existing work most related to the “ACE Interface External Web Agent” includes the following:

### **TRANSLATOR**

*TRANSLATOR*<sup>4</sup> (*TRANSlator from Language TO Rules*). Specifically, rules written in ACE are taken as input, parsed (using the APE Web service) into a variant of first-order logic known as a Discourse Representation Structure (DRS) and then mapped into formal representation in the XML-based Rule Markup Language (RuleML). TRANSLATOR is available as a Java Web Start application and allows non-programmers to write their own rules on the Semantic Web via familiar natural language.

### **Symposium Planner 2011**

The Symposium Planner 2011<sup>5</sup>. Distributed rule agents in Symposium Planner 2011 consist of Prova Agent and OO jDREW Agent. Symposium Planner 2011 consults the knowledge not only from its knowledge repository, but also from Semantic Web Dog Food Corpus found online<sup>6</sup>.

### **DRACE**

A prototype tool called DRACE is already available for translating a DRS into ACE (subject to some limitations) [Fuchs N.E, et.al. 2005b]. If this tool were accessible (e.g., as a Web service), any application using as knowledge representation language one of the RuleML family of languages, could be extended to be bidirectional, supporting translation from RuleML back to ACE by reversing the DRS-RuleML mapping.

Unfortunately, during the progress of the current thesis, it was not possible to have adequate access to the above tools. The reasons for this lie on unavailability or inconsistent behaviors and functionality of the corresponding services. Therefore, the reports, as presented in this chapter, rely solely on the available documentation of the tools and no practical assessment could be made.

---

<sup>4</sup> <http://ruleml.org/translator/>

<sup>5</sup>

<http://198.164.40.210:8080/ACE2ReactionRuleML/index.jsp?agent=SymposiumPlannerSystem>

<sup>6</sup> <http://data.semanticweb.org/>





## 4 Contribution

In this chapter a detailed presentation is given over the implementation of the “ACE Interface External Web Agent” for the RuleML Rule Responder created in the terms of the project as a solution to the problem statement of the dissertation (Chapter 3). In the following sections the architecture, the user interface and the basic components are analyzed.

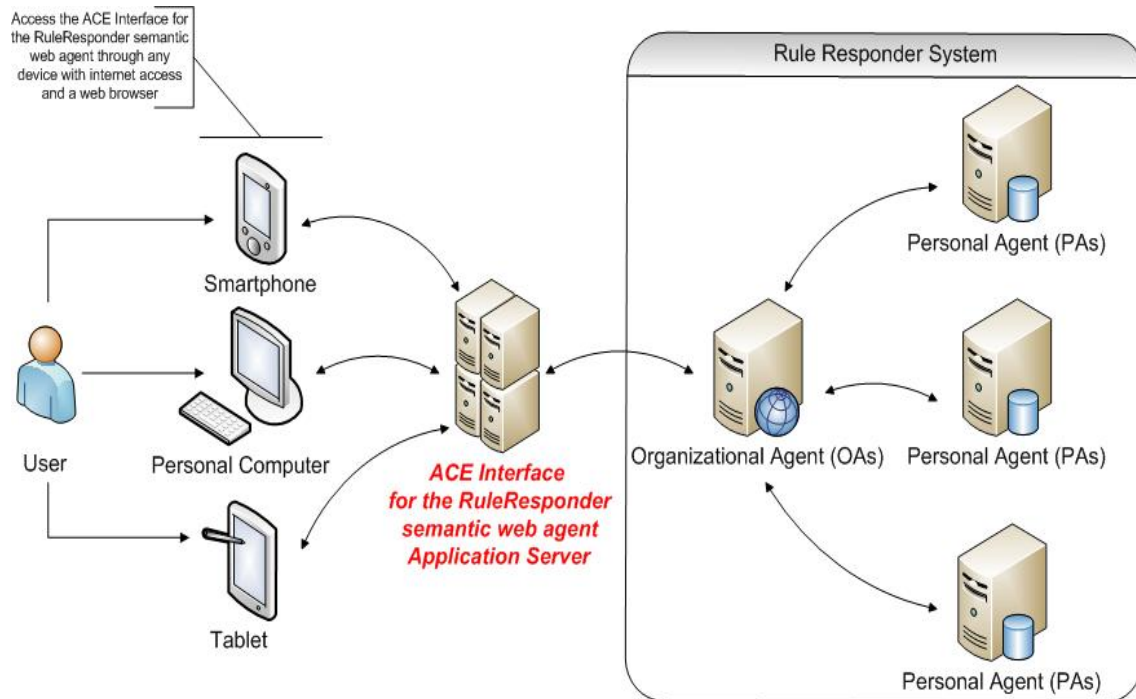
Starting with the basics about the project, a brief overview and the system architecture along with the implementation schemas are presented in sections 4.1, 4.2 and 4.3. The interface which is the user interaction medium, the web pages and their functionality are analyzed in section 4.4. Over at section 4.5 the basic steps of ACE to Reaction RuleML translation are presented. At section 4.6 fragments of the code are presented and explained in order to help the reader of this thesis understand how the project is implemented. Finally, in section 4.7 a short manual is provided in order to install the agent locally on a personal computer and the minimum system requirements.

### 4.1 The ACE interface External Web Agent

The developed ACE Interface External Web Agent is an open source semi-autonomous web platform serving as an intermediate with the RuleML *Rule Responder Organizational Agent* that allows anyone to access it and write queries in natural language without any knowledge of Reaction RuleML employed by the Organizational Agent. This is accomplished by taking input sentences expressed in annotated ACE (having the appearance of plain English but is in fact a formal language) and translating it into Reaction RuleML (a structured, restricted subset of RuleML). The aim is that this user-friendly front-end will *offer an alternative* to the RuleML SymposiumPlanner-2012 and additionally in future extensions will help lower the barrier to enter the Semantic Web and encourage non-experts to get involved.

## 4.2 System Architecture

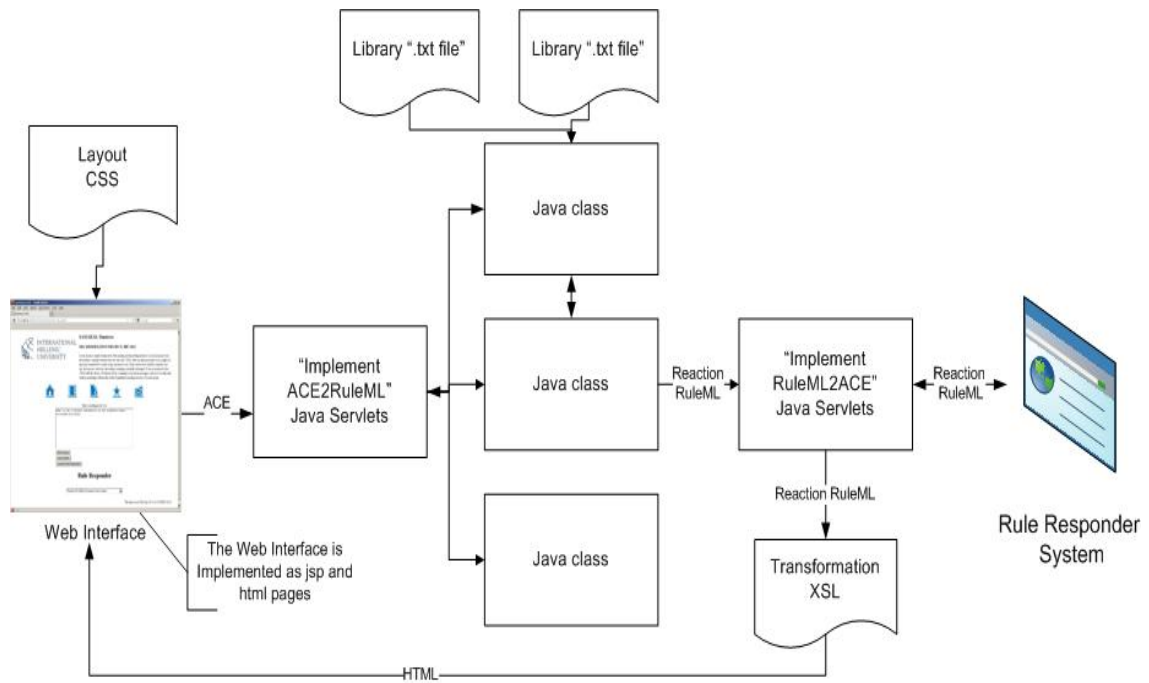
The “ACE Interface External Web Agent” is an intermediate interface between the non-experienced user and the RuleML symposium’s Rule Responder Organizational Agent (*figure 4.2.1*). The system is tightly coupled with Reaction RuleML and assists in retrieving information about the chairs of the symposium



*Figure 4.2.1: Schematic representation of the systems architecture.*

## 4.3 Implementation

The Interface is implemented, as it can be seen in *figure 4.3.1*, as a web service and its source code is in *Java*. Unlike the known competition implementations that use the Attempto project tools, the Prolog engine (Prova) is not used. The web pages for accepting the input, calling the functions and displaying the output are written in *JSP* amplified with *Javascript* code for better functionality and timely responses. Furthermore, the interoperation of the web pages and the source code is implemented by *Java servlets*.



**Figure 4.3.1: Schematic representation of the “ACE Interface External Web Agent” Implementation.**

ACE text is accepted through the User Interface and transmitted to the Java Servlet that parses and translates the input to Reaction RuleML. Java classes add to the functionality of the project as object oriented design patterns are employed, for example the text parsing and analysis methods that will be presented later on in section 4.6. As it is schematically depicted ACE with the assistance of the library files is formatted to Reaction RuleML in order to be sent to the Rule Responder System. The response is formatted in the same knowledge representation language which is parsed again and with the use of XSLT is displayed to the user as HTML. In order to offer a user friendly interface that will ease the navigation of the user in the website all the web pages share a common CSS layout.

## 4.4 User Interface

The two primary design goals of ACE Interface External Web Agent were that it be

- Widely accessible, and
- User friendly.

Both goals mentioned above are satisfied by its availability as the agent can easily be accessed (cross-platform) online via its website and provides users with an easy to use graphical user interface depicted in *figure 4.4.1*.

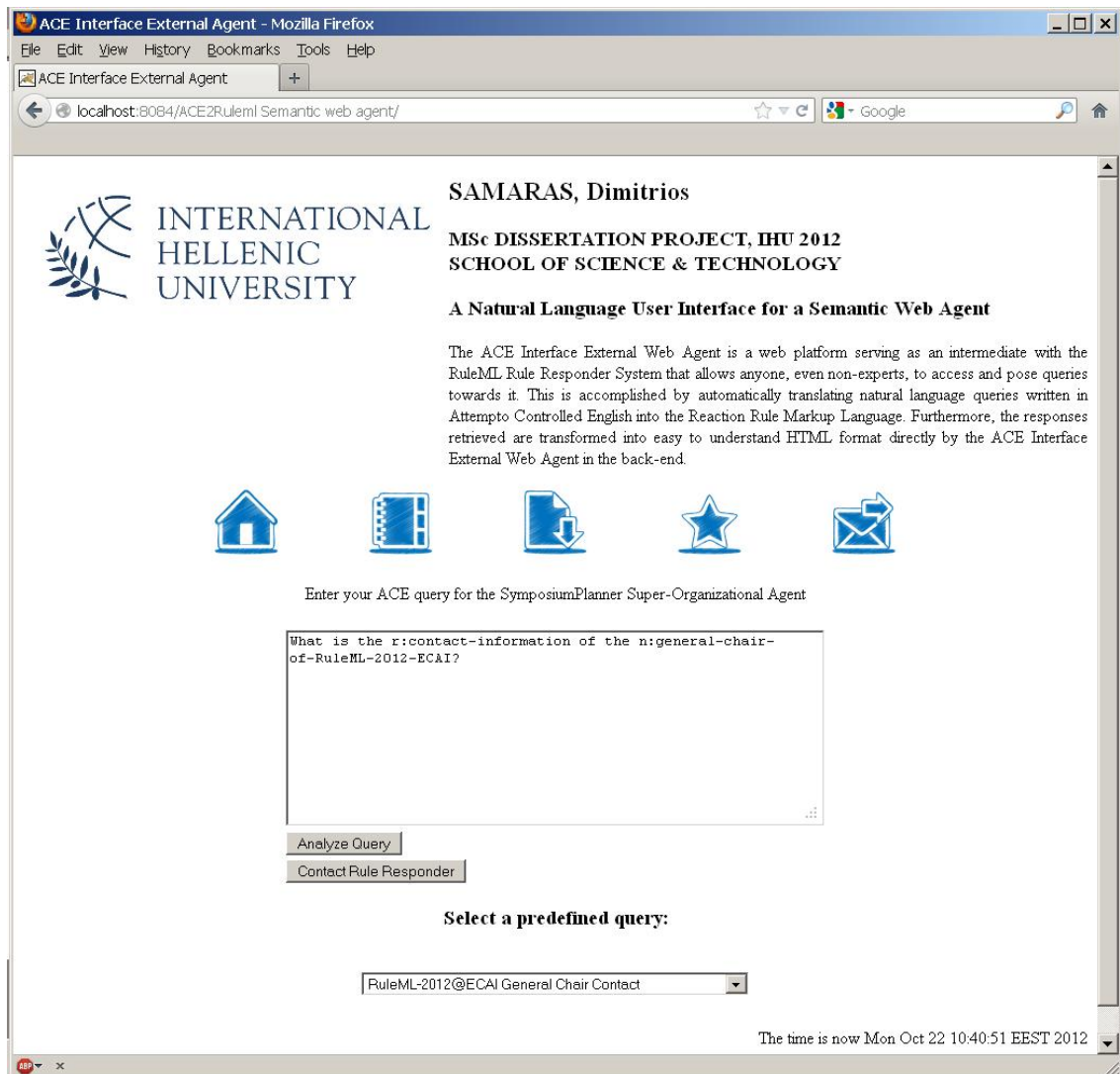
The web site comprises of 3 web pages, a link for downloading the paper of the project and a link for contacting the author<sup>7</sup>. For the web interface a light-weight and simplistic approach was taken so that the focus of the user is not distracted from the main purpose of the web site and at the same time it is easy to navigate in the rest of the web site's pages.

The first page (initial page), *figure 4.4.1*, consists of a text area, a dropdown menu, and action buttons. In the text area the user can insert the ACE query on its own or select a predefined query through the dropdown menu.

The input text accepted in the text area or the one produced through the predefined options uses special annotations for denoting nouns and predicates in the ACE text. This method of writing ACE is applied because unlike other applications for translating ACE to RuleML or other structured languages and formats, the APE parser, provided by the Attempto project, is not used and instead it is bypassed with custom dictionaries. Also ACE input is directly translated to RuleML through the Java source code and not through the DRS outcome of the APE parser. Bypassing the APE parser and using custom dictionaries also gives to the ACE Interface External Web Agent the advantage of even looser interpretation of the query, meaning that as long as the keywords necessary for the query to be interpreted are correct the response will be correct despite any consistency flaws.

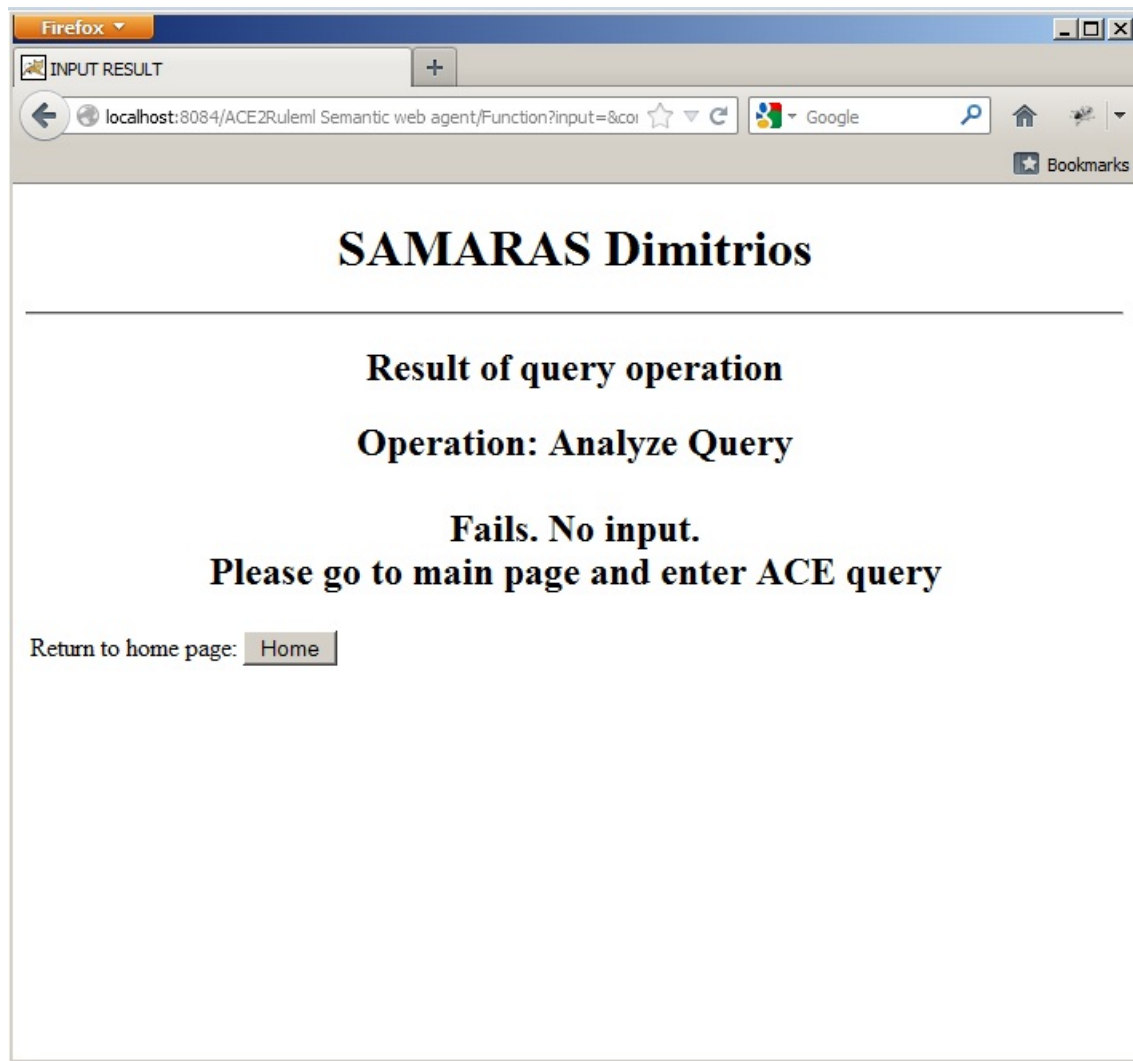
---

<sup>7</sup> Samaras Dimitrios , dimitris.samaras1@gmail.com



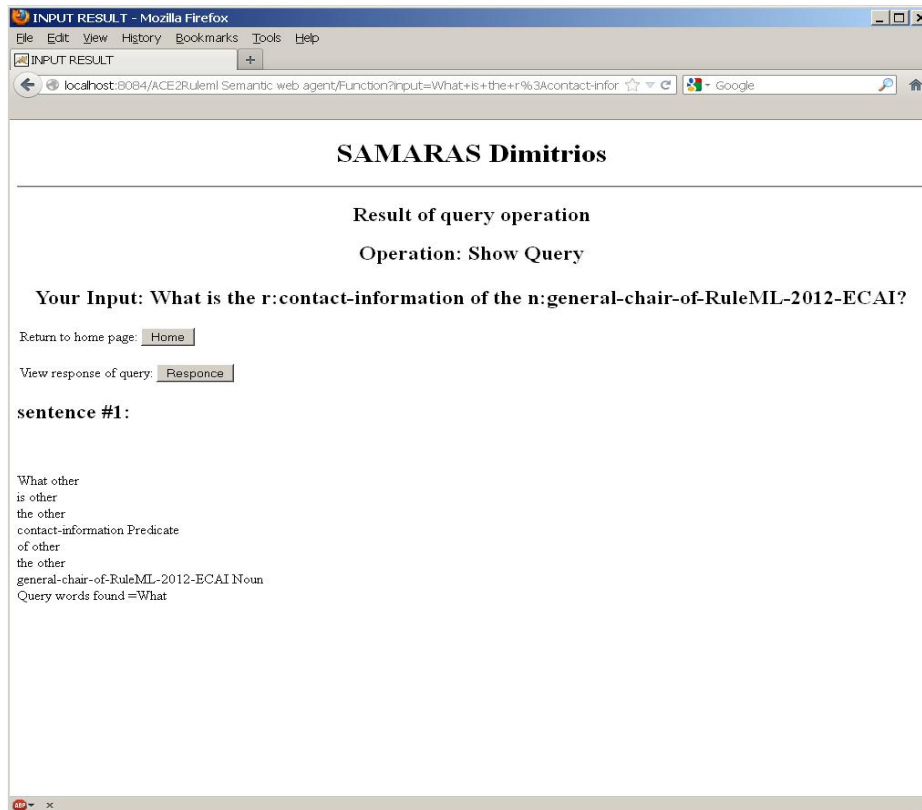
**Figure 4.4.1: Semantic Web Agent front-end, Main page**

After the query is inserted in the text area, either manually or through selecting a predefined query, there are two action buttons. The first: “Analyze Query” performs the analysis of the query and provides the result to the user. In case of malformed or no input an error message is displayed “Fails. No input. Please go to main page and enter ACE query”, *figure 4.4.2*.

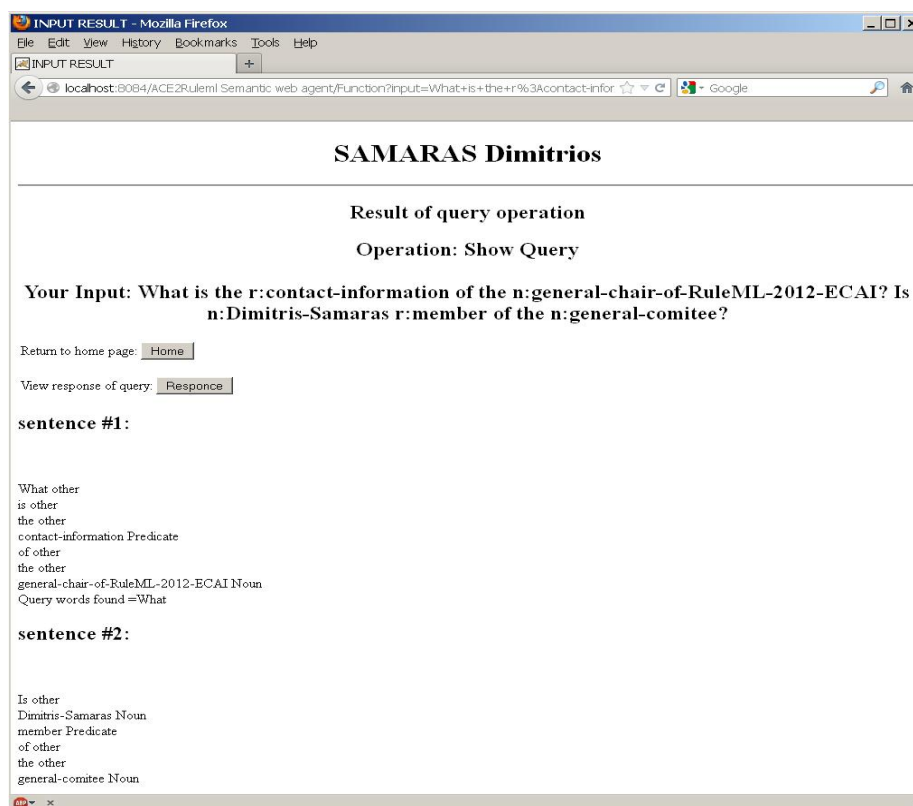


*Figure 4.4.2: Malformed or no input response*

If the input text is valid, it is split to words (tokens) and analyzed as to what is the grammatical context of each word in the sentence according to the dictionaries and what function words (words denoting a query) exist in the sentence, *figure 4.4.3*. Furthermore the ACE Interface External Web Agent is capable of analyzing more than one sentence at the time, *figure 4.4.4*.

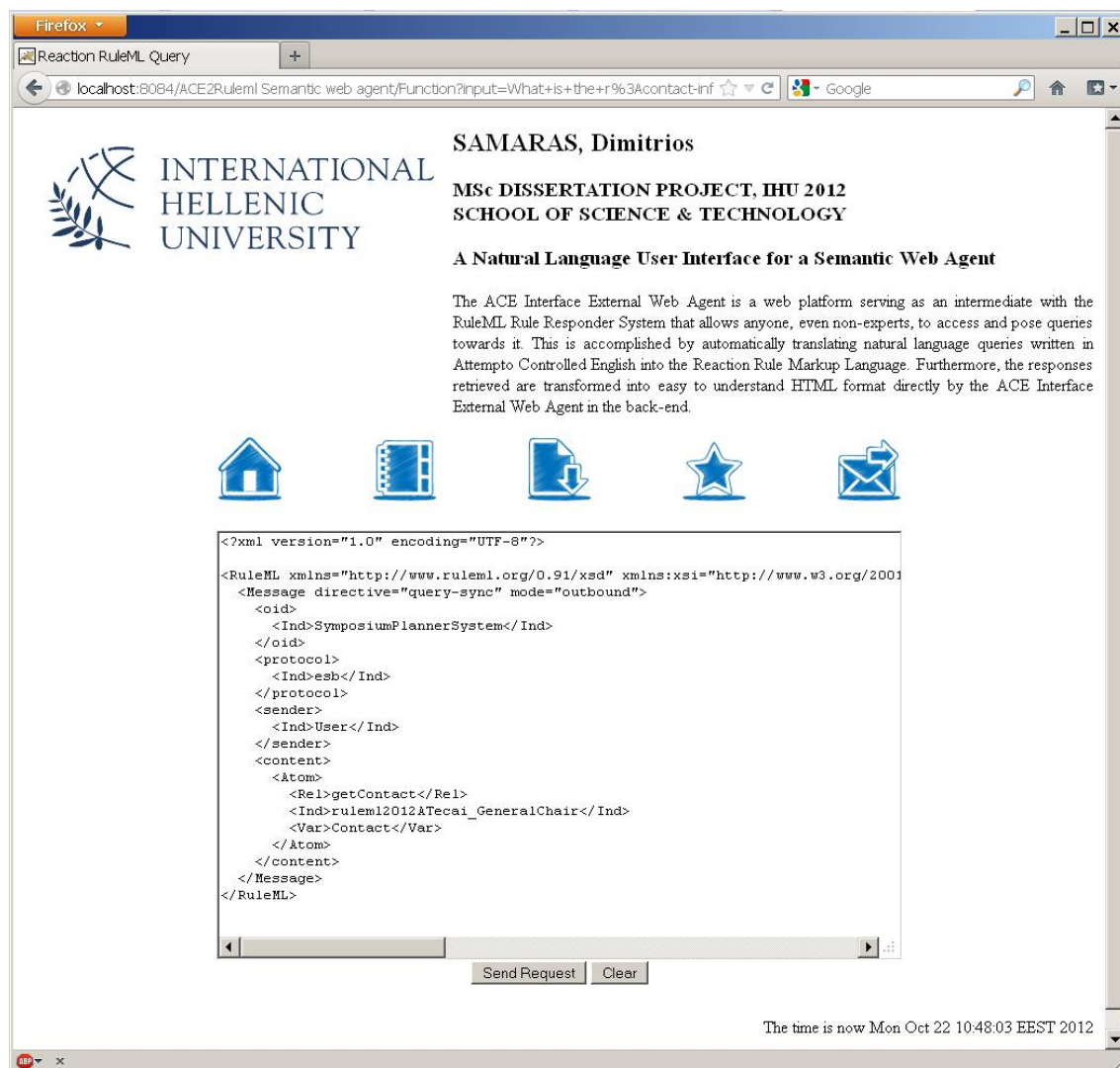


**Figure 4.4.3: ACE Interface External Web Agent Query analysis result a.**



**Figure 4.4.4: ACE Interface External Web Agent Query analysis result b.**

The second action button “Contact Rule Responder” initiates the operation of transmitting the query to the Symposium Planner super-organizational agent and receiving the response. By clicking on the button the translation of the input text to Reaction RuleML is initiated. The resulting page is the query translated into Reaction RuleML in order to be sent to the organizational agent that assist the symposium chairs of the RuleML symposia, *figure 4.4.5*. The resulting Reaction RuleML as it is displayed at the text area can also be modified by the user before being sent to the organizational agent.

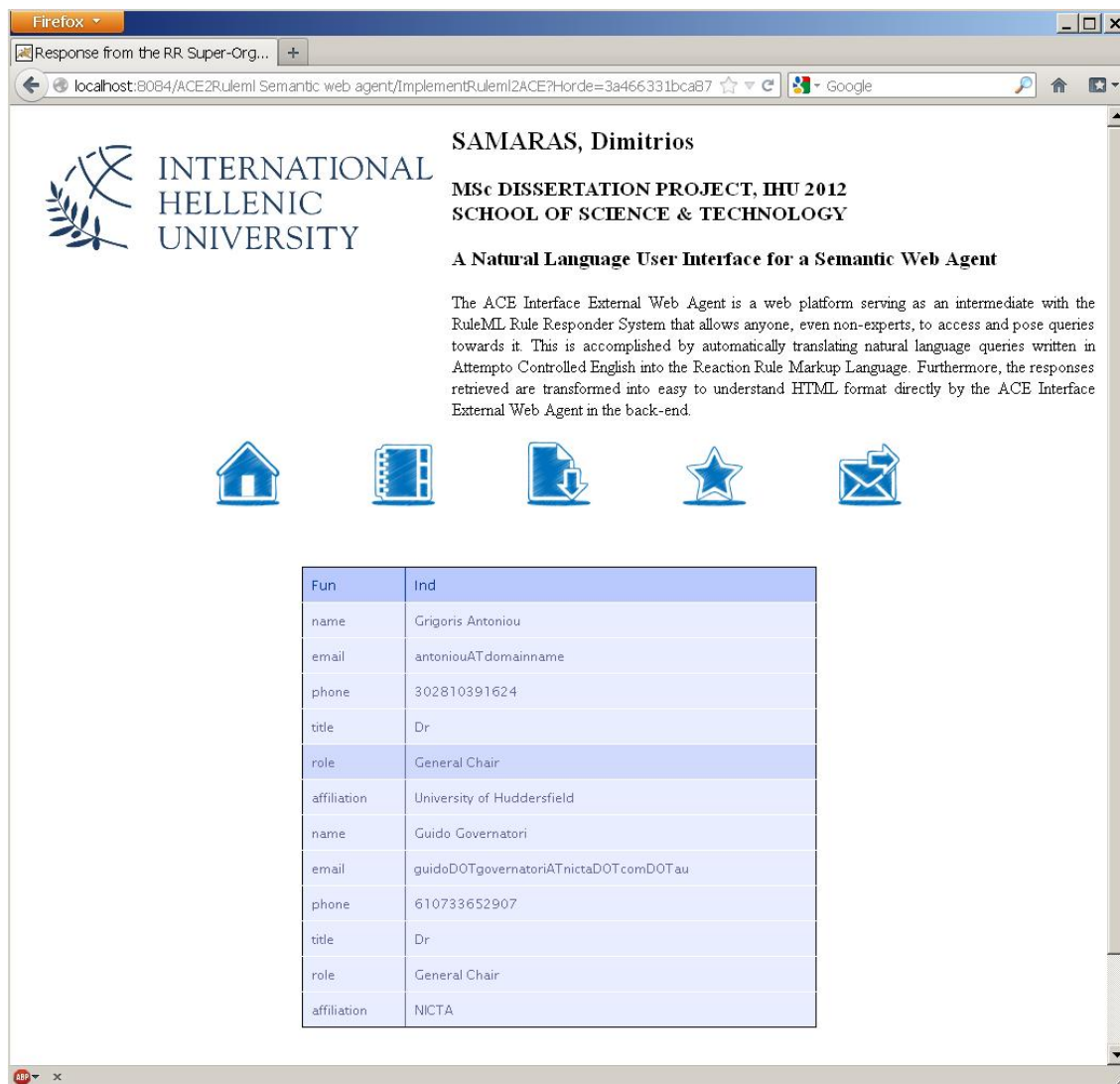


**Figure 4.4.5: ACE Interface External Web Agent Query transformed to Reaction RuleML.**

As a following step of the posing the query procedure the “Send Request” action button is responsible for sending the Reaction RuleML formed query to the Semantic Web



agents, and then receiving and translating the answer in an easy-to-read format. The answer initially is received as Reaction RuleML, then it is processed internally and finally it is displayed as a simple HTML document, *figure 4.4.6*.



**Figure 4.4.6: ACE Interface External Web Agent Response page.**

#### 4.4.1 Additional Content

Apart from the ACE Interface External Web Agent additional explanatory content about the project and assisting content for the use of the external agent is provided for the users.

#### 4.4.1.1 About.html

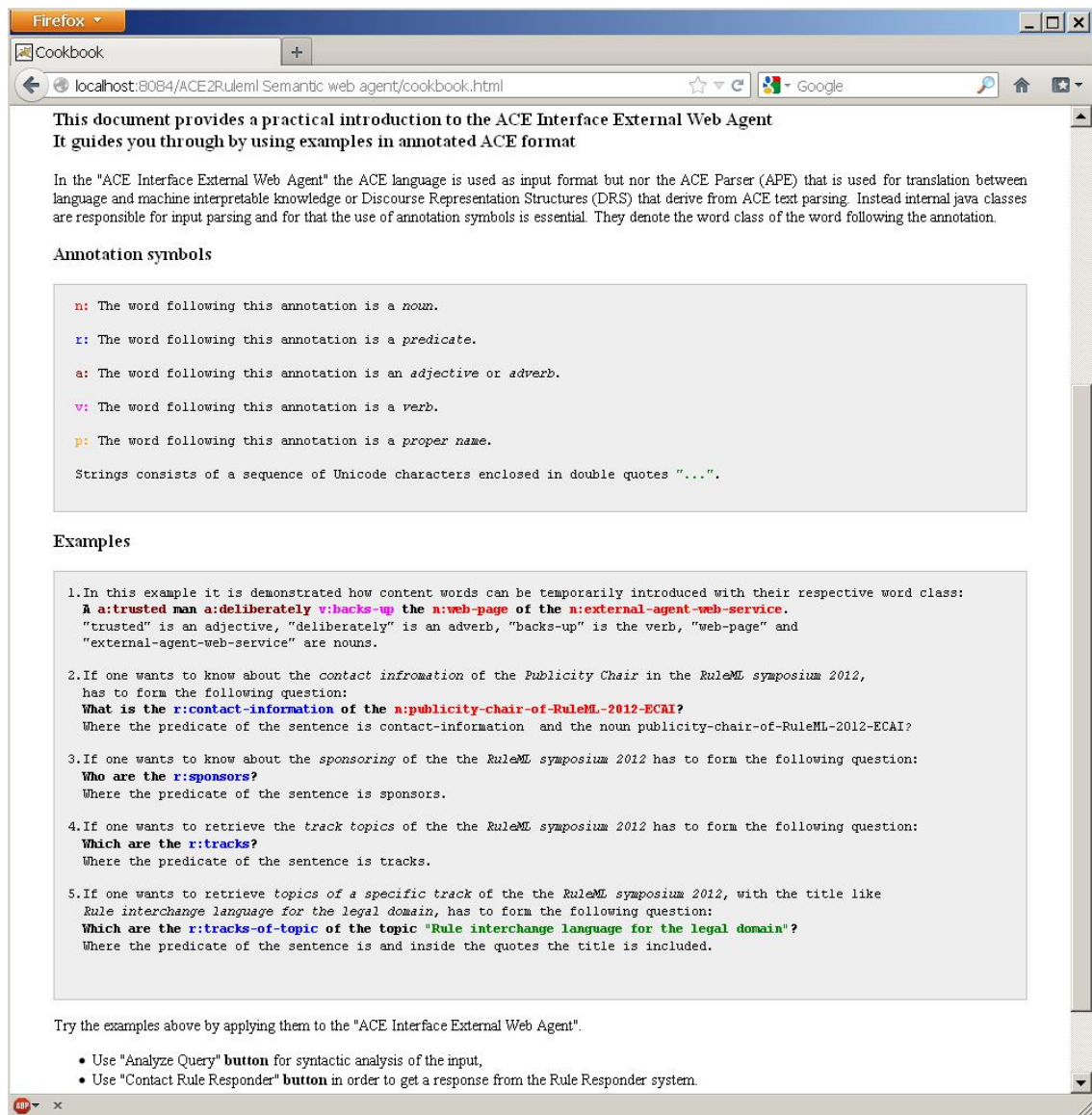
A link containing useful information about the author and the academics involved in the project and the project abstract.



**Figure 4.4.7: About.html Snapshot**

#### 4.4.1.2 Cookbook.html

Use cases are provided on how to use the ACE Interface External Web Agent with the assistance of the set of predefined queries and how to annotate text in order for the user to compose its own text.



**Figure 4.4.8: Cookbook.html Snapshot**

#### 4.4.1.3 Contact Author - hyperlink

Simple email contact form.

#### 4.4.1.4 Download the paper - hyperlink

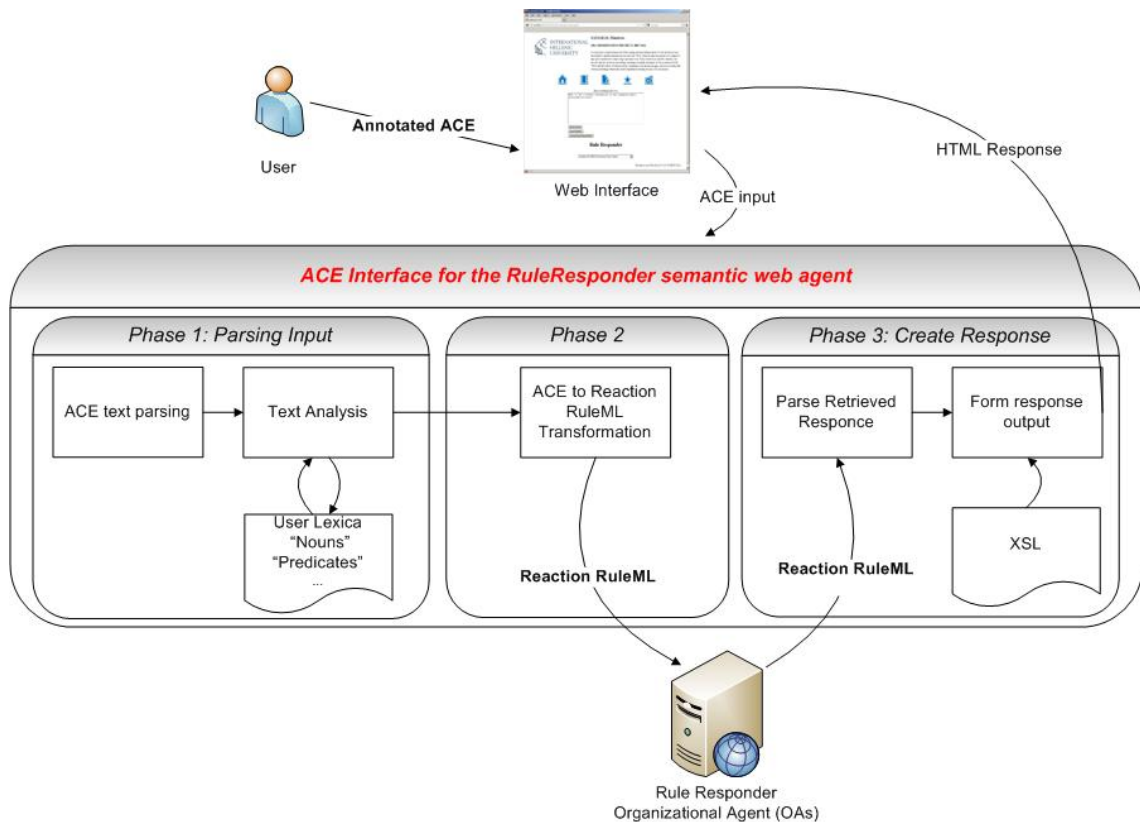
The thesis submitted for the degree of Master of Science (MSc) in Information and Communication Systems from International Hellenic University with the title "A natural language user interface for a semantic web agent" can be found in this hypelink.

## 4.5 Translation Procedure

After the user has inserted his/her input in the text area of the initial page or since a pre-defined query is selected, the translation process can be broken down into the following 9 basic steps:

1. The input text is read and a primary check is initiated whether the sentence is empty,
2. The input text is parsed into sentences,
3. Sentences are parsed into words,
4. Words are characterized according to their form and meaning
  - a. Function words, meaning word denoting the beginning of a query (“What”, “Who”, “How”, “Where”, “When”, “Which”).
  - b. Categorized words: “Predicate”, “Noun”, “Adjective/Adverb”, “Verb”, “Proper Name” as they can be found in the dictionaries.
  - c. Nonfunctional, non-predefined words marked as “other”.
  - d. Change of sentence – end of query characters (“?”, “!”, “.”),
5. Input is translated to the appropriate Reaction RuleML XML serialized format,
6. The Reaction RuleML output of the previous steps is displayed in aJSP file for the user to change or modify the output and proceed with the next steps,
7. The output of *step 5*(as possibly altered by *step 6*) is sent to the Rule Responder Organizational Agent as a server request,
8. Once a response is transmitted back from the Rule Responder Organizational Agent, the Reaction RuleML message is parsed, and finally,
9. The answer is transformed into HTML with the use of XSLT and displayed to the user.

Interpreting ACE to Reaction RuleML and Reaction RuleML to easy-to-read format output can be depicted in three major phases as it can be seen in the *figure 4.5.1*.



**Figure 4.5.1: Schematic representations of the three phases of ACE to Reaction RuleML and Reaction RuleML to HTML translation.**

## 4.6 Classes and Methods

### 4.6.1 Parsing the input

In every possible action as a starting point the user input is parsed to sentences, then to words and words are characterized according to the dictionaries or whether they have a functional purpose. For every action a separate action button exists.

```

<input TYPE="submit" name="com" value="Analyze Query" >
<input TYPE="submit" name="com" value="Contact Rule Responder" >

```

When a button is pressed its value is transferred to the java servlet class "ImplementACE2RuleML".

With the following code the input is checked if it is blank and the parsing to sentences is initiated.

```
// Check there is an input or if the text area is blank!
if ((input == null) || input.equals("")) {
    displayResult(response, com, "Fails. No input. <br>Please go to
main page and enter ACE query");
} else {
    // Retrieve input and parse using ".", "!" and "?" delimiters.
    String[] sentence =
GlobalFunctions.stringToSentenceArr(input, ".!?" );
```

As described above, sentences are separated by the symbols ".", "!" and "?". The "stringToSentenceArr" method is displayed bellow. String arrays are created while parsing the input in order to separate words and characterize them based on the dictionaries.

```
public static String[] stringToSentenceArr(String wordString , String
nontokenDelims) {
    //String nontokenDelims = ".?!";
    String[] result;
    // index into the next empty array element
    int i = 0;
    //--- Declare and create a StringTokenizer
    StringTokenizer st =new StringTokenizer(wordString,nontokenDelims);
    //--- Create an array which will hold all the tokens.
    result = new String[st.countTokens()];
    //--- Loop, getting each one of the tokens
    while (st.hasMoreTokens()) {
        result[i++] = st.nextToken();
    }
    return result;
}
```

The same way sentences are parsed into words "tokens" and instead of using delimiters ".", "!" and "?", space " " is taken into account.

```
public static String[] stringToTokensArr(String wordString){
    String[] result;
    int i =0;

    //--- Declare and create a StringTokenizer
    StringTokenizer st =new StringTokenizer(wordString);

    //--- Create an array which will hold all the tokens.
    result=new String[st.countTokens()];

    //--- Loop, getting each of the tokens
    while(st.hasMoreTokens()){
        result[i++] = st.nextToken();
    }
    return result;
}
```

```
}
```

As long as sentences are broken down to tokens, each token is looked up for special annotations that indicate its grammatical meaning so that is correctly transformed later on to the element nodes or double quotes that indicate that an element node with string type attribute needs to be created. The "findAnnotation" method of the "GlobalFunctions" class is presented bellow.

```
public static String findAnnotation(String s) {
    String strType;
    strType = null;
    String subStr;

    if (s.length() > 2 && s.charAt(0) == '"') {
        subStr = s.substring(1, s.length());
        s = subStr;
        strType = "user defined string start";
    } else if (s.length() > 2
        && s.charAt(s.length()-1) == '"') {
        subStr = s.substring(0, s.length() - 1);
        s = subStr;
        strType = "user defined string end";
    } else if (s.length() > 2 || s.charAt(1) == ':') {
        if ((s.charAt(1) == ':') && (s.charAt(0) == 'r')) {
            subStr = s.substring(2, s.length());
            s = subStr;
            strType = "Predicate";
        } else if ((s.charAt(1) == ':') && (s.charAt(0) == 'n')) {
            subStr = s.substring(2, s.length());
            s = subStr;
            strType = "Noun";
        } else if ((s.charAt(1) == ':') && (s.charAt(0) == 'a')) {
            subStr = s.substring(2, s.length());
            s = subStr;
            strType = "Adjective/Adverb";
        } else if ((s.charAt(1) == ':') && (s.charAt(0) == 'v')) {
            subStr = s.substring(2, s.length());
            s = subStr;
            strType = "Verb";
        } else if ((s.charAt(1) == ':') && (s.charAt(0) == 'p')) {
            subStr = s.substring(2, s.length());
            s = subStr;
            strType = "ProperName";
        }
    }
    if(strType != null){
        return s + " : " + strType;
    }else return s;
}
```

When a word (token) has a grammatical meaning then it is displayed next to the word in the query analysis (*figures 4.4.3, 4.4.4*). Finally, input is looked up for function words denoting request (query) operation.

```
public static HashSet<String> functionWords =
new HashSet<String>(Arrays.asList(new String[] {
    "What", "Who", "How", "Where", "When", "Which"
}));

public static boolean isFunctionWord(String s) {
    return functionWords.contains(s);
}
```

Function words are referenced in the end of the query analysis along with the number of the occurrences in case there is more than one.

#### 4.6.2 Creating the Reaction RuleML request

In order to create the Reaction RuleML request an XML-like document is created and displayed to the user (*figure 4.4.5*) and onwards sent to the Rule Responder organizational agent. This is implemented by the “XMLTreeCreator” class. For this purpose dom4j open source library is used [Dom4j (n.d)].

```
public static Document createDocument(ArrayList tokens) {
    //initialization of new document
    Document document = DocumentHelper.createDocument();
    String xmlns = "http://www.ruleml.org/0.91/xsd";
    //Create root element, add attributes
    Element root = document.addElement("RuleML", xmlns);
    root.addAttribute("xmlns:xsi",
        "http://www.w3.org/2001/XMLSchema-instance");
    root.addAttribute("xsi:schemaLocation",
        "http://www.ruleml.org/0.91/xsd");
    root.addAttribute("xmlns:ruleml2012",
        "http://ibis.in.tum.de/projects/paw#");
}
```

After the new document is initialized the root element of the tree is created. The xml namespace since it cannot be imported as a root element attribute is inserted as a string input when root is created. Element nodes that are the same for every request are created in the appropriate format so that the Reaction RuleML request message is correctly created, code fragment presented bellow.



```

//Create subsequent elements
Element message =
root.addElement("Message").addAttribute("directive", "query-
sync").addAttribute("mode", "outbound");

Element oid = message.addElement("oid");

Element ind =
oid.addElement("Ind").addText("SymposiumPlannerSystem");

Element protocol = message.addElement("protocol");
ind = protocol.addElement("Ind").addText("esb");

Element sender = message.addElement("sender");
ind = sender.addElement("Ind").addText("User");

Element content = message.addElement("content");
Element atom = content.addElement("Atom");

```

Atom nodes consist of sets of element nodes. The element nodes are named and filled up according to the grammatical analysis of the input text. In the following code fragment the way words are distinguished according to their grammatical meaning with the use of annotations is presented.

```

for (int i = 0; i < tokens.size(); i++) {
String prod = null;
String ToNode = (String) tokens.get(i);
//Search for char "n","r" or double quotes
char c = ToNode.charAt(0);
switch (c) {
case 'r':
//if the word-token is annotated as a prepricate "r" then look
at the predicate.txt
String predicate = SearchTxt-
File.SearchThe("C:\\predicate.txt", ToNode.substring(2,
ToNode.length()));
Element elmnt1 = atom.addElement("Rel").addText(predicate);
break;
case '"':
//if the word-token is annotated as a user defined string inside
"quotes"
String tempstr = "";
for (int j = 0; j < tokens.size(); j++) {
// Reconstruct the initial user input to get the value be-
tween the "quotes"
String temp = (String) tokens.get(j);
tempstr = tempstr + " " + temp;
}
Pattern p = Pattern.compile("\"([^\"]*)\"");
Matcher m = p.matcher(tempstr);
while (m.find()) {
prod = m.group(1);
}
Element elmnt2 = atom.addElement("Ind").addText(prod);
//add the type="string" attribute to the node
elmnt2.addAttribute("type", "string");
}
}

```

```

        break;
    case 'n':
        //if the word-token is annotated as a noun "n" then look at the
        noun.txt
        String noun = SearchTxtFile.SearchThe("C:\\noun.txt",
        ToNode.substring(2, ToNode.length()));
        Element elmnt3 = atom.addElement("Ind").addText(noun);
        break;
    }
}
Element elmnt4 = atom.addElement("Var")
                .addText("Contact");
return document;

```

Element nodes that hold the information about the request are included inside the “Atom” node. An <Atom> node with children nodes that needs to be created from the code presented previously has the following format:

```

<Atom>
  <Rel>getContact</Rel>
  <Ind>ruleml2012ATecai_GeneralChair</Ind>
  <Var>Contact</Var>
</Atom>

```

In order for the produced request to be timely displayed to the user, the following JSP code is employed:

```

<textarea name="content" rows="20" cols="80" wrap="off">
<% try {
    FileInputStream fstream = new FileInputStream("C:\\ACE2Ruleml Se
mantic web agent\\web\\output.xml");
    // Get the object of DataInputStream
    DataInputStream in = new DataInputStream(fstream);
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    String strLine;
    //Read File Line By Line
    while ((strLine = br.readLine()) != null) {
        // Print the content on the console
        out.println(strLine);
    }
    //Close the input stream
    in.close();

    } catch (Exception e) {
        //Catch exception if any
        System.err.println("Error: " + e.getMessage());
    }
}%></textarea>

```

Through the JSP presented above, the request XML document is called directly when the page opens. Otherwise the previous produced result would be displayed, although the new one would be produced. This happened because the page opened before the result was cached in the server. With this solution the page is forced to open directly once the result is produced and the problem where the resulting Reaction RuleML message was displayed after the XML document was refreshed manually through the Apache server is avoided.

### 4.6.3 Transmitting the request

Once the request is created and displayed to the user (*figure 4.4.5*) when the button “Send message” is pressed the “ImplementRuleML2ACE” Java Servlet class is called.

```
String iniUrl = "http://131.202.242.154:8888";
String com2;
String content;
String horde;

if (com2.equals("Send Message")) {
    content = request.getParameter("content");
    horde = request.getParameter("Horde");
    //encode string to url
    String contURL = URLEncoder.encode(content);
    //add all parts of the url form the correct adress
    URL myUrl = new URL(iniUrl + "?Horde=" + horde + "&content=" +
        contURL);
    //establish connection and get response from server
    URLConnection conn = myUrl.openConnection();
    ...
}
```

In order to contact the Rule Responder a URL containing the Rule Responder IP address with the complete request message needs to be created. The IP address where Rule Responder is hosted is **131.202.242.154** and the port is **8888**.

### 4.6.4 Collecting and Storing the Response

As soon as the Rule Response publishes the response, it is caught by the “ImplementRuleML2ACE” Java Servlet class in the back end without being displayed to the user and directly saved as a string.

```

//write response to string
BufferedReader in = new BufferedReader(
    new InputStreamReader(conn.getInputStream()));
StringBuffer sb = new StringBuffer();

String inputLine;

while ((inputLine = in.readLine()) != null) {
    //System.out.println(inputLine);
    sb.append(inputLine);
}
in.close();
String Res = sb.toString();

```

An XML document is created so that it is displayed to the user as an html page with the use of XSL transformation.

```

//save string as XML file
OutputStream outputStream;
byte buf[] = Res.getBytes();

outputStream = new FileOutputStream("C:\\ACE2Ruleml Semantic web
agent\\web\\qAnswer.xml");

for (byte element : buf) {
    outputStream.write(element);
}
outputStream.close();
uf = null;
SimpleXMLTransform.transfXML();
request.getRequestDispatcher ("howToHtml.jsp").forward(request,
response);

```

#### 4.6.5 Displaying the Response to the User

The transformed response is displayed to the user as an HTML page and this is done through the "SimpleXMLTransform" class.

```

static public void transfXML() {
String inXML = "C:\\ACE2Ruleml Semantic web agent\\web\\qAnswer.xml";
String inXSL = "C:\\ACE2Ruleml Semantic web agent\\web\\qAnswer.xsl";
String outTXT = "C:\\ACE2Ruleml Semantic web agent\\web\\howto.html";

SimpleXMLTransform st = new SimpleXMLTransform();

try {
    st.transform(inXML,inXSL,outTXT);
} catch (TransformerConfigurationException e) {
    System.err.println("Invalid factory configuration");
    System.err.println(e);
} catch (TransformerException e) {
    System.err.println("Error during transformation");
}
}

```

```

        System.err.println(e);
    }
}

```

The “transfXML” method accepts as input the response XML file and the necessary XSL file for the transformation and produces the html output of the operation by calling the “transform” method. The code of the “transform” method is presented below:

```

public void transform(String inXML,String inXSL,String outTXT)
    throws TransformerConfigurationException,
    TransformerException {

    TransformerFactory factory = TransformerFactory.newInstance();

    StreamSource xslStream = new StreamSource(inXSL);
    Transformer transformer = factory.newTransformer(xslStream);
    transformer.setErrorListener(new MyErrorListener());

    StreamSource in = new StreamSource(inXML);
    StreamResult out = new StreamResult(outTXT);
    transformer.transform(in,out);
}

```

Once again in order to have a timely response and ensure that the correct response is displayed to the user the following JSP code is applied in the webpage where the final output is presented (*figure 4.4.6*):

```

<% try {
    // Open the file that is the first
    // command line parameter
    FileInputStream fstream = new FileInputStream("C:\\ACE2Ruleml
Semantic web agent\\web\\howto.html");
    // Get the object of DataInputStream
    DataInputStream in = new DataInputStream(fstream);
    BufferedReader br = new BufferedReader(new InputStreamReader
    (in));
    String strLine;
    //Read File Line By Line
    while ((strLine = br.readLine()) != null) {
        // Print the content on the console
        out.println(strLine);
    }
    //Close the input stream
    in.close();
} catch (Exception e) { //Catch exception if any
    System.err.println("Error: " + e.getMessage());
}
%>

```

## 4.7 Agent Installation Manual

The complete project with all of its components can be downloaded from the following link: [https://www.dropbox.com/sh/6g4562og06e6rhu/vDN7ATGv\\_2](https://www.dropbox.com/sh/6g4562og06e6rhu/vDN7ATGv_2).

In order to run the project on localhost, “Netbeans”, preferably, or any other integrated development environment (IDE) for java web applications and a web browser are required along with an internet connection. The steps to execute the project are described below:

1. Import project "ACE Interface External Web Agent" on the hard drive "C:\".
2. Import contents of "lib\_files\_on\_c" and "project\_files\_on\_c" folders directly on the hard drive "C:\".
3. Open project using Netbeans.
  - 3a. Apache tomcat is necessary for the project to run on localhost. If not installed it can be found on: <http://tomcat.apache.org/download-60.cgi>
4. When Netbeans opens the project will encounter an error as the external library files cannot be found! The .lib files are on the hard drive. Selecting the resolve issues suggestion by Netbeans will initiate the guide to find the missing library file. Selecting .lib file will fix all issues!
5. Compile and Run project.

Alternatively the ACE Interface External Web Agent can be accessed online at:

[http://iskp.eu/ACE\\_Interface\\_External\\_Web\\_Agent/](http://iskp.eu/ACE_Interface_External_Web_Agent/)

\* Suggested web browsers for the "ACE Interface External Web Agent" are Google Chrome, Mozilla Firefox and Internet Explorer 9.0.1 or later version.

## 5 Conclusions

The market requirements (e.g. B2B applications) were those that raised at high priority the need for semantic description of services. The rigid model of a keyword-based search service, leads to many false-positive results (low precision), limited degree of flexibility because of the limited criteria and, most important, ignorance of what the search results are really about, having as a consequence the thorough inspection of each result, which is a time consuming and often expensive process. By incorporating semantic features describing the services or websites, the search is based on what really the service is about, rather than ineffective string comparison.

Therefore, it is imperative that the World Wide Web evolves into something much more than just a platform that allows users and organizations to host websites. There is a need to develop a mechanism which will allow machines to collect content from different sources, processing information and exchange the results with other computer programs (agents) and users.

A successful example of employment of multiple-agent-systems for knowledge extraction from the Semantic Web is accomplished by the RuleML symposia with the “Rule Responder System” implementation. The Rule Responder System is a multi-agent system for collaborative team and community support on the Semantic Web that enables the rule-based collaboration between the distributed human members of such a virtual organization. Persons of an organization are assisted by semi-automated rule-based agents, which use rules with the use of Reaction RuleML as knowledge interchange language, to describe the decision and behavioral logic.

The External Agent of the Rule Responder System, called *Symposium Planner*, assists in retrieving information about the symposium chairs of the RuleML symposia by communicating with the Organizational and the Personal Agents. Although this External Agent is accessible by anyone who wants to get informed about the symposium, it requires the knowledge of Reaction RuleML. In order to offer a simple and easy to use web interface to retrieve information from the Rule Responder System and create an

External Web Agent that uses the state of art controlled natural language the “*ACE Interface External Web Agent*” project has been implemented.

The developed *ACE Interface External Web Agent*, presented in the 4<sup>th</sup> chapter, is an open source semi-autonomous web platform serving as an External Agent intermediate with the RuleML Rule Responder Organizational Agent allowing everyone to access it and write queries in natural language without any knowledge of Reaction RuleML employed by the Rule Responder multi-agent system. Sentences expressed by the user in annotated ACE are translated into Reaction RuleML so that communication can be established with the Rule Responder System. The aim is that this user-friendly front-end will offer an alternative to the RuleML SymposiumPlanner-2012 and additionally in future extensions will help lower the barrier of entry to the Semantic Web and encourage non-experts to get involved.

As part of the conclusions, the problems met during the implementation of the project are presented in section 5.1. During the implementation period, several improvement possibilities were discovered which are presented in section 5.2 and are suggested to be studied and implemented in the future. Lastly in this chapter the additional applications of the “*ACE Interface External Web Agent*” implementation are discussed.

## 5.1 Problems met during the Implementation

### 5.1.1 XPath Navigation Incompatibility

The use of XPath navigation in order to go through elements and attributes of the Reaction RuleML response from the Rule Responder Organizational Agent is not feasible. This is due to one of the namespace attributes used in the response message, in the “RuleML” parent node as it is transmitted from the organizational agent. Most specifically: `xmlns=http://www.ruleml.org/0.91/xsd` cannot be found thus the document is not navigated with xpath.

XPath would offer a greater functionality and flexibility in the manipulation of the response. In order to use Xpath navigation each response should be parsed and the problematic namespace should be removed.



As an alternative to the problem addressed above EXtensible Stylesheet Language (XSL) and transformations (XSLT) are used in order to provide an easy-to-read response to the users of the ACE Interface External Web Agent. This approach does not lack functionality or display feasibility to the XPath solution. The only drawback is that for responses formatted in different ways have to be accompanied by separate stylesheets because of the different child element that have to be read in every case.

### 5.1.2 SymposiumPlanner-2012 Unavailability

Rule Responder server is unresponsive at times. However, the most recent Rule Responder version is significantly more reliable than previous versions, which were either delaying too long to produce the answer or were frequently unavailable altogether.

### 5.1.3 Prolog Engine Necessity

The use of a Prolog engine for parsing and analyzing input in FOL (First Order Logic), proved to be a necessity. This is due to the fact that even though ACE is well structured only through DRS analysis of the input and then with syntactical analysis along with the use of user-defined dictionaries can knowledge be extracted in its fullest extend. In this matter the use of APE (Attempto Parsing Engine) is the most accepted solution, as documented in the literature, because of the long lasting work and evolution on the Attempto project and its tools.

For example for the transformation of the following query:

Which are the `r:news-feeds`?

Into the necessary Reaction RuleML format in order to receive an appropriate answer from the Rule Responder System (only the Atom node is displayed):

```
<Atom>
  <Rel>mediaNewsFeedResource</Rel>
  <Var>Meeting</Var>
  <Var>Site</Var>
  <Var>News</Var>
</Atom>
```

The predicate of the sentence <Rel> can derive from the annotated ACE query but the variables <Var> that are necessary for the construction of the rule cannot be defined from the sentence as it is.

Such a construction could only be accomplished with the use of analysis into machine interpretable logic and rich user defined lexica. As an example the ACE reasoner (RACE) can be provides where ACE text is translated to DRS. Further on with DRS and a complete set of rules and lexica the correct assumptions can be made.

## 5.2 Future Improvements

During the implementation of the project and through the research conducted during this period, several issues should have been addressed and improvement possibilities were discovered that are suggested to be studied and implemented in the future.

There are several possible opportunities of future development for the ACE Interface External Web Agent. These improvements refer to the possibility of a greater range of query analysis and transformation and involve collaboration with the Attempto project tools as described in section 5.1.4.

### 5.2.1 Text analysis and Interpretation

The ACE Interface External Web Agent would have an even more user friendly interface by offering text prediction and vocabulary assisting functions, such as those that modern search engines offer (i.e. Google<sup>8</sup>).

- Text predictor to work at real-time as the user inputs its own text at the ACE Interface External Web Agent's input text area.
- Text analyzer to correct user input in case of misspell or syntactic errors.

Furthermore, by enhancing the External Agent dictionaries and with the option for greater user interaction would lead to a global application.

---

<sup>8</sup> [www.google.com](http://www.google.com)

- Design of more detailed dictionaries, covering a greater number of words and different parts of the speech.
- Implement the possibility for users to define their custom dictionaries in order to use them with Organizational and External Agents of different domains.

### 5.2.2 Collaboration with ACE tools

The Attempto project, while developing ACE language, has also developed a set of powerful open source tools that are easy to use and offer a stable basis for further development. Thus we recommend the adherence of the External Agent with ACE Editor and RACE.

- Collaboration with RACE (ACE Reasoner offered by the Attempto project) for consistency, grammatical and syntactical checking of the user input text.
- Collaboration with ACE Editor (offered by the Attempto project) for allowing user defined lexica as implemented by the online tool.

## 5.3 Business Applications

A further significant part of this thesis is to specify which are the potential applications of such an implementation apart from offering a controlled language interface for the Rule Responder Organizational Agent of the RuleML symposium.

The ACE Interface External Web Agent for the Semantic Web outside the scientific community boundaries can serve as an External Agent for any structured Semantic Web agent's system applied over various domains. Since it is implemented as a web application with low requirements, it is easily installed and widely accessible. Furthermore, the dictionaries used for ACE to Reaction RuleML mapping can be enriched or changed, so that it perfectly fits every given domain.

A sample application could potentially be an add-on to *PLIS* [Viktoratos, I et.al, 2008], a Personalized Location Information System that delivers personalized and contextualized information to users according to rule-based policies. These policies are represented via Reaction RuleML, making the ACE Interface External Web Agent a perfect fit to

serve as the front-end for users that are not familiarized with the specifics of the RuleML syntax.

# Bibliography

[APE Webclient Help., 2008] APE Webclient Help. (2008). Retrieved June 25, 2012, from APE Webclient Help website:  
[http://attempto.ifi.uzh.ch/site/docs/ape\\_webclient\\_help.html](http://attempto.ifi.uzh.ch/site/docs/ape_webclient_help.html)

[APE Webservice., 2010] APE Webservice. (2010, November 04). Retrieved June 25, 2012, from APE Webservice website:  
[http://attempto.ifi.uzh.ch/site/docs/ape\\_webservice.html](http://attempto.ifi.uzh.ch/site/docs/ape_webservice.html)

[Baader F., et al., 2009] Baader Franz, Andreas Bauer, Peter Baumgartner, Anne Cregan, Alfredo Gabaldon, Krystian Ji, Kevin Lee, Dave Rajaratnam and R. Schwitter. 2009. A Novel Architecture for Situation Awareness Systems, In: Proceedings of TAB-LEAUX 2009, LNAI 5607, pp. 77–92.

[Berners-Lee, T. et al., 2001] Tim Berners-Lee, James Hendler, Ora Lassila(2001, May 17). “The Semantic Web”. Scientific American Magazine at ScientificAmerican.com

[Boley, H., Craig B. 2008] Harold Boley, Benjamin Craig. “Rule Responder Agents in Virtual Organizations” Institute for Information Technology National Research Council, Canada Fredericton, NB, Canada, presentation at APICS 2007, 3 December 2008.

[Boley, H. 2010] Harold Boley. “Distributed Rule Responder Querying on the Semantic Web” Institute for Information Technology National Research Council, Canada Fredericton, NB, Canada, Keynote presentation at ICDIM, 6 July 2010.

[Bry, F., 2008] Bry, F. (2008, February 29). Reasoning on the Web with Rules and Semantics. Retrieved June 25, 2012, from REWERSE : <http://rewerse.net>

[Chapin, D., 2010] Chapin, D. (2010, June 22). SBVR: A Standard for the Language of Business and its Policies and Rules. OMG Business Rules Standards Symposium. Business Semantics Ltd.

[Clark, P. et al., 2005] Clark, P., Harrison, P., Jenkins, T., Thompson, J., & Wojcik, R. (2005). Acquiring and Using World Knowledge using a Restricted Subset of English. Mathematics and Computing Technology Boeing Phantom Works. Seattle.

[Dom4j (n.d)] dom4j open source library for working with XML, XPath and XSLT on the Java platform. Retrieved June 25, 2012, <http://dom4j.sourceforge.net/dom4j-1.6.1/index.html>.

[Feigenbaum L, 2007] Lee Feigenbaum (May 1, 2007). "The Semantic Web in Action". Scientific American Magazine at ScientificAmerican.com. Retrieved September 24, 2012.

[Fuchs, N. E. et. al., 2005] Norbert E. Fuchs, Stefan Hofler, Kaarel Kaljurand, Fabio Rinaldi and Gerold Schneider (2005). Attempto Controlled English: A Knowledge Representation Language Readable by Humans and Machines. [book auth.] N. E. Maluszynski, Reasoning Web 2005 (pp. 213-250). Heidelberg: Springer-Verlag.

[Fuchs N.E, et.al. 2005b] Norbert E. Fuchs, Kaarel Kaljurand, and Gerold Schneider. Deliverable I2-bD5. Verbalizing Formal Languages in Attempto Controlled English I. Technical report, REWERSE, 2005. <http://rewerse.net/deliverables.html>.

[Fuchs, N. E., 2006] Fuchs, N. E. (2006). Attempto project. Retrieved June 25, 2012, from Attempto project website: <http://www.ifi.unizh.ch/attempto>

[Fuchs, N. E. et. al., 2008] Fuchs, N.E., Kaljurand, K., Kuhn, T. (2008). Discourse Representation Structures for ACE 6.0. Zurich: Technical Report ifi-2008.02, Department of Informatics, University of Zurich, Switzerland.

[Fuchs, N. E. et. al., 2008b] Norbert E. Fuchs, Kaarel Kaljurand, Tobias Kuhn (2008). Attempto Controlled English for Knowledge Representation. Bonn: University of Bonn.

[Fuchs, N. E., 2011] Fuchs, N. E. (2011, December 15-16). Reasoning in Attempto Controlled English. Conference on Computing Natural Reasoning COCONAT . Tilburg.

[Fuchs, N. E., 2012] Fuchs, N. E. (2012, January 12-13). Attempto Controlled English and its tools. MOLTO Extended Kick-Off Meeting , 1-41. Gothenburg.

[Hall, J., 2006] Hall, J. (2006, May 23). Semantics of Business Vocabulary and Business Rules (SBVR). Presented at the BPM Think Tank. Alington, VA.

[Halpin, T., 2006] Halpin, T. (2006). Business Rule Modality. Presented at EMMSAD '06 the 11th International Workshop on Exploring Modeling Methods in System Analysis and Design. Luxembourg.

[Hirtle, D. Z., 2006] Hirtle, D. Z. (2006, October). TRANSLATOR: A TRANSLator from LANGUAGE TO Rules. Canadian Symposium on Text Analysis (CaSTA) . Fredericton, Canada.

[Hirtle, D. et al., (n.d.)] David Hirtle, Tshering Dema, Harold Boley (n.d.). The Modularization of RuleML. Retrieved July 12, 2012:  
<http://ruleml.org/modularization/#Model>

[Hoefler, S., 2004] Hoefler, S. (2004). The Syntax of Attempto Controlled English: An Abstract Grammar for ACE 4.0. Zurich: Technical Report ifi 2004.03.

[Internet World Stats, 2012] Internet World Stats (2012), “TOP 20 COUNTRIES WITH THE HIGHEST NUMBER OF INTERNET USERS World Internet Users and Population Stats”, Retrieved October 9, 2012 from: <http://www.internetworldstats.com/top20.htm>

[Juri Luca De Coi, et al., 2009] Juri Luca De Coi, Norbert E. Fuchs, Kaarel Kaljurand, and Tobias Kuhn. Controlled English for Reasoning on the Semantic Web (2009). [book auth.] F.Bry and J.Maluszynski. Semantic Techniques for the Web. Heidelberg : Springer-Verlag, pp. 276-307.

[Kaljurand, K., 2007a] Kaljurand, K. (2007). ACE View - an ontology and rule editor based on controlled English.

[Kaljurand, K., 2007b] Kaljurand, K. (2007). Attempto Controlled English as a Semantic Web language. Tartu: Faculty of Mathematics and Computer Science, University of Tartu.

[Kaljurand, K., 2011a] Kaljurand, K. (2011, june 14). ACE View Ontology and Rule Editor. Retrieved june 26, 2012, from ACE View website: <http://attempto.ifi.uzh.ch/aceview/>

[Kaljurand, K., 2011b] Kaljurand, K. (2011, August 16). APE parser. Retrieved June 25, 2012, from APE parser website: <http://attempto.ifi.uzh.ch/site/tools/>

[Kaljurand, K., 2011c] Kaljurand, K. (2011, August 16). Reasoner ACE. Retrieved June 26, 2012, from Reasoner ACE website: <http://attempto.ifi.uzh.ch/site/tools/>

[Koivunen M.R., Miller E., 2001] Marja-Riitta Koivunen and Eric Miller, W3C Semantic Web Activity. Published on *Semantic Web Kick-off Seminar in Finland Nov 2, 2001*. Retrieved November 20, 2012, from <http://www.w3.org/2001/12/semweb-fin/w3csw>



[Kontopoulos E. et. al., 2011] Efstratios Kontopoulos, Nick Bassiliades, Grigoris Antoniou, Visualizing Semantic Web proofs of defeasible logic in the DR-DEVICE system. Journal: Knowledge Based Systems - KBS , vol. 24, no. 3, pp. 406-419, 2011

[Kontopoulos E. et. al., 2008] Efstratios Kontopoulos, Nick Bassiliades, Grigoris Antoniou, Deploying defeasible logic rule bases for the semantic web. Journal: Data & Knowledge Engineering - DKE , vol. 66, no. 1, pp. 116-146, 2008

[Kuhn, T., 2007] Kuhn, T. (2007). AceRules: Executing Rules in Controlled Natural Language. Zurich: Department of Informatics, University of Zurich, Switzerland.

[Kuhn, T., 2009] Kuhn, T. (2009). Controlled English for Knowledge Representation. Zurich: Faculty of Economics, Business Administration and Information Technology of the University of Zurich.

[Kuhn, T., 2011d] Tobias Kuhn (2011) Attempto project. ACE Documentation, Retrieved June 23, 2012, Attempto project website: <http://attempto.ifi.uzh.ch/site/docs/>

[Kuhn, T., (n.d.)] Kuhn, T. (n.d.). ACE Wiki. Retrieved June 26, 2012, from ACE Wiki website: <http://attempto.ifi.uzh.ch/acewiki/>

[Linehan, M. H., 2009] Linehan, M. H. (2009). SBVR Use Cases. IBM T.J. Watson Research Center. Yorktown Heights, NY 10598.

[Lukichev, G. W., 2007] Lukichev, G. W. (March 2007). Tool improvements/ extensions 2: Verbalization Component. Technical report, REWERSE IST.

[McCarthy, J., Hayes, P. J., 1969] McCarthy, J., Hayes, P. J. (1969), "Some Philosophical Problems from the Standpoint of Artificial Intelligence". In Meltzer, B. and Michie, D., editors, Machine Intelligence 4, pages 463-502. Edinburgh University Press, from <http://wwwformal.stanford.edu/jmc/mcchay69.html>.

[Osmun T., Smith D., 2010] Taylor Osmun, Derek M. Smith(2010). Rule Responder. Retrieved July 08, 2012, from Rule Responder : <http://ruleml.org/RuleResponder/>

[Paschke, A., (n.d.)a] Paschke, A. Reaction RuleML Examples. Retrieved July 12, 2012, from legal ruleml: <https://lists.oasis-open.org/archives/legalruleml/201202/msg00023.html>

[Paschke, A. (n.d.)b] Paschke A., Reaction RuleML 0.2 Schema - Production RuleML Layer. Retrieved July 29, 2012, from <http://ibis.in.tum.de/research/ReactionRuleML/0.2/pr.xsd>.

[Paschke, A. et al., 2007] Paschke, A., Boley, H., Kozlenkov, A., & Craig, B. (2007). Rule responder: RuleML-based agents for distributed collaboration on the pragmatic web. Proceedings of the 2nd international conference on Pragmatic web ICPW '07 (pp. 17-28). New York: ACM.

[Paschke, A. et al., 2007a] Adrian Paschke, Alexander Kozlenkov, Harold Boley, Said Tabet, Michael Kifer, Mike Dean (2007, July 22). Reaction RuleML. Retrieved July 10, 2012, from Reaction RuleML: <http://ruleml.org/reaction/>

[Paschke, A. et al., 2007b] Adrian Paschke, Alexander Kozlenkov, Harold Boley, Said Tabet, Michael Kifer, Mike Dean (2007). Reaction RuleML 0.2 Primer. RuleML consortium. Available: <http://ruleml.org/reaction/0.2/>

[Paschke, A., Boley H., 2011] Adrian Paschke and Harold Boley, Rule Responder: rule-based agents for the semantic-pragmatic web. International Journal on Artificial Intelligence Tools 2011 20:06, 1043-1081

[Peter, C. et al., 2010] Peter, C., Murray, W. R., Harrison, P., & Thompson, J. (2010). Naturalness vs. Predictability: A Key Debate in Controlled Languages. Proceedings 2009 Workshop on Controlled Natural. Seattle: Boeing Research and Technology.

[RACE Web Client Help. (n.d.)] RACE Web Client Help. (n.d.). Retrieved June 26, 2012, from RACE Web Client Help website:  
[http://attempto.ifi.uzh.ch/site/docs/race\\_webclient\\_help.html](http://attempto.ifi.uzh.ch/site/docs/race_webclient_help.html)

[RuleML Symposium, 2010] RuleML 2010. (n.d.). Retrieved July 08, 2012, from The 4th International Web Rule Symposium: Research Based and Industry Focused:  
<http://www.csw.inf.fu-berlin.de/ruleml2010/objectives.html>

[Schwitter, R., 2002] Schwitter, R. (2002). English as a Formal Specification Language. In: Proceedings of the Thirteenth International Workshop on Database and Expert Systems Applications.

[Schwitter, R. et al., 2003] Schwitter, Rolf, Anna Ljungberg, David Hood(2003). ECOLE – A Look-ahead Editor for a Controlled. Proceedings of EAMTCLAW 03, (pp. 141–150).

[Schwitter, R., 2004] Schwitter, R. (2004). Representing Knowledge in Controlled Natural Language: A Case Study. Sydney, Australia: Centre for Language Technology, Macquarie University.

[Schwitter, R., 2009] Schwitter, R. (2009). Working for Two: a Bidirectional Grammar for a Controlled Natural Language. Sydney, Australia: Centre for Language Technology Macquarie University.

[Schwitter, R., 2010] Schwitter, R. (2010). Controlled Natural Languages for Knowledge Representation. Coling 2010: Poster Volume, (pp. 1113–1121). Beijing.

[Schwitter, R., (n.d.)a] Schwitter, R. (n.d.). PENG light processable English. Retrieved July 30, 2012, from <http://web.science.mq.edu.au/~rolfs/PENG-Light.html>

[Schwitter,R., (n.d.)b] Schwitter, R. (n.d.). PENG online processable English. Retrieved July 29, 2012, from <http://web.science.mq.edu.au/~peng/PengEditor.html>

[Tilbrook, M. & Schwitter, R., 2006] Tilbrook, Mark. and Schwitter, Rolf. (2006). Writing RSS Feeds in a Machine-Processable Controlled Natural Language. Sydney, Australia: Centre for Language Technology Macquarie University.

[Viktoratos, I et.al, 2008] I. Viktoratos, A. Tsadiras, N. Bassiliades (2008). Personalizing Location Information through Rule-Based Policies. Retrieved from: [http://iskp.csd.auth.gr/paper\\_details.asp?publicationID=370](http://iskp.csd.auth.gr/paper_details.asp?publicationID=370)

[W3C Semantic Web Activity,2011] “W3C Semantic Web Activity”. World Wide Web Consortium (W3C). November 7, 2011. Retrieved September 24, 2012, from: <http://www.w3.org/2001/sw/>

[Wyner A., 2009]Adam Wyner, Krasimir Angelov, Guntis Barzdins, Danica Damljajnovic, Brian Davis, Norbert E. Fuchs, Stefan Höfler, Ken Jones, Kaarel Kaljurand, Tobias Kuhn: On Controlled Natural Languages: Properties and Prospects. CNL 2009: 281-289. Workshop on Controlled Natural Language, CNL 2009, Marettimo Island, Italy, June 8-10, 2009.

