I5-D9

Prototype on the RDF/OWL level

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R/P (report and prototype)
Dissemination level:	PU (public)
Document number:	IST506779/Lisbon/I5-D9/D/PU/a1
Responsible editors:	José Júlio Alferes
Reviewers:	Wolfgang May
Contributing participants:	Göttingen, Lisbon
Contributing workpackages:	I5
Contractual date of deliverable:	31 August 2007
Actual submission date:	23 November 2007

Abstract
This deliverable concerns the extension of the prototypes introduced in deliverable I5-D5, to deal with evolution and reactivity at the RDF/OWL level, and in accordance with the ontology defined in deliverable I5-D6.

The deliverable is constituted by this report, and by the prototype implementations that are freely available online from `http://rewerse.net/I5/r3/` and `http://www.dbis.informatik.uni-goettingen.de/MARS`.

Keyword List
ECA rules, Reactivity, Evolution and updates of data, OWL ontology, Language and data heterogeneity

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

Prototype on the RDF/OWL level

José Júlio Alferes[1], Ricardo Amador[1], Erik Behrends[2], Tiago Franco[1], Oliver Fritzen[2], Ludwig Krippahl[1], Wolfgang May[2], Franz Schenk[2]

[1] Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa

[2] Institut für Informatik, Universität Göttingen

23 November 2007

Abstract

This deliverable concerns the extension of the prototypes introduced in deliverable I5-D5, to deal with evolution and reactivity at the RDF/OWL level, and in accordance with the ontology defined in deliverable I5-D6.

The deliverable is constituted by this report, and by the prototype implementations that are freely available online from `http://rewerse.net/I5/r3/` and `http://www.dbis.informatik.uni-goettingen.de/MARS`.

Keyword List

ECA rules, Reactivity, Evolution and updates of data, OWL ontology, Language and data heterogeneity

Contents

1 Introduction **1**
- 1.1 On the contents and structure of this deliverable . . . 1
- 1.2 On the current prototypes for evolution and reactivity . . . 2

2 The r^3 Prototype - v0.20 **5**
- 2.1 The r^3 Foundational Ontology . . . 7
  - 2.1.1 An Ontology for Reactive Rules . . . 8
  - 2.1.2 Defining Reactive Rule Languages . . . 9
  - 2.1.3 Defining Reactive Rule Constructions . . . 14
  - 2.1.4 From ECA-ML to r^3DF . . . 17
- 2.2 r^3 Engines and Library . . . 20
  - 2.2.1 Resourceful Reactive Engines and Rules . . . 20
  - 2.2.2 Loading and Evaluating Reactive Rules . . . 24
  - 2.2.3 Variables, Substitutions and Results . . . 32
  - 2.2.4 Rule Instances, et al. . . . 35
  - 2.2.5 Prototype Implementation and Communication Details . . . 36
  - 2.2.6 Building r^3 Component Engines . . . 37
  - 2.2.7 r^3 Component Languages . . . 43

3 r^3 Use-Cases **67**
- 3.1 Use-Cases Scenario . . . 67
  - 3.1.1 Scenario Overview . . . 67
  - 3.1.2 Scenario Use-Cases Overview . . . 69
- 3.2 r^3-based Bio Domain Broker . . . 69
  - 3.2.1 Message Examples . . . 72
- 3.3 r^3-based PubMed Reactive Classifier . . . 75
  - 3.3.1 Message Examples . . . 75
  - 3.3.2 Possible Extensions . . . 100

4 Future Work **101**

Chapter 1

Introduction

1.1 On the contents and structure of this deliverable

Two prototypes have been developed at the XML level for the General Framework for Evolution and Reactivity in the Semantic Web, that have been introduced in the previous deliverable I5-D5 "A first prototype on evolution and behaviour at the XML-Level". Namely, for the general framework, we have developed the framework "MARS – Modular Active Rules for the Semantic Web" and the rule engine for reactive rules "$\boldsymbol{r^3}$ – Resourceful Reactive Rules". These prototypes have been tested in a number of scenarios and use-cases, as described in I5-D7 "Completion of the prototype scenario". Then, in I5-D6 "Reactive rule ontology: RDF/OWL level" we have defined a concrete ontology for reactive rules at RDF/OWL level. This deliverable is concerned with the lifting of the previous prototypes to the RDF/OWL level, according to the defined ontology. It is worth mentioning that, though not required at the time, the $\boldsymbol{r^3}$ prototype used for the example in I5-D7 already included features at the RDF/OWL level, and this is reflected in the description of its implementation there.

Besides this report, the deliverable contains the current state of the two prototypes, $\boldsymbol{r^3}$ and MARS, both freely available from, respectively, `http://rewerse.net/I5/r3/` and `http://www.dbis.informatik.uni-goettingen.de/MARS/`. In these web sites, one can find more documentation of these prototypes and their usage, including online demonstrators of the prototypes with several use-case examples. This report describes in more detail the $\boldsymbol{r^3}$ prototype, leaving the more complete description of MARS to the next and final deliverable report, on pre-standardization.

In the interest of self-containment of the present report chapter 2 starts with a recap of the $\boldsymbol{r^3}$ ontology description (section 2.1) previously presented in [7], and proceeds with the description of the $\boldsymbol{r^3}$ prototype, library and languages (section 2.2). The report includes an substantial number of examples in order to illustrate the usage of $\boldsymbol{r^3}$, and help outside users.

The ontology description is presented in its most up-to-date state that includes only minor changes. These minor changes, well pointed out in the text and figures, are related to MARS/ECA-ML compatibility issues. The reader familiar with [7] may skip section 2.1 entirely. For the sake of readability, we have chosen here to completely define the current state of the $\boldsymbol{r^3}$ ontology using UML 2.0 [31] diagrams (figures 2.1 to 2.12) and to illustrate it with examples written using Turtle [12] notation[1]. Nonetheless, the normative definition used by $\boldsymbol{r^3}$

[1]For the sake of simplicity, the included Turtle examples omit most of the `@prefix` declarations, and they all

is an OWL-DL [45] ontology (`http://rewerse.net/I5/NS/2006/r3`), and currently r^3 only supports XML-based serializations (e.g. RDF/XML [49]).

Chapter 3 further complements the description in the first chapter with realistic use-case scenarios and concrete examples of r^3 usage, and we terminate with some comments about the future of r^3 in chapter 4.

1.2 On the current prototypes for evolution and reactivity

The goal of the Semantic Web is to bridge the heterogeneity of data formats and languages and provide unified view(s) of the Web. In this scenario, XML (as a format for storing and exchanging data), RDF (as an open abstract data model), OWL (as an additional logic model), and XML-based communication (using different protocols, e.g. pure HTTP or SOAP/WSDL, preferably adhering to a REST architectural style) provide the natural underlying concepts.

The Semantic Web does not have any central structure, neither topologically nor thematically, but is based on peer-to-peer communication between autonomous, and autonomously developing, nodes. Furthermore, the Semantic Web should be able not only to support querying, but also to propagate knowledge and changes in a semantic way. This *evolution* and *behavior* depends on the cooperation of nodes. In the same way as the main driving force for RDF and the Semantic Web idea was the heterogeneity and incompleteness of the underlying data, the heterogeneity of concepts for expressing behavior requires for an appropriate handling on the semantic level. Since the contributing nodes are prospectively based on different concepts such as data models and languages, it is important that *frameworks* for the Semantic Web are modular, and that the *concepts* and the actual *languages* are independent. Even if we would agree that for querying the current set of "common" standards for particular data/knowledge representations/models (e.g. XQuery for XML vs. SPARQL for RDF) could evolve into a single universal query language, which is doubtful, the concepts for describing and implementing behavior are much more different, due to different needs, and is really unlikely that there will be a unique language for the latter throughout the Web.

Heterogenous Reactivity. Here, *reactivity* and its formalization as *Event-Condition-Action (ECA) rules* provide a suitable common model because they provide a modularization into clean concepts with a well-defined information flow. An important advantage of them is that the *content* of a rule (event, condition, and action specifications) is separated from the *generic semantics* of the ECA rules themselves that provides a well-understood formal semantics: when an event (atomic event or composite event, using some event algebra for the composition) occurs, evaluate a condition (possibly gathering further data via queries, again possibly combined via an algebra of queries), and if the condition is satisfied then execute an action (or a sequence of actions, a program, a transaction, or even start a process). Another important advantage of ECA rules is their loosely coupled inherent nature, which allows for declaratively combining the functionality of different Web Services (providing events and executing actions). ECA rules constitute a generic uniform framework for specifying and implementing communication, local evolution, policies and strategies, and –altogether– global evolution in the Semantic Web.

Previously, in [26, 27] we have proposed an ontology-based approach for describing (reactive) behavior in the Web and evolution of the Web that follows the ECA paradigm. This work also defines a global architecture and general markup principles for a modular framework capable

assume at least the declaration of `@prefix : <http://rewerse.net/I5/NS/2006/r3#>`.

of *composing* languages for events, conditions, and actions by separating the ECA semantics from the underlying semantics of events, conditions and actions. This modularity allows for high flexibility wrt. the heterogeneity of the potential sub-languages, while exploiting and supporting their meta-level *homogeneity* on the way to the Semantic Web. Further details about this previous work are available [3, 4, 26, 27].

Resourceful Reactive Rules. Since the inception of the Semantic Web, rules have always been proposed as one of its upper layers: an ontology-based one. Although much research effort is being targeted upon defining rules for and about ontologies, pragmatical and compatibility issues seem to be guiding the work on modelling the rules themselves. In what concerns the latter, most of the current proposals are based on XML markups and rely on specific abstract syntax. Markup-based approaches, as such, seem to ignore the fact that rules do not only operate on the Semantic Web, but are themselves also part of it. In general, especially if one wants to reason about evolution, (ECA) rules and their components must be communicated between different nodes, and may themselves be subject to being updated. For that, (ECA) rules themselves must be first class citizens of the Semantic Web. This need calls for a foundational ontology for describing (ECA) rules. Such an ontology, according to the heterogeneity requirement previously presented, must provide also the means (viz. terms) to describe different languages to be used at the rule component level. As such, we make two important assumptions: first, in the Semantic Web, rules are resources like everything else, and secondly, there won't be such a thing as a (concrete) universal rule language, particularly in what concerns ECA rule components. Given these two hypotheses, here we take a third hypothesis: ontologies (viz. OWL-DL, eventually OWL-Full) provide a suitable tool for describing language heterogeneity.

WG I5 Prototypes. Although the examples included in [26] used "syntactical" languages in XML term markup – ECA-ML – to describe ECA rule components, an OWL ontology has been presented in [7], thus leading to fully embrace the approach proposed in [27]. To further experiment with both approaches, namely syntactic and semantic, the two above mentioned REWERSE WG I5 sub-projects, aiming at developing prototypes of the proposed general ECA framework, were launched: MARS [14, 44] at the markup level and $\boldsymbol{r^3}$ [1, 47] at the ontology level. Currently, both prototypes are functional [5] and, eventually they'll be integrated with each other on a semantic/ontology level.

The work towards taking MARS to an ontology level is on the way, taking an OWL-Full (cf. [7]) approach complementary to $\boldsymbol{r^3}$. As already mentioned, the final state of MARS will be described in the next deliverable. Another WG I5 prototype exists: XChange [18]. It precedes and complements both MARS and $\boldsymbol{r^3}$ and continues to evolve (e.g. [19]) under its fully homogenous and markup based approach. This proprietary, purely syntactical approach doesn't doesn't qualify it for inclusion in the present report which is dedicated to prototypes on the RDF/OWL level. Given all this, the focus of this report is the $\boldsymbol{r^3}$ prototype and its OWL-DL foundational ontology.

Related Work. We are not aware of any Web rule engine (prototype or not) that works on an ontology level like $\boldsymbol{r^3}$ does, or like MARS will do. To the best of our knowledge there are only three ontology proposals for describing rules, besides MARS and $\boldsymbol{r^3}$: SWRL [23], WRL [11] and SBVR [32]. Loosely speaking, the rules modelled by SWRL and WRL are Horn rules; none of the two includes any form of reactive rules. SWRL provides an OWL (Full) ontology;

WRL includes a mapping to OWL-DL (but only at the core level that does not include rules). Although following different approaches, both proposals "extend" OWL by providing means to express OWL-DL axioms. On the other hand, SBVR, which is not formalized in OWL terms, does not exclude reactive rules (some illustrative examples are even present in the specification); but it explicitly chooses not to address its specificities, postponing such matters for reevaluation upon OMG's BPDM [30] results. About SBVR, it is worth mentioning, that it is targeted to describe business rules in general with an emphasis on human understanding [34] (which may hinder machine computability) and it is the only one of the three that actually addresses the issue of language heterogeneity (introducing the concept of business vocabularies, as a form of controlled natural language). Nonetheless, it must be stressed that, SBVR is the only one of these three that does not include a formalization of its semantics.

Most current standardization efforts related to rule interchange, e.g. [17, 16], by following a markup-oriented approach, tend to be charged with syntactical details without semantic value, which has a negative impact on any attempt to raise them to the ontology level. Concrete syntax is usually expressed in terms of abstract syntax, not the other way around. Nevertheless, one of such efforts has to be mentioned even if it does not include any Semantic Web transparent proposal: Common Logic (CL) [24], in what concerns language heterogeneity, is probably the standardization effort closest to the spirit of $\boldsymbol{r^3}$. CL achieves semantics formalization in face of language heterogeneity by limiting its family of languages to those that (and we quote) *have declarative semantics* and *are logically comprehensive*, i.e. *it is possible to understand the meaning of expressions in these languages without appeal to an interpreter for manipulating those expressions* and, *at its most general, they provide for the expression of arbitrary first-order logical sentences*. Given the state of the art, this limitation actually excludes most forms of reactive rules from CL.

Chapter 2

The r^3 Prototype - v0.20

Resources provide the foundation for the Semantic Web. Ultimately, in the Semantic Web everything will be described in resource terms (RDF). For rules to be first class citizens in the Semantic Web, they should also be described at the RDF level. This approach allows the manipulation of rules as pure semantic objects, (re-)using other Semantic Web technologies, not only to define rules, but, more importantly, to reason about rules, leading to a truly adaptive Semantic Web.

Semantic Web rules and rule engines should go semantic all the way. Rule engines have to embrace the ontology layer, even when designing their interface, quite similarly to (or even better, applying if possible) the results achieved by the Semantic Web Services community. Furthermore, the ontology layer should also be used to describe the internal state of rule engines; allowing, for instance, to check the consistency of different representations for the same resource (e.g. different instances of the "same" rule, as "held" by different rule engines). The ultimate goal would be to realize the concept of *Reliable (Reactive) Rules*. Defining a foundational ontology and realizing *Resourceful Reactive Rules* is just a first step in a long way yet to be trailed towards that final goal.

Also, rule engines cannot aim at doing everything by themselves, they have to become really Web-oriented, taking an open and collaborative perspective when evaluating different rule components (in cases where such different components exist, like the case of the reactive ECA rules considered in this work). The incompleteness concept is core to the Semantic Web and it applies not only to data but also to behavior. Rule engines must focus on providing the inference mechanisms, evaluating (different types of) rules, towards drawing conclusions (or performing actions) based on given conditions (or event occurrences), regardless of what the specific rule components (conditions, events, conclusions or actions) really are/mean. The semantic interpretation of rule components must be an open matter, for which a main rule engine relies on other sub-engines. In most cases these "sub-engines" will be domain/application-specific, but generic inference engines, rule-based or not, are not to be excluded.

What is r^3? r^3 stands for "*Resourceful Reactive Rules*", it is a REWERSE WG I5[1] sub-project [47] being developed by CENTRIA in Lisbon, and aims at building a functional research prototype implementing the concepts originally proposed in [6], thus trying to fulfill (and at

[1] http://rewerse.net/I5/

the same time evaluate and experiment) the reactive approach to evolution on the Semantic Web as stated in the goals of REWERSE WG I5 (and initially outlined in [28]).

Two major requirements were previously identified in order to realize this open/semantic perspective on reactivity and evolution for the Web: taking an ontology/resource-based perspective (as detailed in [27]), and dealing with language heterogeneity at the rule component level (as explained in [26]). These requirements led to the specification of a model [3] for a general framework that $\boldsymbol{r^3}$ aims to bring to life.

In more concrete terms, $\boldsymbol{r^3}$ is a prototype of a (Semantic) Web Rule Engine for Reactive Rules. Reactive rules have the general form "*on Event if Condition do Action*", and are also known as: ECA rules, triggers, or active rules. They are intuitively easy to understand, viz. when an event (atomic or composite) occurs, evaluate a condition, and if the condition is satisfied then execute an (atomic or complex) action. This prototype is capable of dealing with reactive rules that use different languages either at the rule component level (event, condition, action), or within each component (by algebraic composition, based also on different algebraic languages).

Component languages may range from general purpose languages (with high flexibility, but limited semantic transparency/value), to domain or application-specific languages (with much richer semantic value). For instance, if a language-specific to the currency trading domain exists, an event stating that "`the exchange rate between Euro and US$ has changed`" may be specified using exactly these terms; for sure in any such language, the precise meaning of this statement is well-known. Otherwise, if no such domain-specific language is available, a general purpose language like XChange [18] (viz. $\boldsymbol{XChange^{EQ}}$ [19]) may be used instead, namely by specifying: an XML pattern to be applied to every incoming event message originated from the addresses of the International Monetary Fund or of the Federal Reserve Bank of New York (assuming these addresses and the actual XML markups used are all known). Of course, in the latter case, one would have to ensure (manually in the rule itself) that such a low level specification is (and will be kept) consistent with the currency market common sense; whereas in the former case, the semantic consistency of the rule is intrinsic to the domain-specific language used.

Why Resourceful? $\boldsymbol{r^3}$ fully embraces the Semantic Web by taking an ontology/resource-based perspective on reactive rules. At the heart of the $\boldsymbol{r^3}$ prototype is the decision to fully base its implementation on an RDF [48] model. Every resource that matters to an $\boldsymbol{r^3}$ engine (e.g. rules) is to be described in terms of an OWL-DL foundational ontology: the $\boldsymbol{r^3}$ ontology.

$\boldsymbol{r^3}$ sees Rules as First Class Citizens of the Semantic Web, allowing research results from other areas, related to the Semantic Web, to be applied also to rules. For instance, it becomes possible to reason about rules by defining rule ontologies and by stating rules about rules, eventually leading to rule evaluation policies and adaptive behaviour.

The different components of each reactive rule are specified (and composed) using different component languages. Each of those languages is to be implemented by specific expression sub-engines (all of them, languages and engines, also described using the terminology provided by the $\boldsymbol{r^3}$ ontology). For an $\boldsymbol{r^3}$ engine, everything is a resource, even rule and rule component evaluation instances are to be represented as resources (for these an adequate extension of the $\boldsymbol{r^3}$ ontology is being defined).

Natively $\boldsymbol{r^3}$ "talks" RDF/XML [49] (using HTTP POST for communication, SOAP [52] wrapped or not), but any other XML serialization (concrete markup) of an RDF model is acceptable, provided an appropriate (bi-directional) translator is available. Any request received

by an r^3 engine is expected to be translated into an RDF model based on the r^3 ontology. This model is then added to an internal ontology that includes every resource known to a particular r^3 engine. This internal ontology must also be dynamically completed by means of automatic searching and fetching missing pieces of information (resource representations) directly from the (Semantic) Web (already identified, in [15], as an important issue, yet to be supported by r^3).

Notice that besides adding resources/models to this internal ontology (e.g. loading rules), which already introduces consistency matters, it is also possible to remove resources/models (e.g. unloading rules), which clearly involves non-monotonic matters. Also when we talk about merging ontologies that contain rules, this merging may lead to updating rules and evolving rule-bases.

If it is true that this fully resource-based (or down-to-earth, *resource-full*) approach used by r^3 opens many research issues (some of them yet untouched in what concerns the Semantic Web), it is also true that every research issue opened by r^3 (given its modular architecture) constitutes an opportunity for scientific cooperation with other research communities, paving the way to more *Resourceful* solutions.

Chapter Structure. This chapter is divided into 2 sections, the first dedicated to the foundational components of the r^3 ontology, and the second related to details of a more pragmatical nature both of the ontology and the prototype.

2.1 The r^3 Foundational Ontology

In this section we present current results of the r^3 project towards defining an (OWL-DL) foundational ontology for reactive rules. The current proposal is at a low (structural) abstraction level; the extension of this proposal towards characterizing higher abstraction level concepts, like domain/application specific languages (vs. algebraic and general-purpose languages) is also being considered. Although focusing on reactive rules, the r^3 ontology defines a vocabulary that allows for the definition of rule (component) languages that can be extended to model other types of rules.

We start (in section 2.1.1) by introducing the r^3 ontology from a global point of view[2]. In the following sections we detail the r^3 ontology explaining (and illustrating[3]) how to define different languages (in section 2.1.2), and how to use these languages to define heterogenous rules (in section 2.1.3). The definition of the r^3 ontology is further complemented (in section 2.1.4) by sketching how the ontology can be used to provide additional semantic value to heterogeneous XML markups. We do this by describing how an RDF model, compliant with the r^3 ontology, may be automatically obtained from an XML markup adhering to the previously proposed general markup principles (viz. ECA-ML). We end this section with some future directions of the presented work.

The complete r^3 ontology (`http://rewerse.net/I5/NS/2006/r3`, v0.20), as well as the implementation and demonstration of r^3 prototype can be found at `http://rewerse.net/I5/r3/`.

[2]The r^3 OWL-DL ontology is presented here using UML2 diagrams for the sake of readability.

[3]Examples illustrating RDF models: use N3 notation, omit prefix declarations, and assume r^3 as the empty(':') prefix.

2.1.1 An Ontology for Reactive Rules

From a global point of view, the r^3 ontology defines a vocabulary to describe `RulePackage`s[4]. A `RulePackage` is just a single `Rule` or a `RuleSet`, as depicted in figure 2.1. In order to cope with the language heterogeneity requirements previously stated, the r^3 ontology includes a meta-language/vocabulary to describe rule (component) languages/vocabularies. This meta-level assumes a term based structure for rule languages where atomic functors are composed using algebraic operators. To account for languages not compliant with this term structure (like purely textual languages, e.g. database triggers, external program call for action, etc) opaque/native "functors" are also allowed.

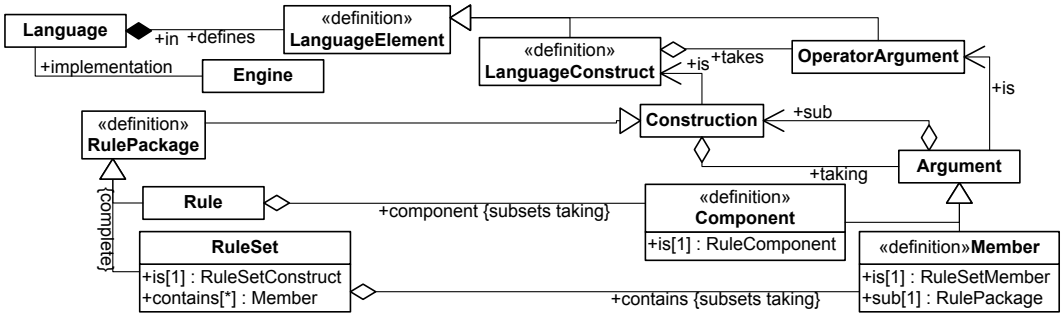

Figure 2.1: Rule Packages

(changes since I5-D6: none)

In terms of this meta-vocabulary, a `RulePackage` is a language `Construction` (exclusively a `Rule` or a `RuleSet`) `taking` a set of `Argument`s (resp. `Rule Component`s or `RuleSet Member`s). The complete definition of a `Construction` is to be provided in section 2.1.3, but for now, cf. figure 2.1, a `Construction is` a `LanguageConstruct taking` a set of `Argument`s, in accordance with its `LanguageConstruct` that `takes` a set of `OperatorArgument`s (non-empty, in case of an algebraic operator). An `Argument is` a specific `OperatorArgument` including a `sub-Construction`.

`LanguageConstruct`s and `OperatorArgument`s constitute some of the specialized forms of `LanguageElement`s defined `in` a `Language`. The actual operational `implementation` of (the elements of) a `Language` is provided by some `Engine`.

A `RuleSet is` a `RuleSetConstruct` that recursively `contains` other `RulePackage`s (viz. `Member is` a `RuleSetMember` including a `sub-RulePackage`). A `Rule`, as further detailed in figure 2.2, is either a `NativeRule` defined by a mandatory (and usually textual) opaque `source`; or an `AbstractRule`, fully defined by a non-empty set of `Component`s (where a `Component is` a `RuleComponent`).

The opaque `Source` of a `NativeRule` is neither a `Construction` nor an `Argument`. Instead, it is defined as a `Parameter` (one that `is` a `RuleSource`). This leads to an extension of the `LanguageConstruct` concept as being a parametric construct. Any `LanguageConstruct has` an (optionally empty) set of `ConstructParameter`s, another kind of `LanguageElement`s. As such, the language `Construction` concept, besides `taking` a set of `Argument`s, is further refined, in figure 2.2, as `having` a set of `Parameter`s, where a `Parameter is` a `ConstructParameter`. The full definition

[4]For the sake of conciseness, we use generic terms, like "Rule" or "Language", whereas more specific terms, like "Reactive Rule" or "Reactive Rule Language", would have been more consistent with the actual motivation of the presented work. As such, it is worth stressing here that the adequacy of the r^3 ontology to describe rules in general, although not excluded, has not been properly evaluated and is, at the time of this writing, an open matter.

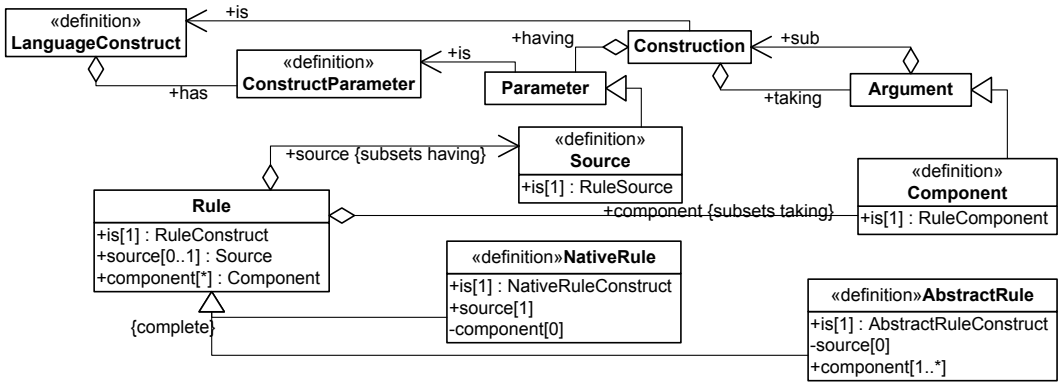

Figure 2.2: Rules

(changes since I5-D6: none)

of an r^3 `Language` is presented in the next section, followed by the complete definition of an heterogenous `Construction` in section 2.1.3.

2.1.2 Defining Reactive Rule Languages

In order to define the different `Language`s involved in a `RulePackage`, the r^3 ontology provides a meta-vocabulary, depicted in figure 2.3 at its core level.

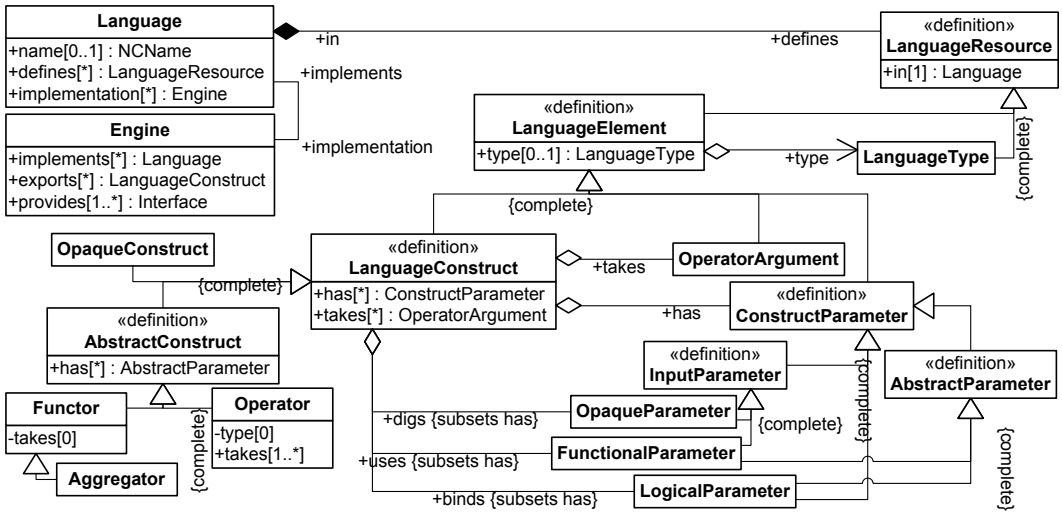

Figure 2.3: Languages

(changes since I5-D6: added `Language.name` and `Engine.exports` mainly for compatibility with MARS)

Some of the more generic terms/concepts (e.g. `Language`, `LanguageElement`) are pretty straightforward and have already been introduced before. Nevertheless, it is worth mentioning that a `Language` actually `defines` a set of `LanguageResource`s (`LanguageElement`s or `LanguageType`s, defined `in` the `Language` itself); and that a `LanguageElement` has to be either a `LanguageConstruct`, a `ConstructParameter` or an `OperatorArgument`.

A LanguageConstruct is further distinguished between an OpaqueConstruct and an AbstractConstruct (one that has only AbstractParameters), the latter being either an algebraic Operator (if it takes at least one OperatorArgument) or a Functor[5].

The set of ConstructParameters a LanguageConstruct has, may be explicitly divided according to the specific nature of each parameter: a LanguageConstruct uses (or digs) InputParameters and binds LogicalParameters. The semantics of a LanguageConstruct is not defined unless the actual value of all its InputParameters is known. Example 2.1.1 further illustrates the concept of an r^3 Language partially presenting the definition of some Languages.

Example 2.1.1 *Below we present an r^3 partial definition of several rule component languages, that will be further exploited in subsequent examples. Namely, an event algebra (snoop), two domain/application languages (travel and rental), and two general-purpose libraries (mail and text).*

```
snoop:sequence a :Operator; :in snoop:event-algebra;
  :takes snoop:first, snoop:other.
travel:booking-place a :Functor; :in travel:domain;
  :binds travel:client, travel:flightnr, travel:seat.
travel:flight-info a :Functor; :in travel:domain;
  :uses travel:flightnr;
  :binds travel:date, travel:origin, travel:destination.
rental:request-quotation-for-flight a :Functor; :in rental:application
  :uses rental:client, rental:flightnr.
rental:get-client a :Functor; :in rental:application
  :uses rental:client;
  :binds rental:client-name, rental:favorite-class, rental:max-price.
rental:get-available-cars a :Functor; :in rental:application
  :uses rental:office, rental:date;
  :binds rental:car, rental:car-class, rental:price.
mail:send a :Functor; :in mail:library;
  :uses mail:from, mail:to, mail:subject, mail:body.
text:join a :Aggregator; :in text:library;
  :uses text:template, text:separator.
text:replace a :Aggregator; :in text:library;
  :uses text:template.
```

An InputParameter may first be seen as a purely FunctionalParameter, whose (atomic) values, the LanguageConstruct transparently uses to define its actual semantics. However, careful observation reveals that some of these parameters may hide semantic value (e.g. variable references to be expanded) by using "internal" (sub-)languages. In these cases, the full semantics of the construction is not accessible without full knowledge of the (sub-)languages actually used inside the (consequently, non-atomic) values of those parameters. Such parameters are called OpaqueParameters and any LanguageConstruct that has (or, more precisely, digs) at least one of such OpaqueParameters must be an OpaqueConstruct (since it cannot be a semantically transparent AbstractConstruct).

The semantics of a Construction that is an OpaqueConstruct is known only to an Engine that implements the associated Language. Most of the times these Engines will only define the semantics of such an OpaqueConstruct in operational terms, meaning that the only form of knowing it is to submit (at runtime) an actual Construction to the evaluation Interface that the Engine provides. Nevertheless, static analysis may some times be possible as long as the

[5]Also distinguishable, among Functors, are Aggregators which provide the means to construct arbitrary variable Aggregations, as a specialized form of Constructions, cf. section 2.1.3.

`Engine` `provides` a translation `Interface` for parsing such an opaque `Construction` into one that `is` an `AbstractConstruct`[6].

In general, the definition of a `LanguageElement` may include a `type`. The exceptions to this are `Operator`s and `OperatorArgument`s, cf. figure 2.5, given their required algebraic nature. A `LanguageType` is considered here only as some resource that implicitly defines a domain of literal values (having a lexical space compatible with rdf:XMLLiteral) and may refer, for instance, to a particular XML Schema (element or type, basic or not). The actual treatment of types, in the context of a general framework like [26], has no yet been studied, but it may provide, for instance, the means for a safer equality relation (e.g. xml:space preserving or not). As such, the concept of a `LanguageType` is already included here providing the means to define, among others[7], the domain of a `ConstructParameter` or the range of a `Functor`.

The r^3 meta-vocabulary for defining languages (given its core level, in figure 2.3) is extended in figure 2.4 with additional `LanguageConstruct`s specific to rule languages, viz. `RulePackageConstruct`s, which are clearly distinguished from other constructs, viz. `ExpressionConstruct`s.

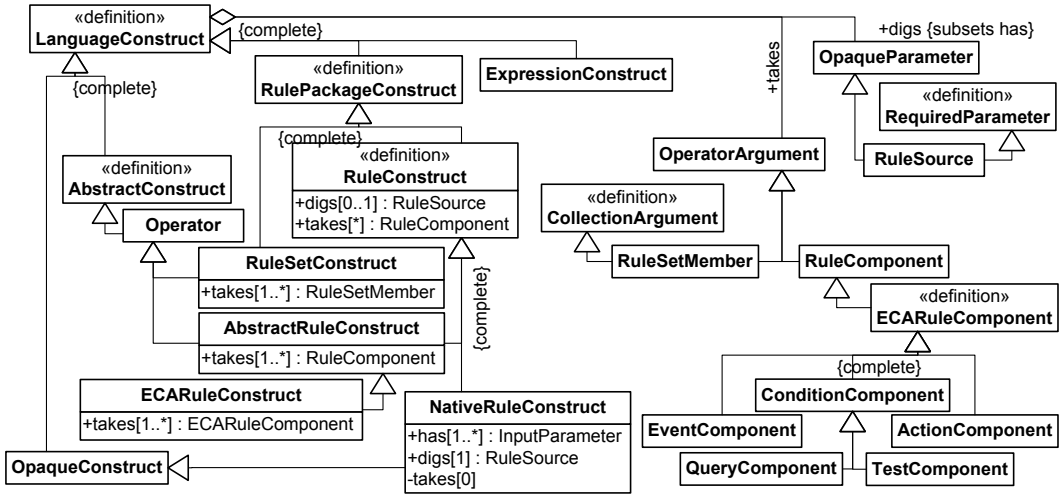

Figure 2.4: Rule Languages

(changes since I5-D6: none)

`ECARuleConstruct`s, which are particularly relevant to our work at this rule level, provide, together with specific `ECARuleComponent`s, the means for constructing ECA rules. Also relevant are `NativeRuleConstruct`s that allow modelling ECA rules at an opaque level (e.g. database triggers). Notice that a `NativeRuleConstruct`, besides its `RuleSource` (that it `digs`), additionally `has` other `InputParameter`s (if needed, as is usually the case of e.g. database name and user in database triggers).

On a general level, most practical language definitions include some forms of syntactic sugar. Among the most common forms are optional/multiple parameters/arguments (e.g. in figure 2.4 a `RuleSetMember` is defined as a `CollectionArgument`). These forms of syntactic sugar very often hide major semantic decisions. In an eclectic heterogenous context like the one proposed, the

[6]The concept of an `Engine` which `implements` a `Language` is core to the general ECA framework and the r^3 prototype, further details about how this concept helps realize the current version of the r^3 prototype may be found in 2.2.

[7]`LanguageType`s are also used to constrain the domain of logical variables and the range of expressions, as explained later in section 2.1.3.

"sweetness" of these syntactic forms may be misleading and hinder the semantic understanding of a set of constructions. For instance, an ECA rule may be defined as taking several condition or action components, but the actual meaning of such constructions remains unknown or opaque unless a particular composer operator is explicitly chosen and identified. Different ECA languages may follow different directions in this respect: usually a simple conjunction is assumed for conditions, but the choice becomes a bit more fuzzy for actions, particularly if transactions are considered, and things get even more involved if multiple event components are considered (where the natural choice would be to interpret them as a disjunction). Multiple/optional parameters/arguments are not actually allowed in a `LanguageConstruct`. Nevertheless appropriate extension of the r^3 ontology (as presented in figure 2.5) may provide enough expressivity for the most "annoying" situations without disregarding semantic transparency.

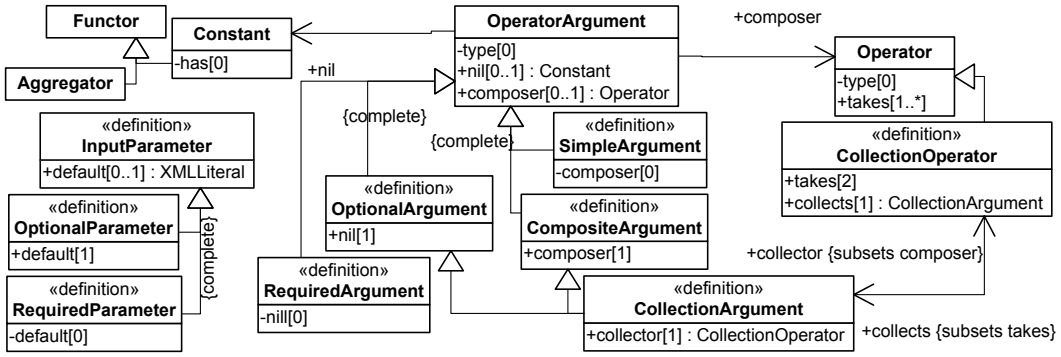

Figure 2.5: Language (not so Syntactical) Details

(changes since I5-D6: none)

Optional parameters (and arguments) are modelled as usual with the introduction of `default` (and `nil`) constants that are to be used if the parameter (or argument) is omitted. For the moment, multiple parameters are not accounted for (assuming that a parameter value may as easily be a single value or a list of values using appropriate markups). Multiple arguments, as `CollectionArgument`s, are considered a specialization of a more general concept: `CompositeArgument`s. A `CompositeArgument`, if present, is always built using a specific `composer Operator` and most often it induces syntactic simplifications when defining markup languages (i.e. the `CompositeArgument` or its `composer` are omitted, as long as ambiguity issues are not introduced). A `CollectionArgument` (named after rdf:parseType="Collection") has a recursive nature given by its `collector`, which, as a binary associative `CollectionOperator`, `takes` one occurrence and further `collects` additional occurrences of the `CollectionArgument` itself (terminated by a required `nil Constant`, which stands for the empty collection if there are no further occurrences of it). To further illustrate some of these issues, consider example 2.1.2 where a definition of the ECA-ML `Language` (viz. `eca:ml`) is presented.

Example 2.1.2 *The* `eca:ml Language` *is defined in* r^3 *as:*

```
eca:rule a :ECARuleConstruct; :in eca:ml;
  :takes eca:event, eca:condition, eca:action.
eca:event a :EventComponent; :in eca:ml.
eca:condition a :ConditionComponent; :in eca:ml;
  :composer eca:collect-and-test; :nil eca:true.
eca:collect-and-test a :Operator; :in eca:ml;
  :takes eca:collect, eca:test.
eca:collect a :CollectionArgument; :in eca:ml;
```

```
  :collector eca:query-collect; :nil eca:true.
eca:query-collect a :CollectionOperator; :in eca:ml;
  :takes eca:query, eca:collect.
eca:query a :QueryComponent; :in eca:ml; :nil eca:true.
eca:test a :TestComponent; :in eca:ml; :nil eca:true.
eca:launch a :CollectionArgument; :in eca:ml;
  :collector eca:launch-all; :nil eca:nop.
eca:launch-all a :CollectionOperator; :in eca:ml;
  :takes eca:action, eca:launch.
eca:action a :ActionComponent; :in eca:ml; :nil eca:nop.
eca:native a :NativeRuleConstruct; :in eca:ml;
  :uses eca:lang; :digs eca:source.
eca:opaque a :ExpressionConstruct; :in eca:ml;
  :uses eca:lang; :digs eca:literal.
```

Some of the `LanguageElement`s, included in example 2.1.2, come directly from the actual ECA-ML markup and are self-explanatory, nevertheless those that don't, are worth mentioning. In the previously proposed ECA-ML markup the `eca:condition` "tree" is not explicitly included since the *collect and test composition* and the multiple *query collection* are always implicitly present and no syntactical ambiguity results from omitting them. As such, only the "leaf" arguments `eca:query` and `eca:test` were actually included in the ECA-ML markup (and the same applies to `eca:action`). Furthermore, the ECA-ML markup does not include any `eca:native` element, instead it chooses to re-use a pair of other elements (`eca:rule` and `eca:opaque`). This pair of elements has no meaning in $\boldsymbol{r^3}$ terms. In fact, the ECA-ML markup has chosen that syntactical form (of a pair) precisely because it bears no other meaning, and overloading these elements avoided the inclusion of a new term (e.g. `eca:native`). This is a good example of how a markup-based approach (even a striped one, favoring RDF/XML) differs from an ontology-based approach. The rationale behind the chosen terms for the vocabulary is simply different. Both approaches reject ambiguity, but in the former, minimizing verbosity and the number of vocabulary terms are driving forces (as long as readability and "syntactical elegance" are kept), whereas in the latter overloading of terms, without proper specialization of concepts, is to be avoided since it reduces the semantic value of the vocabulary. Both approaches are pertinent, but when defining new vocabularies for the Semantic Web, it is our stance that the former should not influence the latter, but the other way around: ontology-based vocabularies should influence and provide an appropriate abstract syntax for concrete markups, which may be improved, later, towards syntactical conciseness and elegance (although an XML markup is hardly the better serialization choice for those purposes).

As a final note about defining $\boldsymbol{r^3}$ `Language`s, it is worth stressing that strict conformance to OWL-DL must be observed at all times. The `Language` meta-level of the $\boldsymbol{r^3}$ ontology may lead to (sometimes tempting) misuses that may introduce undecidability issues. For instance, an ontology that imports the $\boldsymbol{r^3}$ ontology and defines a particular `Language`, must do just that: define a resource that is a `Language` individual. Most frequently, such an ontology will define a set of terms (e.g. individual `LanguageElement`s) in the ontology namespace. This namespace, as a URI reference, refers to *the ontology* itself which *is not* the `Language`, but instead *defines* the `Language` (which in turn, `defines`, e.g., its `LanguageElement`s). Such an ontology (or its namespace, e.g. `eca:`) should never be mistaken for the `Language` itself (e.g. `eca:ml`). Another example of such (tempting) misuses, that must be avoided, is to define an OWL class as being a `LanguageType` individual, which is not a valid OWL-DL statement (one possible way around this would be to extend the `LanguageType` class with appropriate data-type properties).

2.1.3 Defining Reactive Rule Constructions

Given a set of `Language`s defined using the meta-vocabulary presented in the previous section, it is possible to define the general concept of an heterogenous `CodingElement` (a `Construction`, a `Parameter` or an `Argument`) which `is` a `LanguageElement` (respectively, a `LanguageConstruct`, a `ConstructParameter` or an `OperatorArgument`). The definition of a `CodingElement` (depicted in figure 2.6) follows a structure that closely mimics the structure of `Language`s, previously presented in figure 2.3).

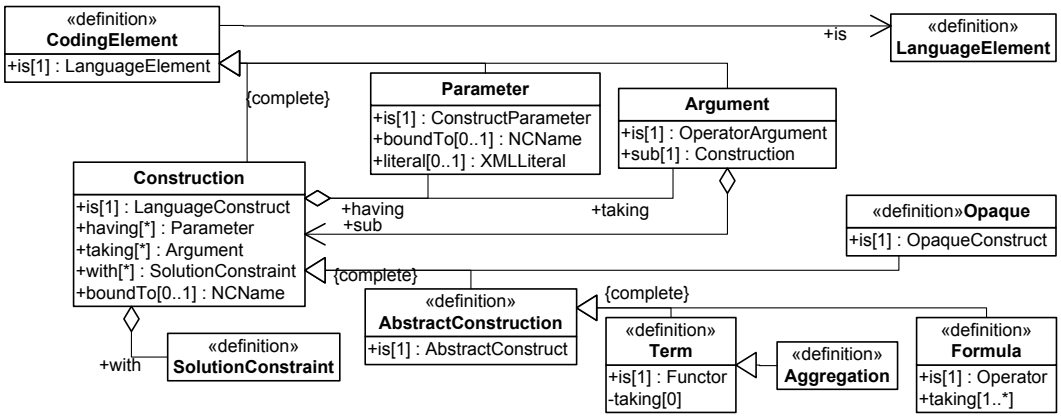

Figure 2.6: Language Constructions

(changes since I5-D6: none)

A `Construction` may be seen as an instance of a `LanguageConstruct`, `having` a set of `Parameter`s possibly with specific `literal` values (or `boundTo` a particular variable), and `taking` a concrete set of `sub-Construction`s as `Argument`s. Example 2.1.3 illustrates this concept with several concrete `Construction` instances.

Example 2.1.3 *Find below some rule component constructions using the languages in example 2.1.1.*

```
ex:event :is snoop:sequence;
  :taking [:is snoop:first; :sub [
    :is travel:booking-place;
    :having [:is travel:client; :boundTo "Mail"];
    :having [:is travel:flightnr; :boundTo "Flight"]]];
  :taking [:is snoop:other; :sub [
    :is rental:request-quotation-for-flight;
    :having [:is rental:client; :boundTo "Mail"];
    :having [:is rental:flightnr; :boundTo "Flight"]]].
ex:get-client :is rental:get-client;
  :having [:is rental:client; :boundTo "Mail"];
  :having [:is rental:client-name; :boundTo "Client"];
  :having [:is rental:favorite-class; :boundTo "Class"];
  :having [:is rental:max-price; :boundTo "Max-Price"].
ex:get-available-cars :is rental:get-available-cars;
  :having [:is rental:office; :boundTo "To"];
  :having [:is rental:date; :boundTo "Date"];
  :having [:is rental:car; :boundTo "Car"];
  :having [:is rental:car-class; :boundTo "Class"];
  :having [:is rental:price; :boundTo "Price"].
ex:flight-info :is travel:flight-info;
```

```
:having [:is travel:flightnr; :boundTo "Flight"];
:having [:is travel:date; :boundTo "Date"];
:having [:is travel:origin; :boundTo "From"];
:having [:is travel:destination; :boundTo "To"].
```

Upon evaluation by an appropriate `Engine`, a `Construction` returns a set of results (given a set of input substitutions). Each returned result contains an optional literal value and a set of output substitutions (that must be joined to one or more of the input substitutions). The returned literal values can only be used as argument values of an algebraic operator, provided the `Construction` is actually a `sub-Construction` of a `Formula` (abstract or "disguised" as an `Opaque`), unless they are `boundTo` some variable (thus extending the output substitutions).

A `Construction` may further restrict its results `with` a set of `SolutionConstraint`s (cf. figure 2.7), each defining a *constraint domain* used to constrain the *constraint range*. The constraint range may be either the actual set of values bound to a particular variable (identified by its `name` in a `VariableDeclaration`[8]), or the set of values returned by a `TestExpression` involving several variables (where an `Expression` is understood as a `Construction` that does not contain any form of `RulePackage`, cf. figures 2.4 and 2.7).

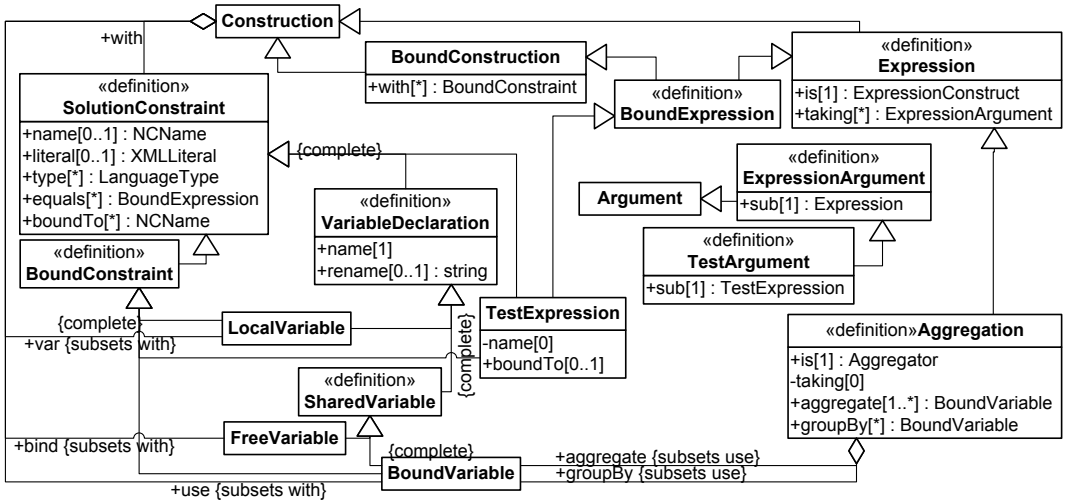

Figure 2.7: Constraints, Variables and Expressions

(changes since I5-D6: none)

Any `Construction` defines a scope where `LocalVariable`s may be declared. The communication between a `sub-Construction` and its ancestor/sibling `Construction`s is achieved through the `SharedVariable`s it either `use`s or `bind`s. A `Construction` cannot be evaluated unless all the variables it `use`s are actually bound to specific values.

If a `Construction` is `boundTo` a variable; that variable, if not explicitly declared, is implicitly declared as a `FreeVariable`, i.e. one that the `Construction` may `bind`. If a `Construction having` a `Parameter boundTo` a variable does not explicitly declare it, the variable is implicitly declared a `SharedVariable` according to the nature of the `Parameter`: a `FreeVariable` if it `is` a `LogicalParameter` that the `Construction` may `bind`, or a `BoundVariable` if it `is` an `InputParameter` that the `Construction` must `use`. A `Construction` implicitly inherits all the

[8] A `VariableDeclaration` may additionally `rename` its variable, in order to support different naming conventions used by different `Opaque` languages.

SharedVariables (implicit or explicit) of the sub-Constructions of its Arguments. All variables in a BoundConstruction must be explicitly declared with BoundConstraints, thus excluding the declaration of FreeVariables. Namely a BoundConstruction cannot bind SharedVariables.

A SolutionConstraint states that the intersection between the constraint domain and the constraint range (the actual set of values of a variable, for a VariableDeclaration, or the actual set of values returned by a TestExpression) must not be empty. If it does not define a constraint domain it simply states that the constraint range must not be empty. The constraint domain of a SolutionConstraint is defined by intersecting several sub-domains, namely a domain with a single literal value, some domains implicitly defined by given types, some domains formed by the values boundTo given variables, or even some domains where each one equals the actual set of values returned by a BoundExpression: a BoundConstruction that is an Expression. Operationally, a SolutionConstraint can only be checked when its constraint range is known (i.e. the variable becomes bound or is possible to evaluate the test expression). If it can be checked then, for efficiency, it must invalidate as much solutions as possible, as soon as possible (i.e. the constraint range must be checked against each of the known constraint sub-domains; this check must not be postponed until the full constraint domain is known).

Example 2.1.4 *Below we illustrate the usage of* SolutionConstraint*s as* VariableDeclaration*s in several* Construction*s (using the* Language*s defined in examples 2.1.1 and 2.1.2).*

```
ex:check-price :is eca:opaque;
  :boundTo "Is-True"; :var [:name "Is-True"; :literal "true"];
    # or simply, cf. :TestExpression, :literal "true";
  :use [:name "Price"];
  :use [:name "Max-Price"; :rename "MaxPrice"];
  :having [:is eca:lang; :literal "http://www.w3.org/XPath"];
  :having [:is eca:literal; :literal "$Price <= $MaxPrice"].
ex:send-quotation :is mail:send;
  :var [:name "Text"; :equals ex:quotation-message];
  :having [:is mail:from; :boundTo "Rental-Mail"];
  :having [:is mail:to; :boundTo "Mail"];
  :having [:is mail:subject; :literal "Car Rental Quotation"];
  :having [:is mail:body; :boundTo "Text"].
ex:quotation-message :is text:replace;
  :use [:name "Quotation" ];
  :var [:name "Priced-Cars";
    :equals [:is text:join;
      :aggregate [:name "Car"], [:name "Price"];
      :having [:is text:template; :literal "|Car|=|Price|"];
      :having [:is text:separator; :literal ", "];]];
  :aggregate
    [:name "Client"], [:name "Priced-Cars"],
    [:name "Flight"], [:name "Date"], [:name "To"];
  :having [:is text:template; :boundTo "Quotation"].
```

To ensure full transparency of an AbstractConstruction, explicit declaration of SharedVariables in these Constructions is restricted and only allowed for those variables already implicitly declared as shared: boundTo the Construction itself or to one of its Parameters, or declared as shared in an Argument. The only form of AbstractConstruction that does not have this restriction is a specialized form of a Term called an Aggregation (that is a purely functional Aggregator), which aggregates some variables grouped by some other variables it uses (groupBy is intended as a synonym of use).

Both the input and output substitutions of any Construction must include only the variables declared in it. Furthermore, input substitutions must be distinct and output substitutions of

an `Aggregation` cannot contain the `aggregated` variables.

Given the definitions contained in the diagrams of the previous figures, an `ECARule` is finally defined as depicted in figure 2.8.

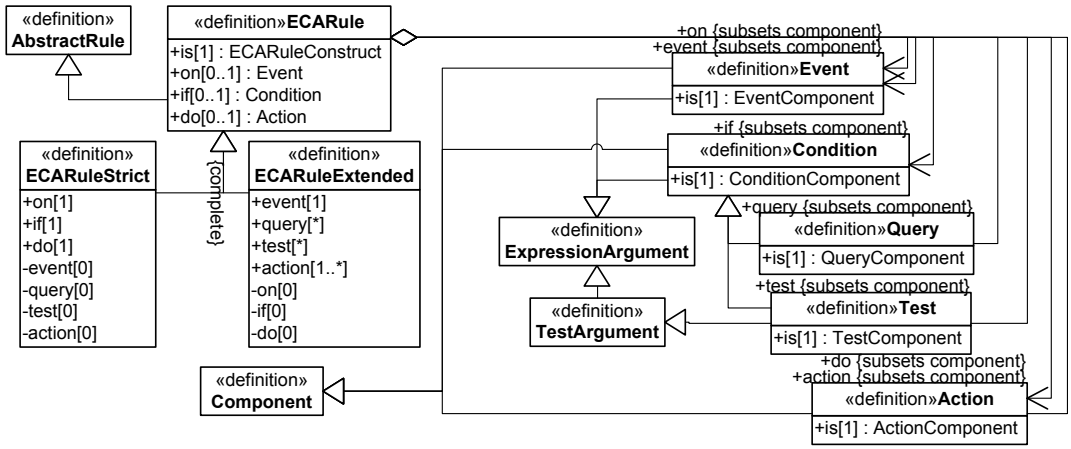

Figure 2.8: ECA Rules

(changes since I5-D6: added `ECARule` partition and `event`/`query`/`test`/`action` sub-properties mainly for compatibility with ECA-ML)

Example 2.1.5 *To further illustrate this definition a concrete* `ECARule` *is presented here, using the* `Language`*s and* `Construction`*s included in the previous examples.*

```
ex:rule :is eca:rule;
  :on [:is eca:event; :sub ex:event];
  :if [:is eca:condition; :sub [
    :is eca:collect-and-test;
    :var [:name "Class"];
    :taking [:is eca:collect;
        :sub [:is eca:query-collect;
      :taking [:is eca:query; :sub ex:get-client];
        :taking [:is eca:collect; :sub [:is eca:query-collect;
      :taking [:is eca:query; :sub ex:get-available-cars];
        :taking [:is eca:collect; :sub [:is eca:query-collect;
      :taking [:is eca:query; :sub ex:flight-info];
        :taking [:is eca:collect; :sub [:is eca:true]]]]]]];
    :taking [:is eca:test; :sub ex:check-price]]];
  :do [:is eca:action; :sub ex:send-quotation].
```

For the time being, the r^3 prototype does not support `CollectionOperator`s, as such in order to easilly support ECA-ML we have chosen to include an `ECARuleExtended` form taking sub-properties that mimic the ECA-ML elements.

2.1.4 From ECA-ML to r^3DF

The original motivation for using an OWL-DL ontology definition as the foundation for r^3 instead of an XML markup (viz. ECA-ML [26]) was that the heterogeneous approach made it quite difficult to fully formalize the XML markup. The level of expressiveness of the available standards (viz. DTD or XML Schema) was not sufficient. Instead of pushing these standards

to the limit (or defining a new abstract syntax) it seemed only natural to resort to an adequate standard like OWL-DL (re-using already defined abstract syntaxes, e.g. RDF, and related concrete markups, e.g. RDF/XML).

To complement the definition of the r^3 ontology just presented, we now informally describe how an RDF model of an `ECARule`, like the one included in example 2.1.5, may be automatically obtained from an XML serialization of an ECA rule. Of course, we do not simply mean RDF/XML serializations. Rather, our aim here is to translate markups that adhere to the ECA-ML proposal [26] (and to its general guidelines for component languages and logical variables). Such an ECA-ML compliant serialization of a rule is shown in example 2.1.6.

Example 2.1.6 *The* `ECARule` *of example 2.1.5 using an ECA-ML serialization:*

```
<eca:rule>
  <eca:event>
    <snoop:sequence>
      <snoop:first>
        <travel:booking-place client="$Mail" flightnr="$Flight"/>
      </snoop:first>
      <snoop:other>
        <rental:request-quotation-for-flight
          client="$Mail" flightnr="$Flight"/>
      </snoop:other>
    </snoop:sequence>
  </eca:event>
  <eca:query>
    <rental:get-client client="$Mail" client-name="$Client"
      favorite-class="$Class" max-price="$Max-Price"/>
  </eca:query><eca:query>
    <travel:flight-info flightnr="$Flight" date="$Date"
      origin="$From" destination="$To"/>
  </eca:query><eca:query>
    <rental:get-available-cars office="$To" date="$Date"
      car="$Car" car-class="$Class" price="$Price"/>
  </eca:query>
  <eca:test eca:variable="Is-True">
    <eca:variable name="Is-True">true</eca:variable>
    <eca:input-variable name="Price"/>
    <eca:input-variable name="Max-Price" use="MaxPrice"/>
    <eca:opaque lang="http://www.w3.org/XPath">
      $Price <= $MaxPrice
    </eca:opaque>
  </eca:test>
  <eca:action>
    <eca:variable name="Text"><eca:query>
      <eca:variable name="Priced-Cars"><eca:query>
        <eca:input-variable name="Car"/>
        <eca:input-variable name="Price"/>
        <text:join template="|Car|=|Price|" separator=", "/>
      </eca:query></eca:variable>
      <text:replace template="$Quotation">
        <eca:input-variable name="Client"/>
        <eca:input-variable name="Priced-Cars"/>
        <eca:input-variable name="Flight"/>
        <eca:input-variable name="Date"/>
        <eca:input-variable name="To"/>
      </text:replace>
    </eca:query></eca:variable>
    <mail:send from="$Rental-Mail" to="$Mail"
      subject="Car Rental Quotation" body="$Text"/>
  </eca:action>
</eca:rule>
```

Given the definition of the `eca:ml Language` provided in example 2.1.2, we immediately identify the root element and its children, in the given ECA-ML example, as `LanguageElements` `in` the `eca:ml Language`: an `Operator`, `eca:rule`, and several `OperatorArguments`. As such, it

should be possible to build a Construction out of it. Two of the children (eca:event and eca:action) are in fact arguments of eca:rule, but there are several eca:query and an eca:test that cannot be "parsed". Furthermore, a third argument is missing: eca:condition. If we look closely at the definition of eca:condition, we see that it is a CompositeArgument which must be built using the particular Operator eca:collect-and-test that takes as OperatorArguments an eca:collect and an eca:test. We do have an eca:test available, so our problem is now narrowed to building an eca:collect out of several eca:query. Another look at the eca:ml definition tells us that eca:collect is also a CompositeArgument (a CollectionArgument), and that its composer (collector) takes an eca:query. So, we just pick one from our queries. The actual order could be relevant, but in fact that is not the case here since a partial order may be inferred, given the added semantic value provided by Language definitions, viz. InputParameters. Once we have picked a query, we are back to our problem of building an eca:collect, only now with one less eca:query. If we continue applying this process we will be reduced to the problem of building an eca:collect, which becomes possible (since it is an OptionalArgument) by simply grabbing its nil constant, eca:true. The result is an ECARule with the structure presented in example 2.1.5 that includes several RuleComponents: an EventComponent, several QueryComponents, a TestComponent, and an ActionComponent. What is still missing is the sub-Construction for each of those RuleComponents.

The same process may be easily applied to obtain the eca:event EventComponent, which leads us to the problem of "parsing" the Terms that relate to the Functors included in it (cf. example 2.1.1, for travel:booking-place and rental:request-quotation-for-flight). As XML elements, these Functors, include attributes without a namespace, which are meaningless in RDF, but nevertheless quite common in current XML markups. To avoid this problem we assume that any attribute without a namespace is to be replaced by an (equally valued) attribute with the same local name but in the namespace of its parent element. Given that, all those attributes may be identified as LanguageElements (viz. ConstructParameters). Furthermore, those Parameters are boundTo variables according to the ECA-ML guidelines, prefixing variables with '$'. Given all this, not only the eca:event EventComponent, but also all the eca:query QueryComponents, may be promptly translated into RDF models containing the sub-Constructions of example 2.1.3.

The eca:test TestComponent and the eca:action ActionComponent both use ECA-ML nodes that are specific to logical variables (i.e. eca:variable and eca:input-variable). Although these nodes belong to the eca: namespace they are not defined as LanguageElements in eca:ml. Dealing with logical variables is orthogonal to language heterogeneity, in fact, logical variables provide the homogeneous "glue" that supports language heterogeneity and are not themselves part of it. Actually, this calls for a markup specific to logical variables, which is out of scope in this paper. So, for now, we stick with the ECA-ML proposal, but we do look at these XML nodes as homogeneous. As such, we map an eca:variable XML attribute to a Construction boundTo a variable; and the eca:variable (and eca:input-variable) XML elements to FreeVariables (and BoundVariables). Each SharedVariable resulting from the latter mapping (notice that ECA-ML does not include LocalVariables): has a name; is optionally renamed (according to its use attribute); and its content (if not empty) defines its value literally or through the (eca:query) sub-Construction that equals it.

Given this (partial) homogeneous mapping for the ECA-ML nodes related to logical variables, both the TestComponent and ActionComponent sub-Constructions are built as the previous RuleComponents (leading to the RDF model included in example 2.1.4), and our ECARule is finally complete and should be recognized as such by an OWL-DL reasoner (given the $\boldsymbol{r^3}$ ontology and the appropriate Languages).

2.2 r^3 Engines and Library

At the heart of the r^3 prototype is the decision to fully base its implementation on an RDF model, fully embracing Semantic Web technologies and standards by taking an ontology/resource-based perspective on reactive rules (to the extent allowed by the actual availability of "stable" prototypes[9]). Every resource that matters to an r^3 engine (e.g. rules) is to be kept in this RDF model and described in terms of the r^3 ontology. As detailed in section 2.1, the different components of each reactive rule are specified (and composed) using different component languages. Each of those languages is to be implemented by specific expression sub-engines, all of them, languages and engines, also described using the terminology provided by the r^3 ontology. For an r^3 engine, everything is a resource, even rule and rule component evaluation instances are to be represented as resources. For these an adequate extension of the r^3 ontology needs to be defined.

Naturally, openness and flexibility (as opposed to efficiency and optimization) are the main drivers for the technical choices behind r^3, nevertheless it is important to correctly abstract shared functionality in order to account as much as possible for future evolution (not excluding optimization). r^3 provides a Java development library intended to help in the development of (generic or domain/application specific) language engines. This library abstracts many of the details needed to realize an engine conforming to the General ECA framework detailed in [3] (e.g. communication/protocols, RDF/XML manipulation, binding variables, joining substitution sets).

In this section we present the current state of the extension of the r^3 ontology in order to include the abstract API of r^3 engines, together with architecture and development details of the r^3 prototype and library. We start by describing the ontology extension that deals with the operations supported by an r^3 engine (section 2.2.1); namely operations to load and evaluate reactive rules (section 2.2.2). This description is further complemented in section 2.2.3 where we detail the lower levels of the ontology that provide the support for dealing with variables, substitutions and results. For the sake of completeness of the presented ontology, in section 2.2.4, additional details are briefly introduced (some more concerned with the actual implementation of the prototype). We proceed with information concerning the prototype implementation and communication infra-structure (section 2.2.5), including also an introduction to the r^3 library (section 2.2.6). Finally, we end this section by describing the current set of languages implemented in the r^3 prototype (section 2.2.7).

2.2.1 Resourceful Reactive Engines and Rules

The r^3 prototype is actually formed by an abstract network (physically distributed or not) of r^3 *Engine*s that cooperate towards evaluating ECA rules. The entry point into this network is an r^3 main *Engine* (or *ECAEngine*, as depicted in figure 2.9) providing a *Load* operation that allows an external *Client* to *activate* a *RulePackage*.

The *ECAEngine* interfaces with r^3 *Language* specific sub-engines (called *LanguageEngine*s), e.g. sub-engines for detecting events, querying Web data, testing conditions, etc. One such r^3 *LanguageEngine implements* one or more *Language*s, or *exports* an ad-hoc set of *LanguageConstruct*s. Each *LanguageConstruct* supplies the semantics of any *CodingElement* that *is* defined

[9]Integration with Semantic Web Services was attempted but had to be postponed. Nevertheless by using a literal SOAP body conforming to an RDF/XML serialization r^3 tries to keep an open perspective for the future.

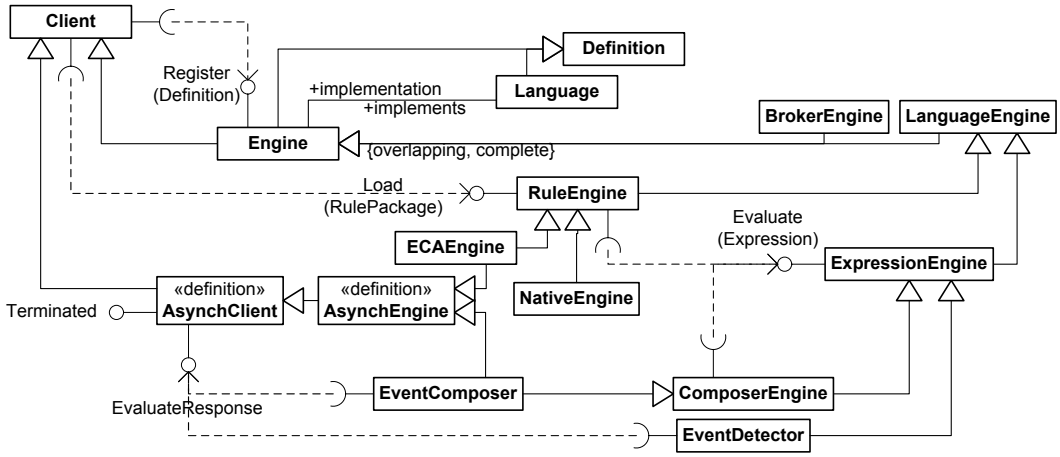

Figure 2.9: Engines

by it. *BrokerEngine*s may also be used in cases where an appropriate *LanguageEngine* is not known.

The API of r^3 *Engine*s is defined at a conceptual level based on an ontology of concepts, fostering the use of other Semantic Web technologies. This resource-based approach is present in every operation supported by an r^3 *Engine*. *RulePackage*s are *Load*ed by *RuleEngine*s whereas *Expression*s are *Evaluate*d by *ExpressionEngine*s.

*ExpressionConstruct*s allow both the specification of atomic *Expression*s (e.g. events or actions) using either generic or domain/application specific *Language*s (that some, e.g., *EventDetector implements*), and also the specification of composite *Expression*s using algebraic *Operator*s, of some (e.g. event or process) algebra provided by *ComposerEngine*s that recursively *Evaluate* the *OperatorArgument*s using other r^3 *ExpressionEngine*s.

According to ECA-ML [26] and as explained in section 2.1, *ECARule*s are formed by several distinct *component*s, namely: an *event component* (specifying the event occurrences that trigger the activation of the *ReactiveRule*), an optional condition part formed by several *component*s (that, based on an actual event occurrence, gather/*query* additional information and define/*test* the applicability of the rule under consideration), and an *action component* (stating the actions to execute whenever the *ECARule* is applicable). It is worth noting that this abstract component structure could also be used to accommodate other kinds of rules. The communication between these heterogeneous *component*s is achieved through the use of logical variables, bound to XML literals[10].

Example 2.2.1 *To illustrate, consider an* ECARule *notifying a teacher in case some administrative office processes a registration cancellation (*`ex:ev1`*), from a particular student to one of the teacher's lectures, provided that the student is considered a good student (according to some departmental criteria). In this example several languages are in fact used, namely: a language about people and notifying people (*`people:`*), an application/domain specific event language particular to some administrative office (*`office:`*), another domain specific query and test language (*`dept:`*), and a generic language for text manipulation (*`text:`*).*

```
ex:ev1 a :Event; :sub [ a :Expression;
```

[10]Currently these literals are opaque to an r^3 main engine, but nevertheless RDF references, serializations, or typed data values are not excluded.

```
  :is office:registration_cancelled;
  :having [ a :Parameter;
    :is office:student; :boundTo "St" ];
  :having [ a :Parameter;
    :is office:lecture; :boundTo "Lect" ] ].
ex:qry1 a :Query; :sub [ a :Expression;
  :is dept:lecture_teacher;
  :having [ a :Parameter;
    :is dept:lecture; :boundTo "Lect" ];
  :having [ a :Parameter;
    :is dept:teacher; :boundTo "Prof" ] ].
ex:tst1 a :Query; :sub [ a :Expression;
  :is dept:good_student;
  :having [ a :Parameter;
    :is dept:student; :boundTo "St" ] ].
ex:r1 a :ECARule;
  :event ex:ev1;
  :query ex:qry1;
  :test ex:tst1;
  :action [ a :Action :sub [ a :Expression;
    :is people:notify;
    :having [ a :Parameter;
      :is people:person; :boundTo "Prof" ];
    :having [ a :Parameter;
      :is people:message; :boundTo "Msg" ];
    :var [ a :Variable; :name "Msg";
      :equals [ a :Expression;
        :is text:merge;
        :use
          [ a :Variable; :name "Prof" :rename "1" ],
          [ a :Variable; :name "Lect" :rename "3" ],
          [ a :Variable; :name "St" :rename "2" ];
        :having [ a :Parameter;
          :is text:pattern;
          :literal """
            Dear %1,
            We are sorry to inform you that
            student %2 has cancelled his
            registration to your lecture %3."""
        ] ] ] ] ].
```

As an example of using logical variables for communicating among these different languages, consider example 2.2.1: a specific event occurrence provides a particular binding for variables `St` and `Lect` (identifying, respectively, a student and a lecture), given this binding, `Lect` is used to issue a query retrieving one or more bindings for variable `Prof` (the lecture professors, provided the query succeeds), whereas `St` is tested in order to filter out all non-good students, and finally the action is executed for all of the resulting substitutions (each containing all the previously bound variables, viz. `St`, `Lect` and `Prof`, and also an auxiliary local variable `Msg`, which is in fact a derived functional variable for each substitution).

*Language*s may be generic or application/domain specific. For instance, in example 2.2.1 instead of using `ex:qry1` based on the domain specific query element `dept:lecture_teacher` we could have used instead an XQuery [56] expression, this way re-using a generic language with a generic (but opaque) semantics (resulting in the domain specific semantics being hidden in `:data` and `:literal` opaque properties and so no longer available):

```
ex:r1 a :ECARule;
  ...
  :query [ a :Query; [ a :Expression;
    :boundTo "Prof";
    :is xquery:opaque;
    :having [ a :Parameter;
      :is xquery:base-uri;
      :data "DEPT_LECTURES" ];
    :having [ a :Parameter;
      :is xquery:literal;
      :literal """
        for $t in doc($Lect)//teacher
        return $t/@id;""" ];
    :use [ a :Variable; :name "Lect" ] ] ]
  ...
```

On the other hand, also in example 2.2.1, instead of using a generic element like `text:merge` we could have used a domain specific language to build the `Msg` text, e.g.:

```
ex:r1 a :ECARule;
  ...
    :var [ a :Variable; :name "Msg";
      :equals [ a :Expression;
        :is office:cancel_registration_warning;
        :having [ a :Parameter;
          :is office:teacher; :boundTo "Prof" ];
        :having [ a :Parameter;
          :is office:student; :boundTo "St" ];
        :having [ a :Parameter;
          :is office:lecture; :boundTo "Lect" ]
        ] ]
      ...
```

The recursive nature of *Expression*s, depicted in figure 2.1, also allows the use of different *Language*s in a single *component* through the composition of several *Argument Expression*s. This composition is achieved using generic algebraic *Language*s (e.g. an event algebra like Snoop [20, 9] or a process algebra like CCS [29, 13]) that may compose different application/domain specific (or even generic) *Language*s, as long as they share a common algebraic domain. For instance, by using an event algebra to define composite events, the event used in example 2.2.1 could be refined to trigger the rule only if if the teacher has previously confirmed the student merit:

```
ex:r1 a :ECARule;
  :event [ a :Event; :sub [ a :Expression;
    :is snoop:sequence;
    :taking [ a :Argument;
      :is snoop:first;
      :sub [ a Expression;
        :is teacher:confirmed_student_merit
        :having [ a :Parameter;
          :is teacher:id; :boundTo "Prof" ]
        :having [ a :Parameter;
          :is teacher:student; :boundTo "St" ] ] ];
    :taking [ a :Argument;
      :is snoop:next;
      :sub ex:ev1 ] ] ];
  ...
```

According to figure 2.2, a *Rule* may be either an *AbstractRule* or a *NativeRule*. A *NativeRule* (as illustrated in example 2.2.2) *is* a specific *NativeRuleConstruct* that *digs* its opaque *source* containing its literal specification (e.g. a DB2 [37] database trigger or an Outlook mail routing rule).

Example 2.2.2 *The* RuleSet *below* contains *the* AbstractRule *already defined in example 2.2.1 and also a* NativeRule *for a DB2 trigger that shows how the firing event of the* AbstractRule *(*`ex:ev1`*) could be generated from an existing DB2 database.*

```
ex:rs1 a :RuleSet;
  :includes ex:r1, ex:nr1.
ex:nr1 a :NativeRule;
  :is officedb2:trigger;
  :having [ a :Parameter;
      :is officedb2:schema;
      :literal "office" ];
  :having [ a :Parameter;
      :is officedb2:user;
      :literal "trigger_adm" ];
  :having [ a :Parameter;
      :is officedb2:passwd;
      :literal "some_pwd" ];
  :source [ a :Source;
      :is officedb2:opaque;
      :literal """
        after delete on office.registration
            referencing old as reg
            for each row mode db2sql
            begin atomic
              values(raise_event(
                "registration_cancelled",
                reg.student, reg.lecture));
            end""" ].
```

*NativeRule*s provide a minimal level of integration for legacy (or any other non-$\boldsymbol{r^3}$) sub-systems, allowing a *RuleSet* to contain all the relevant rules. Ignoring the existence of *NativeRule*s, and simply considering them as outcasts, can lead to unexpected inferences/actions. Considering them, not only integrates these other sub-systems, but it also makes it explicit (in the *RuleSet* description) that those *NativeRule*s and their associated *NativeEngine*s do have an impact on the behavior of the global system.

Nonetheless, *AbstractRules* are the preferred form of a *Rule* and it should be noticed that the *NativeRule* included in example 2.2.2 has an (intuitive) translation into an *AbstractRule*. Extending the interface of *NativeEngine*s to include also a *Compile* operation, although not considered currently, may provide an higher level of integration whenever such a translation is possible.

2.2.2 Loading and Evaluating Reactive Rules

$\boldsymbol{r^3}$ abstract *Message*s provide the foundation for the $\boldsymbol{r^3}$ API, as defined in figure 2.10. Each actual message is to be seen as a serialization of a Resource Description (as in RDF [48]) of an $\boldsymbol{r^3}$ abstract *Messsage*, defining a (possibly incomplete) RDF graph (optionally including references to external resources).

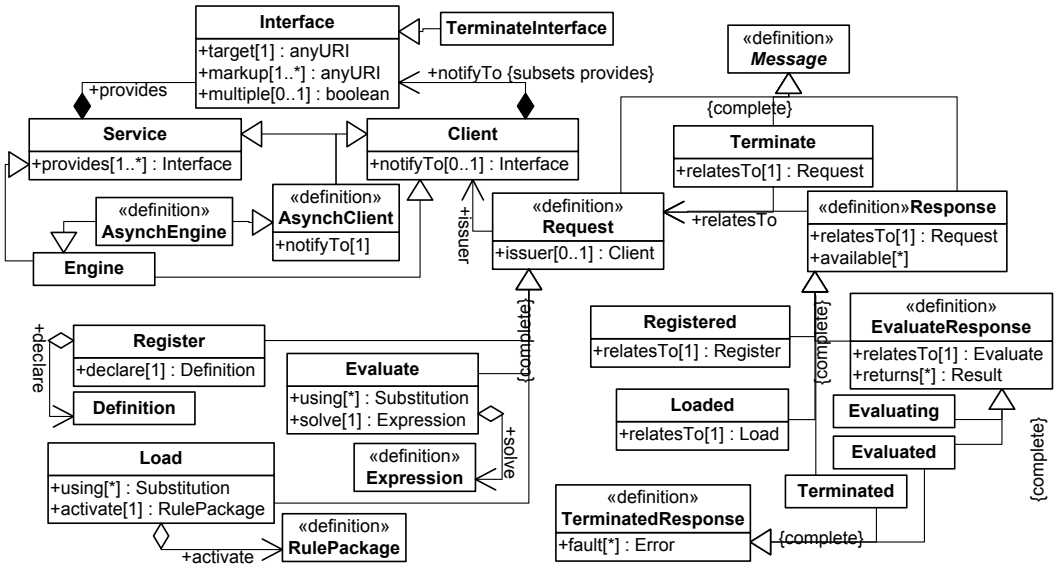

Figure 2.10: Messages

It is of particular relevance to clearly distinguish between the abstract concepts presented in figure 2.10 and any actual messaging framework (abstract or not) that is used to support them. The concepts presented here provide a conceptual view of *Message*s as required by $\boldsymbol{r^3}$ *Engine*s. This conceptual view is to be supported by full-blown messaging frameworks and (concrete) protocols, preferably standardized ones. Such supporting frameworks/protocols/standards must account for asynchronous communication; examples may include (and are not restricted to): pure HTTP using a REST architectural style [21], FIPA [22], (Semantic) Web Services, JMS [43].

*Load*ing a *RuleSet* (like `ex:rs1` in example 2.2.2) into an *ECAEngine* causes this *Engine* not only to activate the included *AbstractRule*s (like `ex:r1` in example 2.2.1) by resorting to the adequate *ExpressionEngine*s to *Evaluate* their *RuleComponent*s, but also to deliver any *NativeRule*s to the appropriate *NativeEngine*s where they must be natively embedded. For this purpose, $\boldsymbol{r^3}$ *NativeEngine*s must also provide a *Load* operation for embedding a *NativeRule* (like `ex:nr1` in example 2.2.2).

A *Load* (like any other) *Request* is sent by its *issuer Client* to the *target* of the appropriate *Interface* that some *Engine provides*, as illustrated in example 2.2.3. The actual *markup* to be used must be compatible with the one required by this *Interface*. This *Request* is then answered by the *Engine* (using the same *markup*) providing at least a *Response* (possibly asynchronous if dealing with an *AsynchClient* that *provides* a *notifyTo Interface* for this specific purpose).

The *ECAEngine* will return a *Loaded Response* as soon as the given *RulePackage* is *activate*d, otherwise, if it is not able to *activate* it, a *Terminated Response* will be returned. In the former case, the *RulePackage* will remain active until the *Client* sends a *Terminate* that will result in a *Terminated Response* acknowledging that the *RulePackage* deactivation is under way[11].

[11] An *Engine* may also send a *TerminatedResponse* at any point (provided the *Client* is an *AsynchClient*), and not only after getting a *Terminate* from the *Client*, thus signalling that, from that moment on, there will be no more *Response*s to the initial *Request*. An *Engine* may choose to do so either because all *Response*s were

Example 2.2.3 *The following illustrates the loading of the* RuleSet `ex:rs1` *from example 2.2.2.*

```
@prefix ecaeng: <http://engine.nop#>.
@prefix ecarls: <http://engine.nop/rules/>.
@prefix clireq: <http://client.nop/requests#>.

ecaeng: a :ECAEngine;
  :provides ecaeng:interface;
  :notifyTo ecaeng:interface.
ecaeng:interface a :Interface;
  :target "http://engine.nop/service";
  :markup "http://rewerse.net/I5/NS/2006/r3".

 # The Client wants to Load a RuleSet
 clireq:rs1 a :Load;
   :issuer [ a :Client;
     :notifyTo [ a :Interface;
       :target "http://client.nop/notify";
       :markup "http://rewerse.net/I5/NS/2006/r3"
     ] ];
   :activate ex:rs1.

 # Rules have been activated, in this case the
 # Engine chooses to expose the Loaded rules
 # through the resource ecarls:rs1
 [] a :Loaded; :relatesTo clireq:rs1;
   :available ecarls:rs1.

 # Rules are active in the Engine and
 # the Client may retrieve the current
 # state of resource ecarls:rs1

 # The Client wants to deactivate
 # the previously Loaded rules
 [] a :Terminate; :relatesTo clireq:rs1.

 # All rules relating to clireq:rs1 are
 # assumed as deactivated/unloaded and
 # any associated resources as released
 [] a :Terminated; :relatesTo clireq:rs1.
```

A *Load Request* causes the *ECAEngine* to issue additional subordinated *Request*s *target*ed to specific *LanguageEngine*s. For instance, as a result of the *Request* included in example 2.2.3, the *ECAEngine* would submit a *Load Request* (as shown in example 2.2.4) to the *target* of a *NativeEngine* that *implements* the `officedb2:` *Language* used in `ex:nr1`.

Example 2.2.4 *Installing the* NativeRule `ex:nr1` *contained in* `ex:rs1` *from example 2.2.2. These* Message*s should be interleaved with the ones already included in example 2.2.3.*

already generated, or because it can no longer honor the initial *Request* due to some (probably unexpected) circumstance. In any case, a *TerminatedResponse* is always the last *Message* that *relatesTo* a particular *Request*, signalling that all the *Engine*'s resources associated with that *Request* are already released (or at least they must be assumed as such). A *Terminate* of a *Request* trivially succeeds, if it is sent after a *TerminatedResponse* for that same *Request* has been issued.

```
@prefix engdb2:
   <http://office.nop/db2service#>.
@prefix ecareq:
   <http://engine.nop/requests#>.

 engdb2: a NativeEngine;
   :provides [ a :Interface;
     :target "http://office.nop/db2service";
     :markup "http://rewerse.net/I5/NS/2006/r3"
   ];
   :implements officedb2:.

 # Requested Load clireq:rs1 and before
 # responding Loaded must Load ex:nr1
 ecareq:nr1 a :Load;
   :issuer ecaeng:;
   :source ex:nr1.

 [] a :Loaded; :relatesTo ecareq:nr1.
 # The Loaded Response can now be sent
 # after knowing that ex:nr1 is Loaded

 # Got Terminate for clireq:rs1, so
 # must also Terminate ecareq:nr1
 [] a :Terminate; :relatesTo ecareq:nr1.

 [] a :Terminated; :relatesTo ecareq:nr1.
 # Now clireq:rs1 can also be Terminated
```

Upon a *Load Request*, an *ECAEngine*, besides *Load*ing contained *NativeRule*s and *RuleSet*s, as explained before, also has to activate (viz. *Evaluate*) any *AbstractRule*s (viz. *ECARule component*s) contained in the given *RulePackage*. To achieve this, it must interface with several *ExpressionEngine*s (that actually provide the specific *Evaluate* implementation) according to the specific *LanguageConstruct*s involved in the different *RuleComponent*s, namely: an *event*, an optional *query* (or more than one), an optional *test* (or more), and a *action*s. Since rule *component*s are *Expression*s and communicate with each other using logical variables, they are *Evaluate*d (by *ExpressionEngines*) *using* a context provided by several alternative *Substitution*s. Each *Substitution* must be unique and must enumerate all (and only those) variables involved in the *Expression* (possibly restricting some of them, by *binding* them to specific values or to other variables, as further detailed in section 2.2.3). While *Evaluating* an expression *using* a given context the *ExpressionEngine returns* all possible *Result*s, or none if the expression fails (meaning the *Expression* was *Evaluated* and *returns* no *Result*s). Besides (optionally) *using* a set of *Substitution*s, any *Result* may also return a *literal* value (cf. figure 2.11). This value may be used as an *Argument* for another *Expression* or *boundTo* a variable.

As an illustrative example consider the activation of the *ECARule* `ex:r1` of example 2.2.1. When an *ECAEngine* receives a *Load Request* for a *RulePackage* containing this rule (as `clireq:rs1` in example 2.2.3), it must activate the rule by issuing a subordinated *Evaluate Request* for the rule *event* (`ex:ev1`) to the *EventDetector* that *implements* the (`office:`) *Language* (as shown in example 2.2.5).

Example 2.2.5 *Activating the* ECARule `ex:r1` *contained in* `ex:rs1` *from example 2.2.2. These* Message*s should be interleaved with the ones already included in examples 2.2.3 and 2.2.4.*

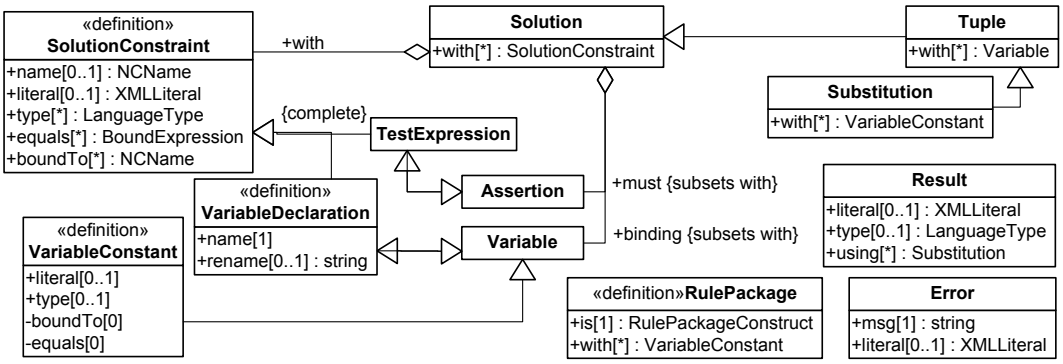

Figure 2.11: Solutions and Results

```
# Requested Load clireq:rs1 and before
# responding Loaded must launch Evaluate
# for ex:ev1 used in ex:r1
ecareq:ev1 a :Evaluate;
  :issuer ecaeng:;
  :using [ a Substitution;
    :binding
      [ a :Variable; :name "St" ],
      [ a :Variable; :name "Lect" ] ];
  :solve ex:ev1.

[] a :Evaluating; :relatesTo ecareq:ev1.
# Evaluate for ex:ev1 accepted, actual
# Results will be communicated later
# The Loaded Response can now be sent

# Later whenever an event occurrence is
# detected the literal event is returned
# together with the substitution
[] a :Evaluating; :relatesTo ecareq:ev1;
  :returns [ a Result;
    :literal """
      <offc:registration_cancelled
        xmlns:offc="http://office.nop/events#"
        offc:docid=
          "http://office.nop?did=DRC20062395"
      />""";
    :using [ a Substitution;
      :binding
        [ a :Variable; :name "St"
          :literal "S4521" ],
        [ a :Variable; :name "Lect"
          :literal "L144" ] ] ].

# Got Terminate for clireq:rs1, so
# must also Terminate ecareq:ev1
[] a :Terminate; :relatesTo ecareq:ev1.

[] a :Terminated; :relatesTo ecareq:ev1.
# Now clireq:rs1 can also be Terminated
```

This *EventDetector* will then signal back (asynchronously) each occurrence (of `ex:ev1`) it

detects, by sending an *Evaluating Response* that *returns* the *Result* associated with the detected event. Note that if, at any point, the *EventDetector* concludes that the event can no longer occur (e.g. because deadline for registration cancellations has expired), it may send an *Evaluated Response* (depicted in figure 2.10) meaning that the event expression has no additional solutions and so the *Evaluate Request* is *Terminated*.

For each *Result* of *Evaluating* the *event* component, the *ECAEngine* will create a rule instance *using* the *Substitution*s it *returns*[12]. For this rule instance, *query* and *test* components are then taken into consideration by issuing *Evaluate Request*s to the appropriate query processors and test evaluators, as shown in example 2.2.6.

Example 2.2.6 *Considering an ECARule instance, originated by the event occurrence* return*ed in example 2.2.5.*

```
ecareq:qry1 a :Evaluate;
  :issuer ecaeng:;
  :using [ a Substitution;
    :binding
      [ a :Variable; :name "Prof" ];
      [ a :Variable; :name "Lect"
        :literal "L144" ] ];
  :solve [ a :Expression;
    :is dept:lecture_teacher;
    :having [ a :Parameter;
      :is dept:lecture; :boundTo "Lect" ];
    :having [ a :Parameter;
      :is dept:teacher; :boundTo "Prof" ] ].
[] a :Evaluated; :relatesTo ecareq:qry1;
  :returns
    [ a Result; :using [ a Substitution;
      :binding
        [ a :Variable; :name "Prof"
          :literal "T57" ],
        [ a :Variable; :name "Lect"
          :literal "L144" ] ] ],
    [ a Result; :using [ a Substitution;
      :binding
        [ a :Variable; :name "Prof"
          :literal "T01" ],
        [ a :Variable; :name "Lect"
          :literal "L144" ] ] ].

ecareq:tst1 a :Evaluate;
  :issuer ecaeng:;
  :using [ a Substitution;
    :binding
      [ a :Variable; :name "St"
        :literal "S4521" ] ];
  :solve [ a :Expression;
    :is dept:good_student;
    :having [ a :Parameter;
      :is dept:student; :boundTo "St" ] ];
[] a :Evaluated; :relatesTo ecareq:tst1;
  :returns [ a Result ]
```

[12]The *literal* value is to be discarded unless the *event Expression* is *boundTo* a variable, as detailed later in section 2.2.3, in which case this variable binding must be added to the *Substitution*.

Assuming no optimizations, first the *query* is *Evaluated*, extending the original *Substitution* with bindings for additional variables (possibly multiple ones, generating alternative *Substitution*s as in example 2.2.6), and if it succeeds, then the *test* is *Evaluated*. Both *test* and *query* may possibly filter out some of the *Substitution*s.[13].

Given the set of *Substitution*s for which all *query* and *test* components succeed, the *action* component is finally executed by an *ExpressionEngine*, as partially shown in example 2.2.7.

Example 2.2.7 *Executing an ECARule instance, with the extended (or filtered)* Substitution*s resulting after consideration of the rule conditions in example 2.2.6.*

```
# {Prof=T57, Lect=L144, St=S4521}
ecareq:ex1 a :Evaluate;
  :issuer ecaeng:;
  :using [ a Substitution;
    :binding
      [ a :Variable; :name "Prof";
        :rename "1"; :literal "T57" ],
      [ a :Variable; :name "Lect";
        :rename "3"; :literal "L144" ],
      [ a :Variable; :name "St";
        :rename "2"; :literal "S4521" ] ];
  :solve [ a :Expression;
      :is text:merge;
      :literal # simplified
        "%1, %2 not attending %3."
      ] ].
[] a :Evaluated; :relatesTo ecareq:ex1;
  :returns [ a Result;
    :literal "T57, S4521 not attending L144." ].
# {Prof=T01, Lect=L144, St=S4521}
# ... omitted ...

ecareq:act1 a :Evaluate;
  :issuer ecaeng:;
  :using [ a Substitution;
    :binding
      [ a :Variable; :name "Prof"
        :literal "T57" ],
      [ a :Variable; :name "Msg"
        :literal
          "T57, S4521 not attending L144." ] ];
  :using [ a Substitution;
    :binding
      [ a :Variable; :name "Prof"
        :literal "T01" ],
      [ a :Variable; :name "Msg"
        :literal
          "T01, S4521 not attending L144." ] ];
  :solve [ a :Expression;
    :is people:notify;
    :having [ a :Parameter;
      :is people:person; :boundTo "Prof" ];
    :having [ a :Parameter;
      :is people:message; :boundTo "Msg" ] ].
[] a :Evaluated; :relatesTo ecareq:act1;
```

[13]Notice that an *Expression* succeeds if it *returns* at least one *Result*, even an empty one like the *test Evaluated* in example 2.2.6.

```
  :returns [ a Result ]
```

For the *action* component, it is of particular relevance to ensure that all (distinct) *Substitution*s (for the variables involved in the *Expression*) are included in a single *Evaluate Request*, in order to account for any possible transactional behavior (associated with the *LanguageConstruct* defining the actual operational semantics of the *action*).

Example 2.2.8 *The following further illustrates the concept of* Language*s showing a possible definition for the* Language*s used in examples 2.2.1, 2.2.2 and 2.2.9.*

```
office:language a :Language;
  :defines office:registration_cancelled,
           office:student, office:lecture,
           office:registration, office:state.
office:registration_cancelled a :Functor;
  :binds office:student, office:lecture.
office:registration a :Functor;
  :binds office:student, office:lecture,
         office:state.

dept:language a :Language;
  :defines
    dept:term_completed,
    dept:course_lecture, dept:lecture_teacher,
    dept:course, dept:lecture, dept:teacher,
    dept:good_student, dept:student.
dept:term_completed a :Functor;
  :binds dept:course.
dept:course_manager a :Functor;
  :binds dept:course, dept:teacher.
dept:course_lecture a :Functor;
  :binds dept:course, dept:lecture.
dept:lecture_teacher a :Functor;
  :binds dept:lecture, dept:teacher.
dept:good_student a :Functor;
  :uses dept:student.

people:language a :Language;
  :defines people:notify,
           people:person, people:format,
           people:message.
people:notify a :Functor;
  :uses people:person, people:format,
        people:message.
people:format a :FunctionalParameter;
  :default "plain".

text:language a :Language;
  :defines text:merge, text:pattern.
text:merge a :OpaqueConstruct;
  :digs text:pattern.

officedb2:language a :Language;
  :defines officedb2:trigger,
           officedb2:schema,
           officedb2:user, officedb2:passwd.
officedb2:trigger a :NativeRuleConstruct;
```

```
  :uses officedb2:schema,
        officedb2:user, officedb2:passwd;
  :digs officedb2:trigger.

util:language a :Language;
  :defines util:count.
util:count a :Aggregator;
  :binds office:student, office:lecture.

xcerpt:language a :Language;
  :defines xcerpt:eval, xcerpt:construct.
util:count a :Aggregator;
  :uses xcerpt:construct.
```

2.2.3 Variables, Substitutions and Results

As described in section 2.2.2, the *Evaluate* of an *Expression*, using some (input) *Substitution*s, *returns* several *Result*s. Each *Result* may provide a *literal* value *using Substitution*s. Usually each returned *Substitution* is subsumed by one of the input *Substitution*s, but it may instead actually subsume several input *Substitution*s (by omitting some of the involved *Variable*s, to be considered as unbound). An empty *Result Substitution* set (subsuming all input *Substitution*s) may be omitted altogether. The *literal* value of a *Result* is required to be present if the *Expression* is *boundTo* a *Variable*. If the *Expression* is *boundTo* a *Variable* a *binding* for this *Variable* is to be created (using the returned *literal* value) and joined with the returned *Substitution*. The *Substitution* resulting from this join is additionally joined with all the input *Substitution*s, yielding the actual set of output *Substitution*s associated with that *Result* (this set must not be empty). Given this, a *Result* may actually stand for several *Result*s, all sharing the same *literal* value (if there is one). The *Evaluate* of the *Expression* is said to succeed if it outputs at least a *Result*.

The *Evaluate* of an *Aggregation* is expected to *aggregate* the values of a set of *Variable*s (contained in the input *Substitution*s) optionally *groupedBy* a set of other *Variable*s (also contained in the input *Substitution*s). As such, the *Result*s of an *Aggregation* are further restricted: their *Substitution*s cannot contain any *binding* for the *aggregate*d *Variable*s (unless explicitly included as *groupedBy* at the same time[14]), and they must always include the *aggregate*d value as the *literal Result* value. An *Aggregation returns* at most a single *Result* for each set of distinct values of the *groupedBy Variable*s. The actual *Aggregation* to be performed *is* defined by parametric *Aggregator*s like `util:count` or `xcerpt:eval` in example 2.2.9.

Example 2.2.9 *To illustrate the use of* Aggregation*s the following specifies an* ECARule *(*`ex:r2`*) that sends a map of all registration cancellations for a given course at the end of every term (re-using* `ex:qry1` *from example 2.2.1). Notice that the action uses only two variables (*`Mngr` *and* `CourseMsg`*) and so it sends a message for each distinct pair of values of these variables.*

```
ex:r2 a :ECARule;
  :event [ a :Event; :sub [ a :Expression;
    :is dept:term_completed;
```

[14] If an *Aggregation* is *groupedBy* on the same set of variables it *aggregate*s, it becomes a simple function that returns a single composed value for each set of distinct values of the given variables.

```
  :having [ a :Parameter;
    :is dept:course; :boundTo "Course" ] ] ];
:query [ a :Query; :sub [ a :Expression;
  :is dept:course_manager;
  :having [ a :Parameter;
    :is dept:course; :boundTo "Course" ];
  :having [ a :Parameter;
    :is dept:teacher; :boundTo "Mngr" ] ] ];
:query [ a :Query; :sub [ a :Expression;
  :is dept:course_lecture;
  :having [ a :Parameter;
    :is dept:course; :boundTo "Course" ];
  :having [ a :Parameter;
    :is office:lecture; :boundTo "Lect" ] ] ];
:query ex:qry1;
:query [ a :Query; :sub [ a :Expression;
  :is office:registration;
  :having [ a :Parameter;
    :is office:state; :literal "cancelled" ];
  :having [ a :Parameter;
    :is office:student; :boundTo "St" ];
  :having [ a :Parameter;
    :is office:lecture; :boundTo "Lect" ] ] ];
:query [ a :Query; :sub [ a :Expression; :boundTo "Num";
  :is util:count; :aggregate "St";
  :groupedBy "Course", "Lect" ] ];
:action [ a :Action; :sub [ a :Expression;
  :is people:notify;
  :having [ a :Parameter;
    :is people:person; :boundTo "Mngr" ];
  :having [ a :Parameter;
    :is people:format; :literal "html" ];
  :having [ a :Parameter;
    :is people:message; :boundTo "CourseMsg" ];
  :var [ a :Variable; :name "CourseMsg";
    :equals [ a :Expression;
      :is xcerpt:eval;
      :aggregate "Lect", "Num", "Prof";
      :groupedBy "Course";
      :having [ a :Parameter;
        :is xcerpt:construct; :literal """
        html { body {
          h1 { var Course },
          table {
            tr {
              th { "Lecture" },
              th { "Cancellations" },
              th { "Teachers" } },
            all tr {
              td { var Lect },
              td { var Num },
              td { ul {
                all li { var Prof } } }
            } } } }""" ] ] ] ] ].
```

Every input *Substitution* must explicitly include all (and only those) *Variable*s involved in the *Expression* to *Evaluate*. If there is a single empty input *Substitution* (no variables involved) it may be omitted. The set of *Variable*s involved in an *Expression* includes the ones *boundTo* its

*Parameter*s or *Argument Expression*s. If there is a *Variable boundTo* the whole *Expression*, it is also included in this set. Additionally, for an *Aggregation*, *groupedBy* and *aggregate Variable*s are also included. Besides these implicitly involved *Variable*s, an *Expression* may also be specified *with* several explicit *Variable* declarations[15] (explicitly declaring also any *Variable*s *boundTo* those). The set of all involved *Variable*s must be fully included in every input *Substitution* (regardless if they are bound or unbound, thus allowing all the *with* declarations to be omitted in an *Expression* to be *solve*d).

An *Opaque Expression* must be specified together *with* all the *Variable*s non-transparently referenced by its *literal* part. An *Opaque* may also *rename* its *Variable*s according to the naming conventions used by its *literal* part[16] (as shown in example 2.2.1).

For each *Expression*, either a single *Evaluate Request* is issued *using* all the distinct input *Substitution*s (this is mandatory for *action component*s, as said before, and also for *Aggregation*s); or several individual *Request*s may be issued (e.g. one for each *Substitution*, or one for each set of distinct values for *FunctionalParameter*s).

As usual, the scope of a *Variable* is the *Rule* that contains the *Expression*, unless it is a *LocalVariable* in a particular *Expression*. The scope of such *LocalVariable*s is the *Expression* where they are explicitly declared, and they are usually used to hold *literal* functional *Results* (e.g. to be used as *Parameter*s, like `Msg` in example 2.2.1) or to express local join conditions between *Expression Argument*s.

All *Variable*s are assumed to be *FreeVariable*s, unless they are explicitly declared otherwise or implicitly enforced to be *BoundVariable*s by the context in which they are involved (viz. *groupedBy*, *aggregate*, *boundTo* a *FunctionalParameter*, or *boundTo* another *BoundVariable*). An *Evaluate Request* can only be issued for a specific *Expression* if all the *Variable*s the *Expression use*s are actually bound to *literal* values[17]. The *FreeVariable*s and *LocalVariable*s *mode* may be unbound at the time of the *Evaluate Request*. The nature of a *Variable* poses no restriction on the *Result*s of an *Evaluate* (for instance, a *FreeVariable* may be bound in a *Result* but is not required to be so).

An explicit *Variable* declaration may also include restrictions to the domain of the *Variable*, namely: it may be restricted to having a specific *literal* value (which may be understood as a simple *binding*), or it may be declared that its value always *equals* one (or more) *Expression*s (as shown in example 2.2.1 for *Variable* `Msg`). Declaring that a *Variable equals* several *Expression*s defines the domain of the *Variable* as the intersection of the sets of *literal* values that each *Expression returns*. Notice that *equals* (and *test component*) *Expression*s can only produce new bindings for *LocalVariable*s, as such, the *Evaluate Request* for these *Expression*s can only be issued when all *Variable*s they *use* or *bind* are actually bound to literal values.

[15] For an *AbstractExpression*, these explicit declarations are restricted to the *Variable*s already implicitly involved in it or in one of its *Argument Expression*s. This restriction stems from the fact that the full semantics of *AbstractExpression*s *is* to be defined by their parametric *ExpressionConstruct*s (together with any *OperatorArgument*s, in case of algebraic *Operator*s).

[16] Notice that this *rename* information is not to be used to perform any form of blind textual replacement in the *literal* text before an *Evaluate*. Instead, this *rename* information is to be included in the *Substitution*s used when issuing the *Evaluate Request* (as illustrated in example 2.2.7).

[17] Notice that, actually, the *Variables used* by different *Expression*s define a partial evaluation order among them (most relevant when there are multiple conditions in a rule).

2.2.4 Rule Instances, et al.

For the sake of completeness of the presented $\boldsymbol{r^3}$ ontology, it is worth mentioning some additional details that, although not fully supported by the current prototype, are also included in the current version of the ontology.

Figures 2.9 and 2.10 include a *Register* operation that may be used to supply *Definition*s of *Language*s and *Engines* (usually to a *ComposerEngine*, a *RuleEngine*, or a *BrokerEngine*). This operation is in fact supported by the current prototype, but disregarding any inconsistency matters that it may introduce.

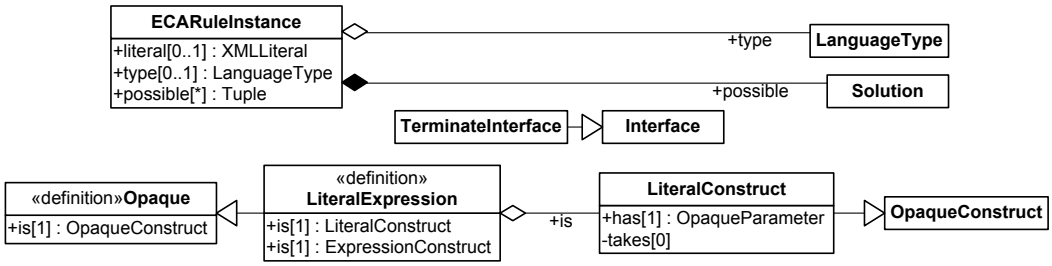

Figure 2.12: Implementation Details

Furthermore, if an *Engine* creates or updates resources, as a result of a *Request*, these resources may be included in any *Response*. In doing so, the *Engine* makes those resources *available* to the *Client*, as shown in figure 2.9. For instance, during a *Load* conversation (as the one in example 2.2.3), chances are that the *Engine* will keep a materialized copy of the *Loaded RulePackage* (like `ecarls:rs1`) and the *ECAEngine* may choose to share that copy with the *Client* making it *available*.

Often these *available* resources will be persistent ones (with an associated URI), at least while the *Request* is not *Terminated*. These resources may even have a dynamic (evolving) nature (for instance `ecarls:rs1` may include information about the current state of evaluation of the different rules by including any *active ECARuleInstance*s) and the *Client* may retrieve updated descriptions, whenever he wishes to. These *available* resources are released (and usually their representation becomes unavailable) when the *Request* is *Terminated*. Currently, the $\boldsymbol{r^3}$ prototype does not expose any *available* resources, but internally it already relies upon *ECARuleInstance*s to keep the evaluation state of reactive rules.

Notice that figure 2.10 does not include any functionality regarding consulting resources (and their representations). Nonetheless, consulting resources is not only needed to retrieve *available* resources, it is also core to the $\boldsymbol{r^3}$ API, since, as said before, a *Message* may be an incomplete RDF graph containing references to external resources. This means that in order to fully understand a *Request* (or a *Response*) an *Engine* (or *Client*) may have to retrieve additional resource representations[18]. Such functionality (of retrieving resource representations) is core to the (Semantic) Web itself and not specific to rule engines.

The ontology also includes (see figure 2.12) an incomplete set of specializations of *Interface* and *Expression* to be refined in future versions. These specializations are currently used for some minor implementation details (viz. ECA-ML integration), but are not yet consistently used throughout the ontology.

[18]How (and to what extent) this is actually done is up to the receiver, as long as it manages to understand the *Message* (correctly).

2.2.5 Prototype Implementation and Communication Details

The $\boldsymbol{r^3}$ prototype is implemented as a Java 5 Servlet [41], with minimal use of JavaServer Pages [40] for the prototype frontend. The $\boldsymbol{r^3}$ Servlet provides an HTTP POST interface (embedding also Apache Axis [35] 1.3, for SOAP 1.1 support), and is packaged as a WAR[19] file, thus allowing easy installation under any servlet container[20].

The core of the $\boldsymbol{r^3}$ ECA rule engine relies heavily upon Jena [42] 2.4 for RDF support (wrapped in Jastor-like [39, 25] Java classes). Nevertheless, the core of this engine is implemented using Prova (version 1.9, commit 14[21], patched[22]). The use of Prova embedded in a Servlet required the development of an extension to the Java/Prova integration mechanisms. The extended/patched version of Prova 1.9 is also included in the WAR package.

The prototype frontend allows to browse through different Use-Case scenarios formed by Groups of Examples, inspecting the actual requests that would be posted. Installed on an application server, it allows the actual posting of requests to that same server and the inspection of the synchronous responses. It is also possible to unpack the WAR file and browse the files offline. This offline configuration may have weaker security restrictions and allow the posting of requests to any application server that you are aware of (just by redirecting the URL of the $\boldsymbol{r^3}$ Server in the frontend). Currently the Use-Case scenarios available are B-Domain (see section 3.2) or PubMed (see section 3.3) related, and an ad-hoc collection of development tests is also included.

Founding the $\boldsymbol{r^3}$ API on an abstract messaging layer, like the one introduced in section 2.2.2, provides an architecture where other more specific layers may be semantically plugged providing support for: many concrete syntaxes (or serializations), alternative protocols, and even ontology reasoners. All that is assumed is that any message, at the time it gets delivered to the core of an $\boldsymbol{r^3}$ *Engine*, is a fully materialized resource description of a *Message* (detailed in figure 2.10), and so the rule engine can focus on what constitutes its true added value (realizing reactive behavior), and rely on (re-using) other Semantic Web technologies for the rest.

For sure, a prototype like $\boldsymbol{r^3}$, in order to provide some actual functionality, must make some choices and eventually commit to concrete syntaxes and protocols, and for that it must include specific code to support its choices, but if the internal architecture of the prototype conforms to the abstract messaging layer keeping a clean separation between these external layers and the core of the engine, the latter will be a re-usable "abstract" library (that actually implements the behavior of reactive rules).

Currently in $\boldsymbol{r^3}$ we have chosen to implement an external layer (as said before, using a Java Servlet based on Axis 1.3) supporting simultaneously a SOAP 1.1 and a pure HTTP REST [21] style interface. The body of an HTTP POST request (SOAP wrapped or not) may be an ECA-ML [3] XML document (viz. the restricted ECA-ML subset defined in `http://rewerse.net/I5/r3/DOC/2006/eval.xsd`) or any of the XML based RDF serializations supported by Jena 2.4. Supporting N3 and other non-XML serializations (e.g. using CDATA XML nodes) would require a more involved architecture given the XML Element orientation of Axis (probably the

[19] r3.war: including r3.jar with sources, released under the Apache License - version 2.0.

[20] Just copy the r3.war file to the `webapps` directory of your Apache Tomcat installation (tested with version 5.5.12).

[21] Regretfully, support for Prova 1.9 was dropped (before final release, cf. `http://prova.ws/forum/viewtopic.php?t=160`), in favor of the new upcoming version 2.0 (which includes a major rewriting of the inference engine and has not yet been released). The available Prova version (1.8) is more than 2 years old and does not constitute a viable alternative.

[22] `http://prova.ws/forum/viewtopic.php?t=149`

recently released Axis2 [36] would be the best choice to achieve this).

A preliminary WSDL 1.1 [54] specification (`http://rewerse.net/I5/r3/DOC/2005/r3.wsdl`) defines the actual SOAP [52] 1.1 binding (`document` style - not `rpc` - with a `literal` body - not `encoded`). This specification is based on an XML Schema [55] (`http://rewerse.net/I5/NS/2006/r3/r3.xsd`) that defines a markup based on RDF/XML for the $\boldsymbol{r^3}$ ontology. This XML Schema is obtained automatically from the OWL-DL definition of the $\boldsymbol{r^3}$ ontology, using a generation tool still under development (which is not the focus of our work but that may prove relevant as a proof-of-concept for other REWERSE Working Groups, viz. WGI1).

It is worth noting that both the protocol and the markup could contribute to the actual serialization of *Message*s. For instance, one could use WS-Addressing [53] in the SOAP header, identifying a *Request* - `wsa:MessageID` - *Message* (to which a *Terminate* - `wsa:RelatesTo` - or *Response* - `wsa:ReplyTo` - *relatesTo*), and the specific subclass - `wsa:Action` - of the *Message*, reserving the SOAP body for providing the serialization of other properties of the *Message* (like parameters, for instance). Similar results may be achieved by introducing proprietary HTTP headers. The external (concrete) layers would be responsible for mapping all this information into a materialized resource description of a *Message*.

Also, these external layers may be capable of reasoning improving their Semantic (Web) potential. For instance, once obtained the explicit RDF graph contained in a message, it is possible to scan that graph for any resources used only as objects of tuples (excluding those that are also used as subjects), and go grab their representation from the Web, thus building an extended graph (that would for instance include ontology information), and right before submitting it to the core of the engine (with the help of an ontology reasoner, like Pellet [46]) materialize also derived tuples.

Finally, the external layers can even be made adaptive by extending the *Interface* class with properties describing the actual markup and protocol supported. Currently the $\boldsymbol{r^3}$ ontology only allows the specification of a *markup* URI, but a richer definition (e.g. introducing a new Markup class, instead of the *markup* property) must be considered in the future, eventually including markup transformations (e.g. XSLT [58]). The current $\boldsymbol{r^3}$ implementation supports, in its external layers, a limited form of adaptive functionality by recognizing two different *markup*s (viz. `http://rewerse.net/I5/NS/2006/r3` for RDF/XML serializations; and `http://www.semwebtech.org/eca/2006/eca-ml` for ECA-ML); and by issuing local (in-memory) calls to *Load* and *Evaluate* (avoiding the overhead of remote calls) for locally supported *component Language*s.

2.2.6 Building r^3 Component Engines

Since the beginning of the work on the $\boldsymbol{r^3}$ ontology and the $\boldsymbol{r^3}$ prototype, it became clear that there was a considerable amount of code that would be shared by $\boldsymbol{r^3}$ main rule engines and the different $\boldsymbol{r^3}$ expression sub-engines (hopefully to be developed for an unlimited set of *component Language*s).

To some extent, this sharing of code, may be solved by an adequate distributed component-based architecture. This is the case for the requirement to evaluate expression arguments (which is not trivial) that is common to all algebraic languages and shared by the main engines (given precisely the algebraic nature of those languages). This requirement may be solved/shared by the introduction of *BrokerEngine*s to be used both by the the main rule engines and any algebraic engines[23]. Nevertheless, other (not so minor) details like actually invoking a broker

[23] Besides *BrokerEngines*, other more involving interfaces were also suggested (like the one included in previous

engine (or, e.g., dealing with variables and unification, and joining substitution sets) are hardly solved by distributed architectures.

Eventually, all fully compliant $\boldsymbol{r^3}$ expression engines will share an enormous amount of code that is already part of an $\boldsymbol{r^3}$ main engine. In order to minimize this issue and ease, as much as possible, the process of implementing and testing new expression engines, $\boldsymbol{r^3}$ facilitates a Java library that abstracts away matters like communication protocols (HTTP, SOAP); binding variables and generating alternative solutions; or even the $\boldsymbol{r^3}$ ontology itself (and dealing with the Jena RDF models).

The final goal is to allow developers of expression engines to focus on the specificities of the languages to implement; freeing them from $\boldsymbol{r^3}$ details which are to be abstracted by tailored evaluation context components (under development at the time of this writing).

Currently, the functionality of the $\boldsymbol{r^3}$ development library is still limited, nevertheless you may use it if you are willing to do so. It provides, e.g., a `net.rewerse.i5.r3.test.Tester` class for offline testing (without requiring the use of a Java application server) and even a `net.rewerse.i5.r3.test.dumpster` *Engine* for receiving asynchronous *Response*s. Actually this library is being used to develop all the *component Language*s included in the $\boldsymbol{r^3}$ prototype (as illustrated by the excerpts of Java code included in section 2.2.7). More documentation is being made available. This documentation is meant to fully support developers of component languages. Nevertheless, you can also contact us if you plan to use this library to develop any component language. We will do our best to further support you!

Here are a few tips on how you may develop your own $\boldsymbol{r^3}$ *Engine*.

First of all you need the development library available at `http://rewerse.net/I5/r3/TST/install/r3lib.jar`. Using this library you should create/build a Web Dynamic Project exporting a Servlet[24]. Then you need to create a Servlet, a Server[25] and an Evaluator as illustrated by the following Java excerpt:

```
package com.ricardoamador.r3.test.engine;

public class Servlet extends net.rewerse.i5.r3.Servlet {
  protected void initServices() {
    super.initServices(); // declare r3 default services if present
    registerService("com.ricardoamador.r3.test.engine.Server", "dummy");
  }
}

package com.ricardoamador.r3.test.engine;

import net.rewerse.i5.r3.eval.ExprEvaluator;

public class Server extends net.rewerse.i5.r3.eval.Server {

  protected ExprEvaluator createEvaluator(String url) throws Exception {
    return new Evaluator(url);
  }
```

$\boldsymbol{r^3}$ versions, and shown in figures reffig:002more01 and reffig:001more01). These are probably too involved to be really useful.

[24]The easiest way to create an $\boldsymbol{r^3}$ Web Dynamic Project in Eclipse is to import `http://rewerse.net/I5/r3/TST/install/r3xra.war` into your own Eclipse workspace

[25]For each engine several instances of Server may be created, whereas one and only one Evaluator is instanciated.

```
}

package com.ricardoamador.r3.test.engine;

import org.w3c.dom.Document;
import org.w3c.dom.Node;
import net.rewerse.i5.r3.R3Utils;
import net.rewerse.i5.r3.eval.ExprEvaluator;
import net.rewerse.i5.r3.juice.ontology.Language;
import net.rewerse.i5.r3.juice.ontology.Tuple;
import net.rewerse.i5.r3.juice.ontology.Variable;

public class Evaluator extends ExprEvaluator {
  public static final Language LANG =
    r3lang(Evaluator.class, "test.owl",
        r3url("http://www.ricardoamador.com/tmp/test.owl"), "test");
  protected Language getLanguage() {
    return LANG;
  }
  protected void evaluate(Context ctx) throws Exception {
    String opname = ctx.opname();

    if (opname.equals("opaque")) {
      String lvars = "";
      for (Tuple t: ctx.getUsing()) {
        lvars += "\n";
        for (Variable v: t.getBinding()) {
          lvars += v.getName()+"="+v.getLiteral()+"; ";
        }
      }
      // a single result for all tuples
      ctx.addResult(opname+": "+ctx.literal("literal")+lvars);

    } else if (opname.equals("functor")) {
      Document d = xmlBuilder().newDocument();
      String res = opname+": "+ctx.literal("fpar")+"|"+ctx.text("fpar");
      // ctx.literal is the literal XML serialization
      // ctx.text is ctx.literal with any XML escapes (e.g. &lt;) removed
      for (Tuple t: ctx.getUsing()) {
        ctx.startResult(t);
        // ctx.literal/text/output/result sensitive to the current tuple
        // result/output must be literal XML serializations
        if (ctx.output("lpar", "Y1")) {
          Node nd = d.createTextNode(res+"#"+ctx.text("lpar"));
          ctx.finishResult(R3Utils.asString(nd));
        } else ctx.cancelResult();
      }
    }
  }

}
```

If you want to test your Servlet offline just derive your class from net.rewerse.i5.r3.test.Tester (instead of), as shown in the following Java excerpt:

```
package com.ricardoamador.r3.test.engine;

public class Servlet extends net.rewerse.i5.r3.test.Tester {
  public static void main(String[] args) throws Exception {
```

```
    String templates = parseArgs(args);
    if (templates == null) templates = "tests.xml";
    testTemplates(templates, new Servlet());
  }

  protected void initServices() {
    super.initServices(); // declare r3 default services if present
    registerService("com.ricardoamador.r3.test.engine.Server", "dummy");
  }
}
```

Finally you have to define your own *Language* in an OWL-DL ontology (importing the r^3 ontology) and implement your Evaluator. Example 2.2.10 shows the definition for the HTTP language included in the r^3 prototype and described in section 2.2.7 where several Java excerpts of the implementation for some of the engines included in r^3 are also presented . These Java excerpts are based on the current r^3 version (v0.20) and are bound to become outdated. Nevertheless they may help the interested reader to get a better insight on the expected features for the r^3 library, for which full documentation is to be postponed until a more stable version is reached.

Example 2.2.10 r^3 HTTP Language. r^3 *prototype include an HTTP language that includes the most common HTTP functionality.*

```
<rdf:RDF
    xmlns="http://rewerse.net/I5/NS/r3/2005/eval/http#"
    xmlns:r3="http://rewerse.net/I5/NS/2006/r3#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
  xml:base="http://rewerse.net/I5/NS/r3/2005/eval/http">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://rewerse.net/I5/NS/2006/r3"/>
  </owl:Ontology>
  <r3:Language rdf:ID="http">
    <r3:defines>
      <r3:Functor rdf:ID="get">
        <r3:uses>
          <r3:FunctionalParameter rdf:ID="uri">
            <r3:in rdf:resource="#http"/>
          </r3:FunctionalParameter>
        </r3:uses>
        <r3:binds>
          <r3:LogicalParameter rdf:ID="status">
            <r3:in rdf:resource="#http"/>
          </r3:LogicalParameter>
        </r3:binds>
        <r3:binds>
          <r3:LogicalParameter rdf:ID="status-reason">
            <r3:in rdf:resource="#http"/>
          </r3:LogicalParameter>
        </r3:binds>
        <r3:binds>
          <r3:LogicalParameter rdf:ID="content-type">
            <r3:in rdf:resource="#http"/>
          </r3:LogicalParameter>
        </r3:binds>
```

```
    <r3:binds>
      <r3:LogicalParameter rdf:ID="content-length">
        <r3:in rdf:resource="#http"/>
      </r3:LogicalParameter>
    </r3:binds>
    <r3:in rdf:resource="#http"/>
  </r3:Functor>
</r3:defines>
<r3:defines>
  <r3:Functor rdf:ID="post">
    <r3:uses rdf:resource="#uri"/>
    <r3:uses>
      <r3:FunctionalParameter rdf:ID="body">
        <r3:in rdf:resource="#http"/>
      </r3:FunctionalParameter>
    </r3:uses>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:binds>
      <r3:LogicalParameter rdf:ID="soapaction">
        <r3:in rdf:resource="#http"/>
      </r3:LogicalParameter>
    </r3:binds>
    <r3:binds rdf:resource="#content-type"/>
    <r3:binds rdf:resource="#content-length"/>
    <r3:in rdf:resource="#http"/>
  </r3:Functor>
</r3:defines>
<r3:defines>
  <r3:Functor rdf:ID="put">
    <r3:uses rdf:resource="#uri"/>
    <r3:uses rdf:resource="#body"/>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:binds rdf:resource="#content-type"/>
    <r3:binds rdf:resource="#content-length"/>
    <r3:in rdf:resource="#http"/>
  </r3:Functor>
</r3:defines>
<r3:defines>
  <r3:Functor rdf:ID="delete">
    <r3:uses rdf:resource="#uri"/>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:in rdf:resource="#http"/>
  </r3:Functor>
</r3:defines>
<r3:defines>
  <r3:Operator rdf:ID="transform">
    <r3:uses>
      <r3:FunctionalParameter rdf:ID="method">
        <r3:default rdf:parseType="Literal">POST</r3:default>
        <r3:in rdf:resource="#http"/>
      </r3:FunctionalParameter>
    </r3:uses>
    <r3:uses rdf:resource="#uri"/>
    <r3:takes>
      <r3:OperatorArgument rdf:ID="source">
        <r3:in rdf:resource="#http"/>
```

```
      </r3:OperatorArgument>
    </r3:takes>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:binds rdf:resource="#content-type"/>
    <r3:binds rdf:resource="#content-length"/>
    <r3:in rdf:resource="#http"/>
  </r3:Operator>
</r3:defines>
<r3:defines>
  <r3:OpaqueConstruct rdf:ID="vget">
    <r3:digs>
      <r3:OpaqueParameter rdf:ID="vuri">
        <r3:in rdf:resource="#http"/>
      </r3:OpaqueParameter>
    </r3:digs>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:binds rdf:resource="#content-type"/>
    <r3:binds rdf:resource="#content-length"/>
    <r3:in rdf:resource="#http"/>
  </r3:OpaqueConstruct>
</r3:defines>
<r3:defines>
  <r3:OpaqueConstruct rdf:ID="vpost">
    <r3:digs rdf:resource="#vuri"/>
    <r3:uses rdf:resource="#body"/>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:binds rdf:resource="#soapaction"/>
    <r3:binds rdf:resource="#content-type"/>
    <r3:binds rdf:resource="#content-length"/>
    <r3:in rdf:resource="#http"/>
  </r3:OpaqueConstruct>
</r3:defines>
<r3:defines>
  <r3:OpaqueConstruct rdf:ID="vput">
    <r3:digs rdf:resource="#vuri"/>
    <r3:uses rdf:resource="#body"/>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:binds rdf:resource="#content-type"/>
    <r3:binds rdf:resource="#content-length"/>
    <r3:in rdf:resource="#http"/>
  </r3:OpaqueConstruct>
</r3:defines>
<r3:defines>
  <r3:OpaqueConstruct rdf:ID="vdelete">
    <r3:digs rdf:resource="#vuri"/>
    <r3:binds rdf:resource="#status"/>
    <r3:binds rdf:resource="#status-reason"/>
    <r3:in rdf:resource="#http"/>
  </r3:OpaqueConstruct>
</r3:defines>
<r3:defines>
  <r3:OpaqueConstruct rdf:ID="vtransform">
    <r3:uses rdf:resource="#method"/>
    <r3:digs rdf:resource="#vuri"/>
    <r3:takes rdf:resource="#source"/>
```

```
        <r3:binds rdf:resource="#status"/>
        <r3:binds rdf:resource="#status-reason"/>
        <r3:binds rdf:resource="#content-type"/>
        <r3:binds rdf:resource="#content-length"/>
        <r3:in rdf:resource="#http"/>
      </r3:OpaqueConstruct>
    </r3:defines>
    <r3:defines>
      <r3:OpaqueConstruct rdf:ID="opaque">
        <r3:in rdf:resource="#http"/>
        <r3:digs>
          <r3:OpaqueParameter rdf:ID="literal">
            <r3:in rdf:resource="#http"/>
          </r3:OpaqueParameter>
        </r3:digs>
        <r3:binds rdf:resource="#status"/>
        <r3:binds>
          <r3:LogicalParameter rdf:ID="status-class">
            <r3:in rdf:resource="#http"/>
          </r3:LogicalParameter>
        </r3:binds>
        <r3:binds rdf:resource="#status-reason"/>
      </r3:OpaqueConstruct>
    </r3:defines>
    <r3:defines rdf:resource="#method"/>
    <r3:defines rdf:resource="#uri"/>
    <r3:defines rdf:resource="#body"/>
    <r3:defines rdf:resource="#status"/>
    <r3:defines rdf:resource="#status-class"/>
    <r3:defines rdf:resource="#status-reason"/>
    <r3:defines rdf:resource="#vuri"/>
    <r3:defines rdf:resource="#literal"/>
    <r3:defines rdf:resource="#source"/>
    <r3:defines rdf:resource="#soapaction"/>
    <r3:defines rdf:resource="#content-type"/>
    <r3:defines rdf:resource="#content-length"/>
  </r3:Language>
</rdf:RDF>
```

2.2.7 r^3 Component Languages

The r^3 prototype includes, not only the main ECA rule engine, but also, several expression sub-engines supporting some particular rule *component Language*s. All these languages have been integrated using the r^3 development library. The *component Language*s currently integrated in the prototype are[26] `http`, `prova`, `xcerpt`, `xchange`, `xquery`, `xpath` and `util`[27]. The OWL-DL ontologies[28] (and Java classes[29]) that currently define (and implement) these *Language*s are available online. Together with the description of each language, an illustrative set of examples is included here, and additionally some Java excerpts of the engine implementation showing how the r^3 library is used.

[26] `http://rewerse.net/I5/NS/r3/2005/eval/<name>#<name>`

[27] An expression engine for (a demonstrative subset of) the SNOOP [20] event algebra also exists [9]. This latter *Language* was implemented using a previous r^3 version (v0.02) and still has to be upgraded to the current r^3 version.

[28] `http://rewerse.net/I5/NS/r3/2005/eval/<name>`

[29] `net.rewerse.i5.r3.eval.<name>.Evaluator`

HTTP support. The `http` *Language defines* a set of *Functor*s, viz. `get`, `post`, `put` and `delete`. Any of these constructs *uses* an absolute request `uri` and *binds* a response `status` and `status-reason`. Additionally each of `put` and `post` *uses* a literal `body`. Only textual (viz. text/*) and XML application (viz. application/xml, application/*+xml) response content-types are supported and the response body is returned after being converted into an XML fragment according to the actual content-type (e.g. text/plain yields an XML text node). Any other response content-type is taken as an error, pretty much like failing to establish a connection or getting an HTTP response status not in the ok (2xx) range (with the exception of "Not Found"/404 and "Gone"/410 that are considered a failure, returning an empty set of results). The extension of these *Functors* so that each of them *binds* a set of parameters relating to (the most common) HTTP headers, is being considered[30].

For any of the above constructs, there exists a similar *OpaqueConstruct* (with the same name prefixed with a '`v`') that, instead of using a `uri`, *digs* a `vuri`. For these *OpaqueConstruct*s, variables may also be used as URI query parameters[31] (if renamed to properly URI encoded strings prefixed with a non-encoded '`=`', variable values must not be encoded); or bound to request/response HTTP headers[32] (if renamed to strings that are valid HTTP headers with the addition of a '`:`' prefix).

Also available is an `opaque` construct that *digs* a `literal` HTTP request[33] and *binds* a response `status`, `status-reason` and `status-class`[34]. This construct does not deal with any redirections (3xx), it succeeds for any response status (contrary to the others that only succeed for 2xx response status) and fails if it is not possible to establish a connection or if an invalid response content-type is returned. It uses variables in the same way the other *OpaqueConstruct*s do.

Under consideration is a `transform` *Operator* that *takes* a `source` body (instead of a literal `body`), having a behaviour similar to the `post` and `put` constructs (to distinguish between them it *uses* a `method` parameter: `POST`, by omission, or `PUT`). Additionally, a `modified uri` event (with an optional poll `interval`) and a `modified-since` test are also being considered.

Examples:

```
<Evaluate
  xmlns="http://rewerse.net/I5/NS/2006/r3#"
  xml:base="http://rewerse.net/I5/NS/r3/2005/eval/http"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <issuer><Client/></issuer>
    <solve><Expression><is rdf:resource="#opaque"/>
        <having><Parameter>
          <is rdf:resource="#status"/>
          <boundTo>Status</boundTo>
        </Parameter></having>
        <having><Parameter><is rdf:resource="#status-class"/>
          <boundTo>StatusClass</boundTo>
        </Parameter></having>
```

[30] Actually, e.g., `post` already *binds* a `soapaction`.

[31] Unbound query parameters are included in the query without the equal character.

[32] Unbound headers are omitted in the request, and bound if present in the response. Bound headers are included in the request and checked (if present) in the response. Beware when using bound general headers (or entity headers in a POST) since these relate to a specific HTTP message (or entity body) and may have different values in the request and in the response causing the construct to fail.

[33] Any included HTTP version is ignored.

[34] `status-class` is the first digit of `status`.

```
        <having><Parameter><is rdf:resource="#status-reason"/>
          <boundTo>StatusReason</boundTo>
        </Parameter></having>
        <having><Parameter><is rdf:resource="#literal"/>
          <literal rdf:parseType="Literal">
POST http://di150.di.fct.unl.pt:15080/r3/service/prova HTTP/1.1 content-type: text/xml

<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
<soapenv:Body> <request xmlns=""
    xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/ddd/solve</subject>
  <opaque>a(X,Y)</opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="X" />
      <log:variable name="Y" />
    </log:tuple>
  </log:variable-bindings>
</request> </soapenv:Body> </soapenv:Envelope>
          </literal>
        </Parameter></having>
    </Expression></solve>
    <using><Substitution>
        <binding><Variable>
          <name>SOAPAction</name><rename>:SOAPAction</rename>
          <literal rdf:parseType="Literal"
          >"http://rewerse.net/I5/NS/2006/r3#Evaluate"</literal>
        </Variable></binding>
        <binding><Variable>
          <name>Type</name><rename>:Content-Type</rename>
        </Variable></binding>
        <binding><Variable>
          <name>Length</name><rename>:Content-Length</rename>
        </Variable></binding>
        <binding><Variable><name>Status</name></Variable></binding>
        <binding><Variable><name>StatusClass</name></Variable></binding>
        <binding><Variable><name>StatusReason</name></Variable></binding>
    </Substitution></using>
</Evaluate>
```

Java Excerpts:

```
package net.rewerse.i5.r3.eval.http;
...

public class Evaluator extends ExprEvaluator {
  ...

  protected void evaluate(Context ctx) throws Exception {
    String op = ctx.opname();
    boolean usevars = op.startsWith("v");
    String puri = usevars ? "vuri" : "uri";
    HTTPRequest req = new HTTPRequest(ctx.text("method"), ctx.text(puri), ctx.text("body"));
    String[] phs = null;
    boolean trstatus;
    if (op.equals("opaque")) {
      usevars = true;
      trstatus = false;
```

```
  req.parse(ctx.text("literal"));
} else { // op.equals("get/post/put/delete/transform")
  if (usevars) op = op.substring(1);
  trstatus = true;
  if (op.equals("transform")) {
    if (!req.method.equals("POST") && !req.method.equals("PUT"))
      throw new Exception("Operator http:transform requires http:method PUT or POST");
    throw new Exception("Operator http:[v]transform not implemented");
    // Naive view: req.body = ctx.opcomposite().arg-eval("source");
    // TODO: implement AlgebraContext with multiple results,
    //     each with its set of ?distinct? tuples to join
  } else {
    req.method = op.toUpperCase();
  }
  phs = new String[] {"soapaction", "content-type", "content-length"};
  for (String hn: phs) {
    String hv = ctx.text(hn);
    if (hv != null) req.headers.add(hn+": "+hv);
  }
}

URI u = new URI(req.uri);
String bqry = u.getRawQuery(), frag = u.getRawFragment();
for (Tuple t: ctx.getUsing()) {
  String qry = bqry;
  ArrayList<String> headers = new ArrayList<String>(req.headers);
  Hashtable<String,String> vhs = new Hashtable<String,String>();
  if (usevars) {
    for (Variable v : t.getBinding()) {
      String n = v.getName(), r = ctx.rename(n);
      char c = r.charAt(0);
      if (c != ':' && c != '=') continue;
      r = r.substring(1);
      String txt = v.getLiteral();
      if (c == ':') {
        if (txt != null) headers.add(r+": "+txt);
        vhs.put(n, r.toLowerCase());
      } else {
        if (txt != null) txt = URLEncoder.encode(txt, "UTF-8");
        qry = r+(txt == null ? "" : ("="+txt))+(qry == null ? "" : ("&"+qry));
      }
    }
  }
  u = new URI(u.getScheme(), u.getUserInfo(), u.getHost(), u.getPort(), u.getPath(), null, null);
  String uri = u.toURL().toString();
  if (qry != null) uri += "?"+qry;
  if (frag != null) uri += "#"+frag;
      SSLSocketFactory sslf = HttpsURLConnection.getDefaultSSLSocketFactory();
      HttpsURLConnection.setDefaultSSLSocketFactory(sslfactory);
      R3Utils.URLStream us;
      try {
        us = new R3Utils.URLStream(req.method, uri, headers, req.body, trstatus, true);
      } finally {
        HttpsURLConnection.setDefaultSSLSocketFactory(sslf);
      }
  if (trstatus && !us.isok()) {
    us.close();
    if (us.isnotfound()) {
      if (usevars) continue; else break; // fail
```

```
        }
        throw new Exception("Unexpected HTTP status "+us.status+ " "+us.reason);
      }
      ctx.startResult(usevars ? t : null);
      if (usevars) {
        for (String vn: vhs.keySet()) {
          String h = us.headers.get(vhs.get(vn));
          if (h != null) ctx.addBinding(vn, h);
        }
      }
      if (phs != null) {
        for (String hn: phs) {
          String hv = us.headers.get(hn);
          if (hv != null) ctx.output(hn, hv);
        }
      }
      if (!trstatus)
        ctx.output("status-class", ""+((int)(us.status/100)));
      ctx.output("status", ""+us.status);
      ctx.output("status-reason", us.reason);
      ctx.finishResult(us.asString(true).trim());
      if (!usevars) break;
    }
  }

  ...
}
```

Prova support. The `prova` *Language* included in the r^3 prototype *defines* a `native` construct that *digs* a `literal` Prova rule; and an `opaque` construct that *digs* a `literal` Prova goal and generates all the possible bindings (for the involved variables) satisfying the given goal (variable values are represented as string values[35] or XML elements/documents, when appropriate). Any of these `prova` constructs *uses* a `rulesdb` that identifies a particular rulebase (viz. Prova shell) to be used (empty by omission, denoting a default one; if given, it may be an URL to be retrieved for initialization of the rulebase).

No asynchronous functionality is currently provided, but *Functor*s for `rcvMsg` and `sendMsg` are under consideration[36].

Examples:

```
<register>
  <subject>http://dummy.nop/ddd/1</subject>
  <opaque>
  a('1',bbb).
  a(2,aaa).
  a(3,bbb).
  </opaque>
</register>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
```

[35]For what is worth, it should be mentioned that several problems remain unresolved relating to the representation of numbers vs. strings, and also that quotes (single or double) in Prova strings must be balanced and have no escape mechanism.

[36]Actually *Functor*s `rcvMsg` and `sendMsg` are implemented but restricted to using the Prova `self` protocol (which does not allow remote messages).

```
  <subject>http://dummy.nop/ddd/solve</subject>
  <opaque>a(X,Y)</opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="X" />
      <log:variable name="Y" />
    </log:tuple>
  </log:variable-bindings>
</request>

<deregister>
  <subject>http://dummy.nop/ddd/1</subject>
</deregister>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/ddd/solve</subject>
  <Expression
    xmlns="http://rewerse.net/I5/NS/2006/r3#"
    xml:base="http://rewerse.net/I5/NS/r3/2005/eval/prova"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <is rdf:resource="#opaque" />
    <having><Parameter>
      <is rdf:resource="#rulesdb" />
      <literal>http://rewerse.net/I5/r3/TST/templates.prova</literal>
    </Parameter></having>
    <having><Parameter>
      <is rdf:resource="#literal" />
      <literal>a(X)</literal>
    </Parameter></having>
  </Expression>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="X" />
    </log:tuple>
  </log:variable-bindings>
</request>
```

Java Excerpts:

```
package net.rewerse.i5.r3.eval.prova;
...

public class Evaluator extends NativeEvaluator {
  ...

  protected void evaluate(Context ctx) throws Exception {
    String rules = normalize(ctx.text("rulesdb"));
    String opn = ctx.opname();
    if (opn.equals("opaque")) {
      solve(ctx, rules, ctx.text("literal"));
    } else if (opn.equals("sendMsg")) {
      sendMsg(ctx, rules);
    } else { // opn.equals("rcvMsg")
      addRcvMsg(ctx, rules);
    }
  }
  protected void solve(Context ctx, String rules, String goal) throws Exception {
    // collect variables with a bad name
    ArrayList<String> badv = new ArrayList<String>();
```

```
for (String n : ctx.vars())
  if (!Character.isUpperCase(ctx.rename(n).charAt(0))) badv.add(n);
// if there are any bad variables
if (!badv.isEmpty()) {
  // compute a conflict free prefix
  String vpref = varPrefix(ctx, goal);
  // and rename those bad variables
  int vind = 0;
  for (String bn : badv) ctx.rename(bn, vpref+(vind++));
}

// build list of goals to solve
String lgoals = "";
ArrayList<Object> lobjs = new ArrayList<Object>();
for (Tuple t : ctx.getUsing()) {
  ArrayList<String> vns = new ArrayList<String>();
  ArrayList<String> vvs = new ArrayList<String>();
  for (Variable v : t.getBinding()) {
    String n = v.getName(), r = ctx.rename(n);
    Object o = asObject(v.getLiteral());
    if (o != null) {
      vns.add(r);
      vvs.add("_"+lobjs.size());
      lobjs.add(o);
    }
  }
  lgoals += ":- solve(ctxderive("+vns.toString()+", "+vvs.toString()+", "+goal+")).\n";
}

// solve goals
List lrs = null;
Communicator sh = provaShell(rules);
synchronized (sh) {
  // We do not want different evaluations to interfere with each other
  // and since we are re-using the knowledge base the different evaluations
  // must already be serialized. :-(
  // BTW, like this we can always re-use the same consultSync key. ;-)
  // For now this is enough, in the future may have to be revised...
  sh.consultSync("ctxderive(X,X,[P|As]) :- derive([P|As]).\n", "ctxderive", null);
  lrs = sh.consultSync(lgoals.toString(), "ctxderive", lobjs.toArray());
  sh.unconsultSync("ctxderive");
}

// collect bindings
for( Iterator itrs = lrs.iterator(); itrs.hasNext(); ) {
  ProvaResultSet rs = (ProvaResultSet)itrs.next();
  Exception ex = rs.getException();
  if (ex != null) throw ex;
  for( Iterator itr = rs.iterator(); itr.hasNext(); ) {
    org.mandarax.kernel.Result r = (org.mandarax.kernel.Result)itr.next();
    ctx.startResult();
    Map m = r.getResults();
    Set ks = m.keySet();
    for (Object ko : ks) {
      VariableTerm vk = (VariableTerm)ko;
      Object vo = m.get(ko);
      ConstantTerm c = (ConstantTerm)vo;
      String vn = ctx.var(vk.getName());
      if (vn == null || vn.startsWith("_")) continue; // skip new or unnamed vars
```

```
          ctx.addBinding(vn, asString(c.getObject()));
        }
        ctx.finishResult("");
      }
    }
  }

  ...

  protected void createNative(Context ctx, String id) throws Exception {
    String rules = normalize(ctx.text("rulesdb"));
    provaShell(rules).consultSync(ctx.text("literal"), id, null);
  }

  protected void freeId(Context ctx, String id) throws Exception {
    String rules = normalize(ctx.text("rulesdb"));
    boolean create = false;
    provaShell(rules, create).unconsultSync(id);
  }

  ...
}
```

Xcerpt support. The `xcerpt` *Language defines* a `native` construct that *digs* a `literal` Xcerpt rule; and an `opaque` construct that *digs* a `literal` Xcerpt query (trivially succeeds, if empty) producing several variable bindings. Any of these constructs *uses* a `rulesdb` that identifies a particular rulebase to be used (empty by omission, denoting a default one; if given, it may be an URL to be retrieved for initialization of the rulebase).

Additionally `opaque` *uses* an Xcerpt `construct` term to build and return a result for each produced tuple (empty by omission, in which case nothing is returned).

Two additional *Functor*s are also available to obtain the Xcerpt `term` or `program` corresponding to a given XML `document`.

Finally, two other constructs are currently under consideration: an `eval` *Aggregator* that *uses* an Xcerpt `construct` term to build the aggregated result based on the involved variables (if omitted, a `tupleset` of `tuple`s with the aggregated variables could be returned, for each group); and a `transform` *Operator* similar to the `opaque` construct, but that instead of evaluating a literal query against a rulebase, *takes* a `source` *Argument* and *uses* an Xcerpt `match` pattern to filter it (trivially succeeding, if empty) and possibly produce additional variable bindings (an Xcerpt `construct` term may also be provided to specify a transformation of the `source` to be returned, defaults to identity).

Examples:

```
<register>
  <subject>http://dummy.nop/xcerpt/bib</subject>
  <opaque>
CONSTRUCT
  bib [
    book [
      title [ "Dummy33" ],
      price [ "33" ]
    ],
    book [
      title [ "Dummy44" ],
```

```
      price [ "44" ]
    ],
    book [
      title [ "Dummy55" ],
      price [ "55" ]
    ]
  ]
END
  </opaque>
</register>

<register>
  <subject>http://dummy.nop/xcerpt/rev</subject>
  <opaque>
CONSTRUCT
  reviews [
    entry [
      title [ "Dummy33" ],
      price [ "66" ]
    ]
  ]
END
  </opaque>
</register>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xcerpt?ex=00</subject>
  <opaque>
    and {
      bib {{
        book {{
          title { var T  },
          price { var Pa }
        }}
      }}
      ,
      reviews {{
        entry {{
          title { var T },
          price { var Pb }
        }}
      }}
    }
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="T" />
      <log:variable name="Pa" />
      <log:variable name="Pb" />
    </log:tuple>
  </log:variable-bindings>
</request>

<deregister>
  <subject>http://dummy.nop/xcerpt/bib</subject>
</deregister>

<deregister>
  <subject>http://dummy.nop/xcerpt/rev</subject>
```

```
</deregister>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xcerpt?ex=01</subject>
  <opaque>
    in {
      resource {"file:bib.xml"},
      bib {{
        book {{
          title { var T  },
          price { var Pa }
        }}
      }}
    }
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="T" />
      <log:variable name="Pa" />
    </log:tuple>
  </log:variable-bindings>
</request>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xcerpt?ex=4</subject>
  <Expression
    xmlns="http://rewerse.net/I5/NS/2006/r3#"
    xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xcerpt"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <is rdf:resource="#opaque" />
    <having><Parameter>
      <is rdf:resource="#rulesdb" />
      <literal>http://rewerse.net/I5/r3/TST/templates.xcerpt</literal>
    </Parameter></having>
    <having><Parameter>
      <is rdf:resource="#construct" />
      <literal>
  book [
    title   [ var Title  ],
    price-a [ var PriceA ],
    optional price-b [ var PriceB ]
  ]
      </literal>
    </Parameter></having>
    <having><Parameter>
      <is rdf:resource="#literal" />
      <literal>
  or {
      bib {{
        book {{
          title { var Title  },
          price { var PriceA }
        }}
      }}
      ,
    books-with-prices [[
      book-with-prices [
        title   [ var Title  ],
        price-a [ var PriceA ],
```

```
        price-b [ var PriceB ]
      ]
    ]]
  }
      </literal>
    </Parameter></having>
  </Expression>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="Title" />
      <log:variable name="PriceA" />
      <log:variable name="PriceB" />
    </log:tuple>
  </log:variable-bindings>
</request>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xcerpt?ex=5</subject>
  <opaque>
    in {
      resource {"http://rewerse.net/I5/r3/TST/templates.xml"},
      templates {{
      group {{
        attributes {{ name { var G } }},
        description { var GD },
        template {{
          attributes {{ name { var T } }},
          description { var TD }
        }}
      }}
      }}
    }
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="G" />
      <log:variable name="T" />
    </log:tuple>
  </log:variable-bindings>
</request>
```

Java Excerpts:

```
package net.rewerse.i5.r3.eval.xcerpt;
...

public class Evaluator extends XcerptEvaluator {
  ...

  protected void evaluate(Context ctx) throws Exception {
    String opn = ctx.opname();
    if (opn.equals("program")) {
      // program(document)
      ctx.finishResult(xcerptClient().convertProgramXML2Xcerpt(
          ctx.literal("document")));
      return;
    } else if (opn.equals("term")) {
      // term(document)
      ctx.finishResult(xcerptClient().convertTermXML2Xcerpt(
```

```
        ctx.literal("document")));
    return;
  } else if (opn.equals("eval")) {
    // eval(?construct)+aggregate+groupedBy
    // TODO: query/frm must ensure groupedBy, default construct is all tuple(aggregate)
    throw new Exception("xcerpt:eval aggregator not implemented");
  } else if (opn.equals("transform")) {
    // transform(?construct, match)+takes(source)
    // TODO: init/frm must use match/(ctx*source), default construct is source
    throw new Exception("xcerpt:transform operator not implemented");
  } else { // opn.equals("opaque")
    // opaque(?rulesdb, ?construct, literal), default construct is no result
    solve(ctx, ctx.text("rulesdb"));
  }
}

void solve(Context ctx, String rules) throws Exception {
  XcerptClient cli = xcerptClient();
  rules = getRules(cli, rules);

  String tupleset = "tupleset";
  while (rules.contains(tupleset)) tupleset += "_";

  String lresult = "result";
  ArrayList<String> tpl = new ArrayList<String>();
  for (String n : ctx.vars()) {
    String r = ctx.rename(n);
    if (lresult.equals(r)) lresult += "_";
    tpl.add("optional "+r+" { var "+r+" }");
  }

  String frm = ""
    + "    "+tupleset+" {{ tuple {\n"
    + "        "+join(tpl, ",\n        ")+"\n"
    + "    } }}";
  String query = notEmpty(ctx.text("literal"));
  if (query != null) {
    query = "  and {\n"+query.trim()+",\n\n    "+frm+"\n  }";
  } else {
    query = frm;
  }

  String construct = notEmpty(ctx.text("construct"));
  if (construct != null)
    construct = lresult+" { "+construct+" }";
  construct = ""
    + "  out { resource { \"stdout:xml\" },\n"
    + "    "+tupleset+" { all tuple {\n"
    + "        "+join(construct, tpl, ",\n        ")+"\n"
    + "    } }"+"\n"
    + "  }";

  String goal = "\nGOAL\n"+construct+"\nFROM\n"+query+"\nEND\n";
  String init = "\n"
    + "CONSTRUCT\n"
    + tupleset+" ["+join(getTuples(cli, ctx), ", ")+"\n"
    + "]\n"
    + "END\n";
```

```
// System.out.println("\n> Xcerpt ######\n"+goal+init+"\n"+rules+"\n###### Xcerpt >\n"); // System.out.println(xc.convert
    String res = cli.evaluateProgram(goal+init+"\n"+rules);
// System.out.println("\n> Xcerpt ######\n"+res+"\n###### Xcerpt <\n");
    if (!res.trim().equals("<xcerpt:error xmlns:xcerpt=\"http://xcerpt.org\">no results</xcerpt:error>"))
      buildResult(res, lresult, ctx);
  }

  ...
}

package net.rewerse.i5.r3.eval.xcerpt;
...

public abstract class XcerptEvaluator extends NativeEvaluator {
  ...

  protected Collection<String> getTuples(XcerptClient cli, Context ctx) throws Exception {
    ArrayList<String> tsl = new ArrayList<String>();
    ArrayList<String> tl = new ArrayList<String>();
    for (Tuple t : ctx.getUsing()) {
      for (Variable v : t.getBinding()) {
        String n = v.getName(), r = ctx.rename(n);
        String vs = v.getLiteral();
        if (vs != null) {
          if (vs.trim().length() == 0)
            // quote to avoid problems with r being a reserved word
            vs = "\""+r+"\""+" [ \"\" ]";
          else
            // XRA: try to avoid convert
            vs = cli.convertTermXML2Xcerpt("<"+r+">"+vs+"</"+r+">");
          tl.add("\n        "+vs);
        }
      }
      tsl.add("\n    tuple ["+join(tl, ", ")+"\n    ]");
      tl.clear();
    }
    return tsl;
  }

  protected void buildResult(String res, String lresult, Context ctx) throws Exception {
    Document tmpd = xmlBuilder().parse(new ByteArrayInputStream(res.getBytes()));
    Element el = tmpd.getDocumentElement();
    NodeList lr = el.getChildNodes();
    for ( int i=0, szr=lr.getLength(); i<szr; i++ ) {
      Node nr = lr.item( i );
      if ( nr.getNodeType() != Node.ELEMENT_NODE ) continue;
      ctx.startResult();
      String r = null;
      NodeList lt = nr.getChildNodes();
      for ( int j=0, szt=lt.getLength(); j<szt; j++ ) {
        Node nt = lt.item(j);
        if ( nt.getNodeType() != Node.ELEMENT_NODE ) continue;
        String nm = nt.getLocalName();
        String vl = asString(nt, true).trim(); // Xcerpt not xml:space aware
        if (nm.equals(lresult)) {
          r = vl;
        } else {
          nm = ctx.var(nm);
          if (nm != null) ctx.addBinding(nm, vl);
```

```
        }
      }
      ctx.finishResult(r);
    }
  }
}
```

XChange support. The `xchange` *component Language defines* a `native` construct that *digs* a `literal` XChange rule; a `detect`/`on` *OpaqueConstruct* that *digs* an XChange `event`/`query`[37] and signals any events matching it; a `raise` *Functor* that *uses* a `recipient` and an `event` body to generate an XChange event (if `recipient` is omitted it generates a local event); and an `execute` *OpaqueConstruct* that *digs* an XChange `transaction` and executes it.[38]

It should be stressed that all these *ExpressionConstruct*s (even if they are only action related, e.g. `execute`) return their results asynchronously, and so any `xchange` *Client* must always provide a *notifyTo Interface* in order to receive any possible results.

Examples:

```
<register>
  <subject>http://dummy.nop/xchange/1</subject>
  <opaque>
RAISE
  "xchange":event {{
    "xchange":recipient { "http://localhost:4711" },
    blablub { got { var X }, "in" { var Y } }
  }}
ON
  var Y -> "xchange":event {{
    blablub { var X }
  }}
END
  </opaque>
</register>

<deregister>
  <subject>http://dummy.nop/xchange/1</subject>
</deregister>

<Evaluate
  xmlns="http://rewerse.net/I5/NS/2006/r3#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
  rdf:about="http://dummy.nop/xchange?id=detect">
  <solve><Expression><is rdf:resource="#detect" />
    <having><Parameter><is rdf:resource="#query" />
      <literal rdf:parseType="Literal">
"xchange":event {{
  blablub {{ }}
}}
      </literal>
```

[37] `event` is the XChange body of an atomic `query`.

[38] Alternatively, mainly for compatibility with ECA-ML, it is also available an `opaque` construct that *digs* a `literal`. This `literal` may stand for a `detect`, `raise` or `execute` using for this purpose a syntax similar to XChange rules (starting with a line containing `ON`, `RAISE` or `TRANSACTION`, respectively; containing the appropriate XChange term; and ending with a line containing `END`).

```
    </Parameter></having>
  </Expression></solve>
  <issuer><Client><notifyTo><Interface>
    <target
    >http://localhost:8080/r3/service/dumpster</target>
  </Interface></notifyTo></Client></issuer>
</Evaluate>

<deregister>
  <subject>http://dummy.nop/xchange?id=detect</subject>
</deregister>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xchange?id=detect2</subject>
  <reply-to>http://localhost:8080/r3/service/dumpster</reply-to>
  <opaque>
ON
  andthen [
    "xchange":event {{
      var Ev1 -> blablub {{ }}
    }},
    "xchange":event {{
      var Ev2 -> blablub {{ }}
    }}
  ]
END
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="Ev1"/>
      <log:variable name="Ev2"/>
    </log:tuple>
  </log:variable-bindings>
</request>

<deregister>
  <subject>http://dummy.nop/xchange?id=detect2</subject>
</deregister>

<Evaluate
  xmlns="http://rewerse.net/I5/NS/2006/r3#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
  rdf:about="http://dummy.nop/xchange?id=theevent">
  <solve><Expression><is rdf:resource="#raise" />
    <having><Parameter><is rdf:resource="#event" />
      <literal rdf:parseType="Literal"
      >blablub { "o tal do xxx" }</literal>
    </Parameter></having>
  </Expression></solve>
  <issuer><Client><notifyTo><Interface>
    <target
    >http://localhost:8080/r3/service/dumpster</target>
  </Interface></notifyTo></Client></issuer>
</Evaluate>

<Evaluate
  xmlns="http://rewerse.net/I5/NS/2006/r3#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
```

```
  xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
  rdf:about="http://dummy.nop/xchange?id=prxevent">
  <solve><Expression><is rdf:resource="#raise" />
    <having><Parameter><is rdf:resource="#event" />
      <literal rdf:parseType="Literal"
      >blablub { "o tal do xxx" }</literal>
    </Parameter></having>
  </Expression></solve>
  <issuer><Client><notifyTo><Interface>
    <target
    >http://localhost:8080/r3/service/dumpster</target>
  </Interface></notifyTo></Client></issuer>
</Evaluate>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xchange?id=mlevent</subject>
  <reply-to>http://localhost:8080/r3/service/dumpster</reply-to>
  <opaque>
RAISE
  "xchange":event {{
    "xchange":recipient { "http://localhost:4711" },
    blablub { var X }
  }}
END
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="X">o tal do zzz</log:variable>
    </log:tuple>
  </log:variable-bindings>
</request>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xchange?id=upd</subject>
  <reply-to>http://localhost:8080/r3/service/dumpster</reply-to>
  <opaque>
TRANSACTION
  or [
  in { resource {"file:travel5.xml"},
    travel {{
      delete train {{ name{var X} }},
      insert name{var X},
      currency {"EUR" replaceby "DM"}
    }}
  },
  in { resource {"file:travel5.xml"},
    travel {{
      delete name{var X},
      insert train {{ name{var X} }},
      currency {"DM" replaceby "EUR"}
    }}
  }
  ]
END
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="X">t1</log:variable>
    </log:tuple>
```

```
  </log:variable-bindings>
</request>
```

Java Excerpts:

```
package net.rewerse.i5.r3.eval.xchange;
...

public class Evaluator extends XcerptEvaluator implements XChangeListener {
  ...

  protected void evaluate(Context ctx) throws Exception {
    XChangeFullClient cli = xchangeClient();

    String detect = null, action = null;
    String opn = ctx.opname();
    String lresult = "result", lsender = null;
    if (opn.equals("on")) {
      detect = notEmpty(ctx.text("query"));
    } else if (opn.equals("detect")) {
      detect = notEmpty(ctx.text("event"));
      if (detect != null) lsender = "sender";
    } else if (opn.equals("execute")) {
      action = notEmpty(ctx.text("transaction"));
    } else if (opn.equals("raise")) {
      String recipient = notEmpty(ctx.text("recipient"));
      action = cli.buildEventRaise(recipient, notEmpty(ctx.text("event")));
    } else { // opn.equals("opaque")
      String txt = notEmpty(ctx.text("literal"));
      if (txt == null)
        throw new Exception("xchange:literal cannot be empty");
      if (txt.endsWith("END")) {
        int nl = txt.length()-"END".length();
        txt = txt.substring(0, nl).trim();
        if (txt.length() == nl)
          // no space before END
          txt = null;
      } else {
        txt = null;
      }
      if (txt == null)
        throw new Exception("xchange:literal must terminate with 'END'");
      boolean isqry = false;
      String pref = null;
      if (txt.startsWith("ON")) {
        isqry = true;
        pref = "ON";
      } else if (txt.startsWith("TRANSACTION")) {
        pref = "TRANSACTION";
      } else if (txt.startsWith("RAISE")) {
        pref = "RAISE";
      }
      if (pref != null) {
        int nl = txt.length()-pref.length();
        txt = notEmpty(txt.substring(pref.length()));
        if (txt == null || txt.length() == nl)
          // txt empty || no space after pref
          pref = null;
      }
```

```
  if (pref == null)
    throw new Exception("xchange:literal must start with 'ON', 'RAISE' or 'TRANSACTION'");
  if (isqry) detect = txt; else action = txt;
}

String tupleset = "tupleset";
String init = tupleset+" ["+join(getTuples(cli.xcerptClient(), ctx), ", ")+"\n]";
ArrayList<String> tpl = new ArrayList<String>();
for (String n : ctx.vars()) {
  String r = ctx.rename(n);
  tpl.add("optional \""+r+"\" { var "+r+" }");
  while (r.startsWith(lresult)) lresult += "_";
  while (lsender != null && r.startsWith(lsender)) lsender += "_";
}
if (lsender != null) {
  String s = notEmpty(ctx.text("sender"));
  if (s != null) s = s.trim(); else lsender = s = null;
  detect = cli.buildEventMatch(s, detect);
}
String tuple = ""
  +"        "+join(tpl, ",\n        ")+"\n";
String from = ""
  +"    "+tupleset+" {{ tuple {\n"
  +tuple
  +"    } }}";
tuple = ""
  +"    tuple {\n"
  +tuple
  +"    }";

String iid = ctx.incomplete();
String seed = cli.proxyEventMatch("id { \""+iid+"\" }, "+from);
String seedRaise = "id { \""+iid+"\" }, "+init;
ArrayList<String> rules = new ArrayList<String>();
if (detect != null) {
  rules.add(""
      +"RAISE\n"
      +cli.proxyEventRaise("event { "+
          "attributes { id { \""+iid+"\" } }, "+
          (lsender == null ? "" : "sender { var "+lsender+" }, ")+
          "result { var "+lresult+" }, "+tuple+" }")+"\n"
      +"ON\n"
      +"andthen [\n"
      +seed+",\n"
      +"var "+lresult+" ->\n"
      +detect+"\n"
      +"]\n"
      +"END\n");
} else { // action != null
  rules.add(""
      +"TRANSACTION\n"
      +"or [\n"
      +"  and [\n"
      +"    "+action+",\n"
      +cli.proxyEventRaise("done { "+
          "attributes { id { \""+iid+"\" } }, "+
          "result { \"\" }, "+tuple+" }")+"\n"
      +"  ],\n"
      +cli.proxyEventRaise("error { "+
```

```
            "attributes { id { \""+iid+"\" } } }")+"\n"
        +"]\n"
        +"ON\n"+seed+"\n"
        +"END\n");
  }
  // System.out.println(rules);
  // System.out.println(seedRaise);
  cli.registerRules(iid, rules, seedRaise);
}

protected void createNative(Context ctx, String id) throws Exception {
  xchangeClient().registerRule(id, ctx.text("literal"));
}

protected void freeId(Context ctx, String id) throws Exception {
  xchangeClient().freeRules(id);
}

public synchronized void received(String msg) {
  // public to be called by the xchangeClient() proxy
  try {
    Element el = getReceivedEvent(msg);
    if (el == null) return;
    String iid = el.getAttribute("id"), tp = el.getLocalName();
    boolean islast = !tp.equals("event");
    if (islast) xchangeClient().freeRules(iid);
    NodeList l = el.getElementsByTagName("result");
    int sz = l.getLength();
    if (sz > 1) return;
    Element result = sz == 1 ? (Element)l.item(0) : null;
    l = el.getElementsByTagName("sender");
    sz = l.getLength();
    if (sz > 1) return;
    Element sender = sz == 1 ? (Element)l.item(0) : null;
    l = el.getElementsByTagName("tuple");
    sz = l.getLength();
    if (sz > 1) return;
    Element tuple = sz == 1 ? (Element)l.item(0) : null;
    if (result == null && tuple != null) return;
    if (tuple == null && result != null) return;
    Context ctx = incompleteEvaluation(iid);
    if (ctx == null) {
      System.err.println("[r3 WARNING] Context for "+iid+" no longer active, discarded message:\n"+msg+"\n");
      return;
    }
    if (tuple != null) {
      boolean ok = true;
      XChangeFullClient cli = xchangeClient();
      ctx.startResult();
      try {
        l = tuple.getChildNodes();
        sz = l.getLength();
        for (int i=0; i<sz; i++) {
          Node nd = l.item(i);
          if ( nd.getNodeType() != Node.ELEMENT_NODE ) continue;
          String nm = nd.getLocalName();
          nm = ctx.var(nm);
          if (nm != null)
            ctx.addBinding(nm, asString(nd, true).trim()); // XChange not xml:space aware
```

```
          }
          if (sender != null) {
            String s = asString(sender, true).trim(); // XChange not xml:space aware
            if (s.equals(cli.proxyUri())) {
              ctx.cancelResult();
              ok = false;
            } else {
              ctx.output("sender", s);
            }
          }
        } catch (Exception ex) {
          ctx.cancelResult();
          throw new Exception("Result aborted", ex);
        }
        if (ok)
          ctx.finishResult(asString(result, true).trim()); // XChange not xml:space aware
      }
      ctx.notifyResults(islast);
    } catch (Exception e) {
      // XRA: log exception
      e.printStackTrace(System.out);
    }
  }

  ...
}
```

Xcerpt and XChange TCP servers. The *LanguageEngine*s that support Xcerpt and XChange languages are mainly wrappers around the existing TCP servers for both languages, so in order to use these languages the appropriate TCP servers must be running (the XChange wrapper requires also the Xcerpt TCP server).

Currently the set of TCP ports used for integrating with r^3 is fixed. The Xcerpt and XChange TCP servers must be started with the appropriate parameters, viz. '`xcerptd 15003`' and '`xchange -p4711 -nhttp://localhost:4711`'. The available implementations for Xcerpt and XChange TCP servers are research prototypes (just like r^3), so versioning is a bit elusive. Both prototypes are developed using Haskell [38], and binary versions may not be available for all platforms, or if available they might be a bit outdated.

The current version of the r^3 wrappers requires the use of the most recent version of the Xcerpt and XChange binaries (as of March 2007) for both prototypes. Please refer to the r^3 site[39] for information on how to obtain the appropriate binaries.

XQuery and XPath support. The r^3 prototype also supports XQuery and XPath as *component Language*s. For each of the two *Language*s (viz. `xquery` and `xpath`) it implements, based on Saxon 8.7 [51], an `opaque` construct that *digs* a `literal` XQuery or XPath query.

This construct returns the XML literal results of evaluating the opaque query for all the input *Substitution*s. XQuery external variable declarations (for all the variables included in the input *Substitution*s) are added if needed[40]. Additionally it is also possible to specify a `base-uri` or a context `document`[41].

[39] `http://rewerse.net/I5/r3/TST/install/`

[40] In `xpath`, generated variable declarations are always added, whereas in `xquery`, these are added only if a related error occurs during compilation of the XQuery expression.

[41] In `xpath`, any namespace prefix used in the context `document` is added to the set of static namespace prefixes

By default a `raw format` is used and literal results are returned as string values or XML elements (when appropriate); unless one explicitly *uses* a `wrap format`. In the latter case, each result is always wrapped with Saxon elements (e.g. `result:sequence / element / attribute / atomic-value`) providing details of its type and value.

Examples:

```
<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xqq</subject>
  <opaque>
  for $X in ("aaa", "bbb")
  return <doc><xxx>{$X}</xxx><yyy>{$Y}</yyy></doc>
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="Y">ccc</log:variable>
    </log:tuple>
    <log:tuple>
      <log:variable name="Y">ddd</log:variable>
    </log:tuple>
    <log:tuple>
      <log:variable name="Y">eee</log:variable>
    </log:tuple>
  </log:variable-bindings>
</request>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xqq</subject>
  <opaque>
  fn:doc($W)/*:feed/*:entry/*:link/@href
  </opaque>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="W"
      >http://code.google.com/feeds/updates.xml</log:variable>
    </log:tuple>
    <log:tuple>
      <log:variable name="W"
      >http://code.google.com/feeds/featured.xml</log:variable>
    </log:tuple>
  </log:variable-bindings>
</request>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xqq</subject>
  <Expression
    xmlns="http://rewerse.net/I5/NS/2006/r3#"
    xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xquery"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <is rdf:resource="#opaque" />
    <having><Parameter><is rdf:resource="#format" />
      <literal>wrap</literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="#base-uri" />
      <literal>http://code.google.com/feeds/</literal>
    </Parameter></having>
```

available.

```
    <having><Parameter><is rdf:resource="#document" />
      <literal rdf:parseType="Literal">
  <config xmlns="">
    <mode>escaped</mode>
    <entry>1</entry>
  </config>
      </literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="#literal" />
      <literal>
  xquery version "1.0";
  declare namespace atom = "http://purl.org/atom/ns#";
  declare variable $W external;
  let $m := fn:string(config/mode), $i := fn:number(config/entry)
  for $r in fn:doc($W)/atom:feed/atom:entry[$i]/atom:title[@mode=$m]
  return (fn:string($r), $r/@type, $r)
      </literal>
    </Parameter></having>
  </Expression>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="X">x1</log:variable>
      <log:variable name="W">updates.xml</log:variable>
    </log:tuple>
    <log:tuple>
      <log:variable name="X">x2</log:variable>
      <log:variable name="W">updates.xml</log:variable>
    </log:tuple>
    <log:tuple>
      <log:variable name="X"/>
      <log:variable name="W">featured.xml</log:variable>
    </log:tuple>
  </log:variable-bindings>
</request>

<request xmlns:log="http://www.semwebtech.org/lang/2006/logic#">
  <subject>http://dummy.nop/xqq</subject>
  <Expression
    xmlns="http://rewerse.net/I5/NS/2006/r3#"
    xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xpath"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <is rdf:resource="#opaque" />
    <having><Parameter><is rdf:resource="#format" />
      <literal>raw</literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="#base-uri" />
      <literal>http://code.google.com/feeds/</literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="#document" />
      <literal rdf:parseType="Literal">
  <config xmlns="" xmlns:atom="http://purl.org/atom/ns#">
    <!-- using xpath -->
    <!-- all namespace prefixes used here, get declared -->
    <atom:entry>1</atom:entry>
  </config>
      </literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="#literal" />
      <literal>
```

```
  for $i in config/atom:entry
  return fn:string(
    fn:doc($W)/atom:feed/atom:entry[fn:number($i)]/atom:title)
      </literal>
    </Parameter></having>
  </Expression>
  <log:variable-bindings>
    <log:tuple>
      <log:variable name="W">updates.xml</log:variable>
    </log:tuple>
    <log:tuple>
      <log:variable name="W">featured.xml</log:variable>
    </log:tuple>
  </log:variable-bindings>
</request>
```

Java Excerpts:

```
package net.rewerse.i5.r3.eval.xquery;
...

public class Evaluator extends ExprEvaluator {
  ...

  protected void evaluate(Context ctx) throws Exception {
    String buri = notEmpty(ctx.text("base-uri"));
    String xmldoc = notEmpty(ctx.literal("document"));
    String query = ctx.literal("literal");
    boolean wrap = !ctx.text("format").equals("raw");

    XQueryEvaluator exp = new XQueryEvaluator(
        autoDeclare(),
        buri, xmldoc, query, ctx.renames());

    LinkedList<String> pvars = new LinkedList<String>();
    for (String vr: exp.getParameters()) {
      String vn = ctx.var(vr);
      if (vn != null) pvars.add(vn);
    }
    Iteratable<String> restrict = new Iteratable<String>(pvars);

    for (Tuple t : ctx.getUsing(restrict)) {
      QryEvaluation eval = exp.evaluate(wrap);
      for (Variable v : t.getBinding()) {
        String n = v.getName(), r = ctx.rename(n);
        eval.setParameter(r, asObject(v.getLiteral()));
      }
      while (true) {
        String res = eval.nextResult();
        if (res == null) break;
        ctx.addResult(t, res);
      }
    }
  }

  ...
}
```

Basic utilities support. The r^3 prototype further includes the `util` *Language* that *defines* several ad-hoc utility *LanguageConstruct*s:

- A `replace-all` *Functor* with three *FunctionalParameter*s (viz. `text`, `pattern` and `replacement`) for regular expression text replacement (based on java.util.regex package[42]).
- A `text2xml` *Functor* that *uses* a `text`that is an XML serialization and returns the respective XML node (e.g. `<x/>` becomes `<x/>`).

An introspective *OpaqueConstruct* is currently under consideration (viz. `opaque`) that *digs* a `literal` RDF/XML serialization of an r^3 *Message*, i.e. a *Request*/*Terminate* is to be issued and the synchronous *Response* returned, and a *Response* is to be validated and "echoed"[43].

The `util` *Language* will probably be extended in the future. Some useful *Aggregator*s may be added (e.g. count, sum, avg, min, max). Also some basic comparison *Functor*s[44] (e.g. eq, neq, lt, lte, gt, gte) and some basic *Operator*s (e.g. and/and-then, or/or-then, not, select-when-otherwise/if-then-else) may be added. Nevertheless, the addition of many of these *Functor*s should

[42]Beware of \ and $ occurrences in the `replacement`, particularly if this parameter is *boundTo* a variable, you may need to escape them. You may be better of using the (less powerful, but safer) *Functor* `util:replace(util:text, util:old-text, util:new-text)` which is also available for literal text replacement.

[43]Eventually, echoing a *Response* seems to be the only functionality that cannot be achieved using the `http` *Language*.

[44]Extension of the `xquery` and `xpath` *Language*s to export also their functions and operators [57] may be a better solution.

Chapter 3

r^3 Use-Cases

For testing and showcasing of r^3, some scenarios more realistic than mere ad-hoc development tests have been implemented. For that, we mostly resorted to the the bioinformatics scenario and use-cases contained in the previous deliverable [8]. It is worth noting that the other scenarios and use-cases from that deliverable have been tackled by other prototypes within I5. Namely, the travel scenario has been tried with the MARS framework, and the project's node scenario with XChange. There is also an ongoing test of MARS with this latter scenario.

Chapter Structure. In this chapter we present the current state of the r^3 use-cases derived from [8]. In the first section we recapitulate the information contained in [8] briefly relating it to the r^3 use-cases. The following two sections are each dedicated to a distinct r^3 use-case, namely the Bio-Domain Broker and the PubMed Reactive Classifier.

3.1 Use-Cases Scenario

The use-cases scenario considered for testing the r^3 prototype is based on the scenario *Updates and evolution in bioinformatics data sources* described in [8].

3.1.1 Scenario Overview

In bioinformatics there are many publicly accessible data sources, which are often mirrored locally and integrated with other data. The bioinformatics use case discusses four specific data sources: PubMed, a database of 12.000.000 biomedical literature abstracts, GeneOntology, an ontology for molecular biology, with 19.000 concepts, PDB, a database with some 25.000 protein structures, and SCOP, the Structure Classification of Proteins, which groups PDB structures according to their evolutionary relationships. The scenario is concerned with mirroring these data sources locally, keeping them consistent and integrating them, and its relation and usefulness to the work being developed in working group A2 is clear. These public bioinformatic data sources can be classified as primary and secondary (i.e. derived from one or more primary data sources, e.g. SCOP or Astral[1]). The scenario, besides the four specific data sources, also considers online applications that integrate them, viz. (Gene)Ontology-based Literature

[1]The ASTRAL Compendium for Sequence and Structure Analysis, `http://astral.berkeley.edu/`

search, GoPubMed, which given the appropriate interfaces may also be considered as secondary data sources. Users of such data sources (final users or applications) keep local copies of these primary and secondary databases and often derive tertiary data sources. Keeping local and remote databases in sync and consistent is an important problem, which requires techniques to deal with evolution and reactivity.

PubMed PubMed, `http://www.pubmed.gov/`, the main biomedical literature database references over 12.000.000 abstracts. It has grown by some 500.000 in 2003 alone. Besides biology it covers fields such as medicine, nursing, dentistry, veterinary medicine, the health care system, and the preclinical sciences. PubMed contains bibliographic citations and author abstracts from more than 4,600 biomedical journals published in 70 countries. Abstracts date back to the mid-1960's. Coverage is worldwide, but most records are from English-language sources or have English abstracts. PubMed is available in XML.

PDB Another source, which is widely used in the A2 group, is PDB, `http://www.rcsb.org/pdb/`, the protein databank. PDB is a repository of the atomic coordinates of proteins and nucleic acids. PDB entries contain among others, besides the coordinates, the resolution at which the coordinates have been obtained, the authors (who submitted the data), literature references (some recorded in PubMed), the species the data is coming from. PDB is updated every week and is available as XML and flat file.

SCOP Also widely used in the A2 group, SCOP, `http://scop.mrc-lmb.cam.ac.uk/scop/`, classifies PDB structures according to their evolution. SCOP contains four main structural classes, which are refined into some 1000 structural families of proteins. SCOP is updated every 6 months and is available as flat file. Inconsistencies introduced by PDB updates, given their weekly update rate, are a possibility not to be discarded lightly.

GeneOntology GeneOntology (GO), `http://www.geneontology.org/`, is a controlled, hierarchical vocabulary. GO has been designed for the annotation of genes. It comprises over 19.000 terms organized in three sub-ontologies for cellular location, molecular function and biological process. GO was initially created to reflect gene function of fruitflies, but has expanded to encompass many other genomes as well as sequence and structure databases. The hierarchical nature of GO allows one to quickly navigate from an overview to very detailed terms. As an example, there are maximally 16 terms from the root of the ontology to the deepest and most refined leave concept in GO. GO is available in free text, XML and as database. The XML version is updated on a monthly basis. The deliverable A2-D2 contains further details on GO.

GoPubMed GoPubMed, `http://www.gopubmed.org/`, a tool developed by TU Dresden in the A2 group, uses GO to structure large amounts of relevant literature to realize the concept of *Ontology-based Literature search*[2]. GoPubMed submits keywords to PubMed, extracts GO-terms from the retrieved abstracts, and presents the relevant sub-ontology for browsing. GoPubMed has a number of advantages, e.g. users get a high-level overview of the whole search

[2]MeshPubMed, `http://www.meshpubmed.org/`, also based on PubMed and developed by TU Dresden, realizes the same concept of *Ontology-based Literature search* using the hierarchical vocabulary "Medical Subject Headings", MeSH, `http://www.nlm.nih.gov/mesh/`.

result and are not forced to view multi-dimensional and thus often incomparable articles in a one-dimensional list. The latest versions of GoPubMed also integrate with Wikipedia.

3.1.2 Scenario Use-Cases Overview

This scenario, cf. [8], includes the following two specific use-cases.

Use Case 3.1.1 (Caching and Actuality of data in GoPubMed, PubMed, and GO) *GoPubMed is a distributed application: A query entered in GoPubMed is submitted on-the-fly to the remote PubMed site, which returns relevant articles. These are then annotated by GoPubMed with relevant terms from the GO, a local copy of which is residing at the GoPubMed site. To integrate the three sources, the GoPubMed application needs to exhibit reactive behaviour. On the event of a user query, a request is sent to PubMed. On the event of an answer from PubMed, a local cache is consulted. If the abstracts are not cached, then GoPubMed sends another message to PubMed requesting the abstracts. On receipt of them, they are annotated. Finally, the results are compiled an presented to the user. Overall, there are different distributed data sources, which are communicating with each other using event-condition-action rules.*

Use Case 3.1.2 (Mirroring, Actuality, and Consistency of data in SCOP and PDB) *The original SCOP data is published on a website hosted in Cambridge. A researcher may have a copy of SCOP on his laptop besides a copy hosted at his university. The copy on the laptop is not up-to-date, so that the researcher usually uses the remote database, but when offline he is forced to use the local laptop copy. The researcher wants to transparently access SCOP and this access needs to handle the preference of the remote SCOP copy over the local SCOP copy.*

A reactive agent acts as a wrapper of the original SCOP site and data and upon the event of a new release it informs a local agent, who updates the local SCOP copy.

Updates of PDB can lead to inconsistencies as SCOP is derived from PDB and as PDB is updated weekly, while SCOP is only updated every six months. If a PDB entry gets withdrawn between two SCOP releases, then the constraint is violated that every SCOP entry should have a PDB entry it is derived from. The constraint can be satisfied if we know which PDB entry replaces a withdrawn PDB entry. Then the local SCOP copy can be updated accordingly.

Relationship to r^3 Use-Cases The work reported in the following r^3 use-cases is an attempt towards integrating the different components needed to realize the [8] use-cases summarized here, eventually identifying and anticipating future needs/limitations.

Use-case 3.1.1 embodies the concept of reactive querying where synchronicity drives the use of reactive rules. Since reactive rules have an asynchronous nature, this use-case must be considered very carefully and it is not clear at this point if a reactive model will constitute an appropriate solution to support this use-case.

The r^3 use-cases are restricted to a proof-of-concept of use-case 3.1.2. A more comprehensive coverage of the mirroring concerns contained in this use-case is to be pursued in the future, and if possible consistency concerns should also be addressed.

3.2 r^3-based Bio Domain Broker

The Bioinformatics Domain Broker (B-Domain) provides/manages personalized mirrors of the protein databank (PDB), i.e. tertiary bioinformatics data sources (cf. the use-case scenario "*Updates and evolution in bioinformatics data sources*" described in [8]). Extension of B-Domain to

other bioinformatics specific functionalities is far from excluded (e.g. a Personalization Service for the Personal Reader framework is currently under consideration).

B-Domain uses the r^3 library to implement a bioinformatics specific language[3] that currently includes three atomic constructs. An atomic event that signals the addition of *new* structures to PDB (occurs every time a new structure is added and returns the PDBID of the added structure). An atomic condition that checks if a specific PDBID *satisfies* a criterion specified using domain specific concepts. An atomic action that *store*s a PDBID in a personal repository of structures. Using this language, the maintenance of the personalized PDB mirrors is achieved by a set of reactive rules generated and loaded to an r^3 ECA engine by B-Domain, once again, using the r^3 development library.

The current implementation of B-Domain is available online and allows the user to monitor new PDB structures. Given a criterion specified at the level of the advanced query functionality available at the PDB site, a storage (viz. a personalized PDB mirror) is created to keep all the new structures that satisfy that criterion. Try it yourself at `http://di150.di.fct.unl.pt:15080/b-domain/monitor_form.jsp`:

- Define the monitor criterion[4] that new PDB entries must satisfy in order to be included in your personal mirror;
- Submit and B-Domain will create a storage for your own personal mirror and will also load the appropriate ECA rules that will ensure the appropriate monitoring of new PDB entries cf. the provided criterion;
- Bookmark the returned URL for the created storage;
- You may see the ECA rules responsible for the actual update of your storage in the r^3 ECA engine model (`http://di150.di.fct.unl.pt:15080/r3/service`), or by analyzing the latest logged requests in the r^3 dumpster (`http://di150.di.fct.unl.pt:15080/r3/service/dumpster`);
- Visit the bookmarked URL periodically to see the new PDB entries conforming to your criterion (you will have to wait for the next PDB release[5] - updated every Tuesday/Wednesday - actually containing new structures that fit your monitor criterion).

PDB provides an advanced query facility (`http://www.rcsb.org/pdb/search/advSearch.do`) which is commonly used by bioinformatic researchers to locate relevant structures. Also, every week the new structures added to PDB are listed in an RSS feed (`http://www.rcsb.org/pdb/rss/LastLoad`). B-Domain combines these two functionalities providing an infra-structure by which researchers may define their own personalized mirrors of PDB. These personalized mirrors should contain all the new structures relevant for each researcher, and for each new structure (depending on its "kind") mirror the relevant information (either from PDB or other sources, e.g. PubMed).

The current implementation allows a researcher to create a storage (viz. personalized mirror) by specifying a criterion to filter the relevant new structures. This criterion is currently specified by an XML document (cf. `http://di150.di.fct.unl.pt:15080/b-domain/schemas/2007/`

[3] `http://di150.di.fct.unl.pt:15080/b-domain/schemas/2007/b-domain.owl`

[4] An example of a monitor criterion is available at `http://di150.di.fct.unl.pt:15080/b-domain/schemas/examples/criteriaExample2.xml`.

[5] RCSB Protein Data Bank - This week's new structures: `http://www.rcsb.org/pdb/rss/LastLoad`.

`criteria.xsd`) and mimics some of the PDB advanced query parameters. Upon submission of a criterion, an URL is returned. The researcher may later use this URL to browse to its personalized mirror.

The currently available set of PDB advanced query parameters is only a restricted set of the full set available at PDB, and is meant primarily as a proof-of-concept. Nevertheless a complete survey of the existing parameters has been conducted and work is on the way towards defining an OWL ontology to be used as a model for defining the criteria instead of the current XML Schema. The final goal is not to develop a B-Domain front-end for the PDB advanced query functionality, but instead to make this functionality available to be used (at an RDF level) in different front-ends allowing the creation (and browsing) of personalized mirrors. As possible front-ends, two main options are currently under consideration, viz. the Chemera[6] application and the Personal Reader[7] framework, both developed within REWERSE by members of WGA2 and WGA3, respectively).

The personalized mirrors are maintained through the use of appropriate reactive rules. For this, B-Domain provides a domain specific language and generates reactive rules realizing a (more or less) complex plan that is responsible for detecting relevant updates and retrieve the relevant information to be stored/updated. Currently the plans generated by B-Domain are far too simple (cf. example 3.2.1) for the intended use. They store a simple link for the PDB page dedicated to the new structure. These plans are to be improved with actual replication of information available not only at PDB but also at SCOP and PubMed.

Example 3.2.1 *To illustrate the kind of plans that B-Domain may generate, consider the following very simple plan formed by two rules that use the B-Domain language and that are chained together by an XChange event.*

```
ex:r1 a :ECARule;
  :event [ a :Expression;
    :is b-domain:newStructure;
    :boundTo [ a :Variable; ;name "PDBID" ] ];
  :test [ a :Expression;
    :is b-domain:satisfies;
    :having [ a :Parameter;
      :is b-domain:pdbId; :boundTo "PDBID" ];
    :having [ a :Parameter;
      :is b-domain:criteria; :boundTo "MyCriterion" ] ];
  :action [ a :Expression;
    :is xchange:raise;
    :having [ a :Parameter;
      :is xchange:event;
      :literal """
        newpdb {
          id {var PDBID},
          crit{var MyCriterion},
          for{var MyStorage}
        }""" ] ].
ex:r2 a :ECARule;
  :event [ a :Expression;
    :is xchange:detect;
    :having [ a :Parameter;
      :is xchange:event;
      :literal "newpdb {{ id {var Id}, for{var St} }}" ] ];
```

[6] http://www.cqfb.fct.unl.pt/bioin/chemera/Chemera/Intro.html

[7] http://www.personal-reader.de/

```
  :action [ a :Expression;
    :is b-domain:store;
    :having [ a :Parameter;
      :is b-domain:storage; :boundTo "St" ];
    :having [ a :Parameter;
      :is b-domain:pdbId; :boundTo "Id" ] ].
```

3.2.1 Message Examples

Assuming no cascading of reactive rules, the interaction between B-Domain and the r^3 engine could be reproduced and experimented by the following examples.

Example 3.2.2 B-Domain Register (di150). *Registering B-Domain engine di150:15080. Since B-Domain is not an engine/language known to the r^3 engine it needs to be registered before any rule that uses the B-Domain language can be activated. Notice that `http://di150.di.fct.unl.pt:15080/b-domain/service/b-domain` has an RDF description available online to be fetched. The availability of such description and the auto-fetching of the* declared *resource are example of the functionalities that the r^3 library provides for you out-of-the-box.*

```
<r3:Register
 xmlns:r3="http://rewerse.net/I5/NS/2006/r3#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 rdf:about="http://dummy.nop/b-domain">
 <r3:declare rdf:resource="http://di150.di.fct.unl.pt:15080/b-domain/service/b-domain" />
 <r3:issuer><r3:Client/></r3:issuer>
</r3:Register>
```

Example 3.2.3 B-Domain Rule. *An ECA rule similar to the ones created by B-Domain.*

```
<register xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml">
 <subject>http://dummy.nop/b-domain/rules/0</subject>
 <eca:rule
   xmlns:r3="http://rewerse.net/I5/NS/2006/r3#"
   xml:base="http://b-domain.w4sys.com/schemas/2007/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <eca:variable name="STORAGE-ID">45</eca:variable>
   <eca:variable name="PDB-ID">
     <eca:event><r3:Expression><r3:is rdf:resource="b-domain#newStructure" />
     </r3:Expression></eca:event>
   </eca:variable>
   <eca:query><r3:Expression><r3:is rdf:resource="b-domain#satisfies" />
     <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#pdbId" />
       <r3:boundTo>PDB-ID</r3:boundTo>
     </r3:Parameter></r3:having>
     <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#criteria" />
       <r3:literal rdf:parseType="Literal"
><b-domain:criteria
 xmlns:b-domain="http://b-domain.w4sys.com/schemas/2007/criteria#">
 <b-domain:section>
   <b-domain:structureSummary>
     <b-domain:structureTitle>
       <b-domain:comparator>contains</b-domain:comparator>
       <b-domain:keywords>UDP-GLUCOSE 4-EPIMERASE</b-domain:keywords>
```

```
     </b-domain:structureTitle>
   </b-domain:structureSummary>
 </b-domain:section>
</b-domain:criteria></r3:literal>
     </r3:Parameter></r3:having>
   </r3:Expression></eca:query>
   <eca:action><r3:Expression><r3:is rdf:resource="b-domain#store" />
     <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#storage" />
       <r3:boundTo>STORAGE-ID</r3:boundTo>
     </r3:Parameter></r3:having>
     <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#pdbId" />
       <r3:boundTo>PDB-ID</r3:boundTo>
     </r3:Parameter></r3:having>
     <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#content" />
       <r3:literal rdf:parseType="Literal"
><div xmlns="http://www.w3.org/1999/xhtml">
New PDB structure description...</div></r3:literal>
     </r3:Parameter></r3:having>
   </r3:Expression></eca:action>
 </eca:rule>
</register>
```

Example 3.2.4 Free B-Domain Rule. *Unloading previous ECA rule.*

```
<deregister><subject>http://dummy.nop/b-domain/rules/0</subject></deregister>
```

Example 3.2.5 B-Domain Event. *Registering a B-Domain event detection.*

```
<request
   xmlns:r3="http://rewerse.net/I5/NS/2006/r3#"
   xml:base="http://b-domain.w4sys.com/schemas/2007/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/b-domain/events/0</subject>
 <reply-to>http://localhost:15080/r3/service/dumpster</reply-to>
 <r3:Expression><r3:is rdf:resource="b-domain#newStructure" /></r3:Expression>
</request>
```

Example 3.2.6 Free B-Domain Event. *Terminating previous event detection.*

```
<deregister><subject>http://dummy.nop/b-domain/events/0</subject></deregister>
```

Example 3.2.7 B-Domain Query. *Evaluating a B-Domain query.*

```
<request
   xmlns:r3="http://rewerse.net/I5/NS/2006/r3#"
   xml:base="http://b-domain.w4sys.com/schemas/2007/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/b-domain/query/0</subject>
 <r3:Expression><r3:is rdf:resource="b-domain#satisfies" />
```

```
   <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#pdbId" />
     <r3:boundTo>PDB-ID</r3:boundTo>
   </r3:Parameter></r3:having>
   <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#criteria" />
     <r3:literal rdf:parseType="Literal"
><b-domain:criteria
 xmlns:b-domain="http://b-domain.w4sys.com/schemas/2007/criteria#">
 <b-domain:section>
   <b-domain:structureSummary>
     <b-domain:structureTitle>
       <b-domain:comparator>contains</b-domain:comparator>
       <b-domain:keywords>UDP-GLUCOSE 4-EPIMERASE</b-domain:keywords>
     </b-domain:structureTitle>
   </b-domain:structureSummary>
 </b-domain:section>
</b-domain:criteria></r3:literal>
   </r3:Parameter></r3:having>
 </r3:Expression>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="PDB-ID">4HHB</log:variable>
   </log:tuple>
   <log:tuple>
     <log:variable name="PDB-ID">2C20</log:variable>
   </log:tuple>
 </log:variable-bindings>
</request>
```

Example 3.2.8 B-Domain Action. *Execute a B-Domain action.*

```
<request
   xmlns:r3="http://rewerse.net/I5/NS/2006/r3#"
   xml:base="http://b-domain.w4sys.com/schemas/2007/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/b-domain/action/0</subject>
 <r3:Expression><r3:is rdf:resource="b-domain#store" />
   <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#storage" />
     <r3:boundTo>STORAGE-ID</r3:boundTo>

   </r3:Parameter></r3:having>
   <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#pdbId" />
     <r3:boundTo>PDB-ID</r3:boundTo>
   </r3:Parameter></r3:having>
   <r3:having><r3:Parameter><r3:is rdf:resource="b-domain#content" />
     <r3:literal rdf:parseType="Literal"
><div xmlns="http://www.w3.org/1999/xhtml">
New PDB structure description...</div></r3:literal>
   </r3:Parameter></r3:having>
 </r3:Expression>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="PDB-ID">2C20</log:variable>
     <log:variable name="STORAGE-ID">43</log:variable>
   </log:tuple>
 </log:variable-bindings>
</request>
```

Example 3.2.9 B-Domain Unregister. *Deregistering B-Domain engine.*

```
<deregister><subject>http://dummy.nop/b-domain</subject></deregister>
```

3.3 r^3-based PubMed Reactive Classifier

PubMed, `http://www.pubmed.gov/`, is the main biomedical literature database, and references over 12.000.000 entries. Based on events signaling additions to PubMed, this use-case generates events detecting the relevant additions filtered according to a set of (user-defined) relevant terms.

The relevant terms and publications are maintained in a Prova rulebase, based on the occurrence of events stating new publications added to PubMed and new or no longer relevant terms.

A publication is deemed relevant if it contains, according to PubMed eSearch utility, any of the supplied relevant terms. For each relevant publication information is retrieved (and mirrored) for inclusion in the generated events. Additionally events are also generated signaling any publication that becomes irrelevant due to the removal of a relevant term.

3.3.1 Message Examples

These use-cases are available online at our development server (`http://di150:15080.di.fct.unl.pt/r3/TST/`) including the following messages.

Event Interface The event interface of the present scenario is formed only by XChange events. This scenario assumes the occurrence of 4 different kinds of events:

- new_pubmed(PMID)
- not_pubmed(PMID)
- new_relevant(Term)
- not_relevant(Term)

The form in which these occurrences are actually generated is outside the scope of the scenario. The first two could be generated by PubMed itself (or by a broker application monitoring the PubMed site), whereas the other two could be generated by an application allowing a user to define the relevant set of terms (or by a broker application monitoring the Gene Ontology site, `http://www.geneontology.org/`, if one wishes to include as relevant all Gene Ontology terms). For the purpose of this scenario, and especially of running it as demo which otherwise would have to wait until new entries are actually added in PubMed, all of them are manually generated with the help of the following xchange:raise examples.

Based on these occurrences, the scenario generates 2 different kinds of events:

- new_pubmed_relevant(PMID, Summary, Term)
- not_pubmed_relevant(PMID, Summary, Term)

Such generated events are to be consumed by anyone interested. Here, they may be caught by the following xchange:detect examples (redirected to the dumpster). Other possibilities would be to e.g. send an email with notification.

Example 3.3.1 Raise Relevant Term. *Generate an occurrence stating that a new relevant term is to be considered, providing the new relevant term.*

When testing this use-case, please be careful when choosing the values in order to avoid overloading NCBI's servers. Choose relevant terms that are restrictive enough, for instance DO NOT choose 'cancer' as a relevant term (you will get almost 2 million entries). At the time of this writing, there is a single entry (viz. 15980585) in PubMed matching the term in this example (viz. "gene ontology levamisole").

```
<Evaluate
 rdf:about="http://dummy.nop/events/new_relevant"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#raise" />
   <having><Parameter><is rdf:resource="#event" />
     <literal rdf:parseType="Literal"
     >new_relevant { "gene ontology levamisole" }</literal>
   </Parameter></having>
 </Expression></solve>
 <issuer><Client><notifyTo><Interface>
   <target
   >http://localhost:15080/r3/service/dumpster</target>
 </Interface></notifyTo></Client></issuer>
</Evaluate>
```

Example 3.3.2 Raise Dropped PubMed Entry. *Generate an occurrence stating that an entry has been dropped from PubMed, providing the identifier of the dropped entry.*

```
<Evaluate
 rdf:about="http://dummy.nop/events/not_pubmed"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#raise" />
   <having><Parameter><is rdf:resource="#event" />
     <literal rdf:parseType="Literal"
     >not_pubmed { "15980585" }</literal>
   </Parameter></having>
 </Expression></solve>
 <issuer><Client><notifyTo><Interface>
   <target
   >http://localhost:15080/r3/service/dumpster</target>
 </Interface></notifyTo></Client></issuer>
</Evaluate>
```

Example 3.3.3 Raise Added PubMed Entry. *Generate an occurrence stating that an entry has been added to PubMed, providing the identifier of the added entry.*

```
<Evaluate
 rdf:about="http://dummy.nop/events/new_pubmed"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#raise" />
   <having><Parameter><is rdf:resource="#event" />
     <literal rdf:parseType="Literal"
     >new_pubmed { "15980585" }</literal>
   </Parameter></having>
 </Expression></solve>
 <issuer><Client><notifyTo><Interface>
   <target
   >http://localhost:15080/r3/service/dumpster</target>
 </Interface></notifyTo></Client></issuer>
</Evaluate>
```

Example 3.3.4 Raise Irrelevant Term. *Generate an occurrence stating that a term is to be considered as no longer relevant, providing the previously relevant term.*

```
<Evaluate
 rdf:about="http://dummy.nop/events/not_relevant"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#raise" />
   <having><Parameter><is rdf:resource="#event" />
     <literal rdf:parseType="Literal"
     >not_relevant { "gene ontology levamisole" }</literal>
   </Parameter></having>
 </Expression></solve>
 <issuer><Client><notifyTo><Interface>
   <target
   >http://localhost:15080/r3/service/dumpster</target>
 </Interface></notifyTo></Client></issuer>
</Evaluate>
```

Example 3.3.5 Detect Relevant PubMed Entry. *Detect, in XChange, relevant terms found in PubMed entries, providing:*

- *the identifier of the PubMed entry,*
- *its summary and*
- *the relevant term found.*

In general, such an event may be caused by new PubMed entries or by the addition of new relevant terms. In this scenario, we do not distinguish whether this is caused by one or the other reason.

If you have all the Reactive Rules of this scenario in place (viz. Registered/Loaded), with this expression you will be able to find such events in the "dumpster".

```
<Evaluate
 rdf:about="http://dummy.nop/events/new_pubmed_relevant"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#detect" />
   <with><Variable><name>ID</name></Variable></with>
   <with><Variable><name>S</name></Variable></with>
   <with><Variable><name>T</name></Variable></with>
   <having><Parameter><is rdf:resource="#event" />
     <literal rdf:parseType="Literal"
     >new_pubmed_relevant {
       entry { var ID },
       summary { var S },
       term { var T }
     }</literal>
   </Parameter></having>
 </Expression></solve>
 <using><Substitution>
   <binding><Variable><name>ID</name></Variable></binding>
   <binding><Variable><name>S</name></Variable></binding>
   <binding><Variable><name>T</name></Variable></binding>
 </Substitution></using>
 <issuer><Client><notifyTo><Interface>
   <target
   >http://localhost:15080/r3/service/dumpster</target>
 </Interface></notifyTo></Client></issuer>
</Evaluate>
```

Example 3.3.6 Detect Irrelevant PubMed Entry. *Detect relevant terms found in PubMed entries, providing:*

- *the identifier of the PubMed entry and*
- *the no longer relevant term.*

In general, such an event may be caused by the removal of the entry from PubMed or by the term becoming irrelevant, and in this scenario we don't distinguish what was the actual cause.

If you have all the Reactive Rules of this scenario in place (viz. Registered/Loaded), with this expression you will be able to find such events in the "dumpster".

```
<request xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/events/not_pubmed_relevant</subject>
 <reply-to>http://localhost:15080/r3/service/dumpster</reply-to>
 <Expression
     xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xchange"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns="http://rewerse.net/I5/NS/2006/r3#">
   <is rdf:resource="#detect" />
   <having><Parameter><is rdf:resource="#event" />
     <literal rdf:parseType="Literal"
     >not_pubmed_relevant {
       entry { var ID },
       term { var T }
     }</literal>
   </Parameter></having>
```

```
 </Expression>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="ID" />
     <log:variable name="T" />
   </log:tuple>
 </log:variable-bindings>
</request>
```

Prova Queries. A Prova rulebase is maintained by this scenario containing:

- relevant(Word): all relevant terms
- relevant(Word, Id): all relevant terms for each relevant PubMed entry
- pubmed(Id, Details): a mirror of the details for all relevant PubMed entries

Here you will find some examples on how to query this Prova rulebase.

Example 3.3.7 Prova Predicates. *One can load the Prova rulebase with a set of predicate. In particular, in the use case, for using some of the queries exemplified there, it is useful to add some utility predicates. In fact they are required for the reactive rules included in this scenario to work properly, as shall be illustrated below.*

```
<register xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml">
 <subject>http://dummy.nop/rules/prova</subject>
 <eca:rule>
 <eca:opaque lang="http://rewerse.net/I5/NS/r3/2005/eval/prova#prova">
not(G) :- call(G), !, fail().
not([_|_]).

and(G1, G2) :- and([G1, G2]).
and([G|T]) :- call(G), and(T).
and([]).

or(G1, G2) :- or([G1, G2]).
or([G|_]) :- call(G).
or([_|T]) :- or(T).

once(G) :- call(G), !.

if(C, T) :- if(C, T, true()).
if(C, T, _) :- call(C), !, call(T).
if(_, _, E) :- call(E).

test(G) :- not(not(G)).

call(G) :- equals(G, [P|Args]), derive([P|Args]).
\% built-in derive/1 throws an exception if called
\%   with derive(G) instead of derive([P|Args])
 </eca:opaque>
 </eca:rule>
</register>
```

Example 3.3.8 PubMed Entries. *List all relevant publications in the Prova rulebase with details.*

```
<request xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/prova/details</subject>
 <opaque>pubmed(ID, S)</opaque>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="ID" />
     <log:variable name="S" />
   </log:tuple>
 </log:variable-bindings>
</request>
```

Example 3.3.9 Relevant Terms. *List all relevant terms.*

```
<request xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/prova/terms</subject>
 <opaque>relevant(T)</opaque>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="T" />
   </log:tuple>
 </log:variable-bindings>
</request>
```

Example 3.3.10 Relevant Terms for Entries. *List all relevant terms for each publication.*

```
<request xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/prova/irrelevant</subject>
 <opaque>relevant(T, ID)</opaque>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="ID" />
     <log:variable name="T" />
   </log:tuple>
 </log:variable-bindings>
</request>
```

Example 3.3.11 Irrelevant Terms for Entries. *List all irrelevant terms for each publication in the rulebase.*

```
<request xmlns:log="http://www.semwebtech.org/lang/2006/logic">
 <subject>http://dummy.nop/prova/irrelevant</subject>
 <opaque>and([relevant(T), pubmed(ID, _), not(relevant(T, ID))])</opaque>
 <log:variable-bindings>
   <log:tuple>
     <log:variable name="ID" />
     <log:variable name="T" />
   </log:tuple>
 </log:variable-bindings>
</request>
```

Querying PubMed. For online querying of its entries, PubMed supplies a set of tools: the Entrez utilities. This scenario uses two of those utilities:

- EFetch, to fetch the summary of a publication, and
- ESearch, to search for publications that refer to specific terms.

Here you will find some examples. For more information about the Entrez utilities see:

- `http://eutils.ncbi.nlm.nih.gov/entrez/query/static/eutils_help.html`

Example 3.3.12 Fetching an Entry. *The EFech Utility, given a PubMed identifier, fetches details about an entry (cf. 'rettype', e.g. abstract or citation).*

You may also try to change the URL query parameter 'retmode' (e.g. text, html, xml) and see the results (notice how content-type never changes).

```
<Evaluate
 rdf:about="http://dummy.nop/queries/efetch"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/http"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#vget" />
   <with><BoundVariable><name>ID</name><rename>=id</rename></BoundVariable></with>
   <with><BoundVariable><name>RT</name><rename>=rettype</rename></BoundVariable></with>
   <with><Variable><name>st</name></Variable></with>
   <with><Variable><name>tp</name></Variable></with>
   <having><Parameter><is rdf:resource="#vuri" />
     <literal rdf:parseType="Literal"
     >http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed</literal>
   </Parameter></having>
   <having><Parameter><is rdf:resource="#status" />
     <boundTo>st</boundTo>
   </Parameter></having>
   <having><Parameter><is rdf:resource="#content-type" />
     <boundTo>tp</boundTo>
   </Parameter></having>
 </Expression></solve>
 <using><Substitution>
   <binding><Variable><name>ID</name><rename>=id</rename>
     <literal>17727721</literal></Variable></binding>
   <binding><Variable><name>RT</name><rename>=rettype</rename>
     <literal>abstract</literal></Variable></binding>
   <binding><Variable><name>RM</name><rename>=retmode</rename>
     <literal>text</literal></Variable></binding>
   <binding><Variable><name>st</name></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>ID</name><rename>=id</rename>
     <literal>15980585</literal></Variable></binding>
   <binding><Variable><name>RT</name><rename>=rettype</rename>
     <literal>citation</literal></Variable></binding>
   <binding><Variable><name>RM</name><rename>=retmode</rename>
     <literal>text</literal></Variable></binding>
   <binding><Variable><name>st</name></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
</Evaluate>
```

Example 3.3.13 Fetching Bad IDs. *When using the EFetch utility some caution must be taken as errors are not identified by proper HTTP status and returned content type may be different from the expected.*

Try to change the URL query parameter 'retmode' (viz. xml) and notice that the returned HTTP status and content type are always, resp., 200 and text/html.

```
<Evaluate
 rdf:about="http://dummy.nop/queries/efetch"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/http"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#vget" />
   <with><BoundVariable><name>ID</name><rename>=id</rename></BoundVariable></with>
   <with><BoundVariable><name>RT</name><rename>=rettype</rename></BoundVariable></with>
   <with><Variable><name>st</name></Variable></with>
   <with><Variable><name>tp</name></Variable></with>
   <having><Parameter><is rdf:resource="#vuri" />
     <literal rdf:parseType="Literal"
     >http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed</literal>
   </Parameter></having>
   <having><Parameter><is rdf:resource="#status" />
     <boundTo>st</boundTo>
   </Parameter></having>
   <having><Parameter><is rdf:resource="#content-type" />
     <boundTo>tp</boundTo>
   </Parameter></having>
 </Expression></solve>
 <using><Substitution>
   <binding><Variable><name>ID</name><rename>=id</rename>
     <literal>99999999</literal></Variable></binding>
   <binding><Variable><name>RT</name><rename>=rettype</rename>
     <literal>citation</literal></Variable></binding>
   <binding><Variable><name>RM</name><rename>=retmode</rename>
     <literal>text</literal></Variable></binding>
   <binding><Variable><name>st</name></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>ID</name><rename>=id</rename>
     <literal>9999999a</literal></Variable></binding>
   <binding><Variable><name>RT</name><rename>=rettype</rename>
     <literal>citation</literal></Variable></binding>
   <binding><Variable><name>RM</name><rename>=retmode</rename>
     <literal>text</literal></Variable></binding>
   <binding><Variable><name>st</name></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
</Evaluate>
```

Example 3.3.14 Searching for Entries. *The ESearch utility provides a list of (identifiers for) the set of PubMed entries satisfying a search term.*

The number of returned entries may be limited with the 'retmax' URL query parameter. The total number of entries satisfying the criteria is always included in the resulting XML (viz. /eSearchResult/Count).

Notice that the returned content-type is always text/html (which is invalid here), resulting in XML characters being escaped in the result (which otherwise would be a well-formed XML document).

```
<Evaluate
 rdf:about="http://dummy.nop/queries/esearch"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/http"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#vget" />
   <with><BoundVariable><name>T</name><rename>=term</rename></BoundVariable></with>
   <with><BoundVariable><name>N</name><rename>=retmax</rename></BoundVariable></with>
   <with><Variable><name>tp</name></Variable></with>
   <having><Parameter><is rdf:resource="#vuri" />
     <literal rdf:parseType="Literal"
     >http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed</literal>
   </Parameter></having>
   <having><Parameter><is rdf:resource="#content-type" />
     <boundTo>tp</boundTo>
   </Parameter></having>
 </Expression></solve>
 <using><Substitution>
   <binding><Variable><name>T</name><rename>=term</rename>
     <literal>ontology pubmed</literal></Variable></binding>
   <binding><Variable><name>N</name><rename>=retmax</rename>
     <literal>0</literal></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>T</name><rename>=term</rename>
     <literal>ontology pubmed</literal></Variable></binding>
   <binding><Variable><name>N</name><rename>=retmax</rename>
     <literal>1</literal></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>T</name><rename>=term</rename>
     <literal>ontology pubmed</literal></Variable></binding>
   <binding><Variable><name>N</name><rename>=retmax</rename>
     <literal>5</literal></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>T</name><rename>=term</rename>
     <literal>(ontology pubmed)+AND+(15980585[uid])</literal></Variable></binding>
   <binding><Variable><name>N</name><rename>=retmax</rename>
     <literal>5</literal></Variable></binding>
   <binding><Variable><name>tp</name></Variable></binding>
 </Substitution></using>
</Evaluate>
```

Example 3.3.15 From HTML to XML. *To remove the escaping of XML characters (explained in the previous example), we resort to a text to XML utility.*

```
<Evaluate
 rdf:about="http://dummy.nop/queries/esearch-xml"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
```

```
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/util"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <solve><Expression><is rdf:resource="#text2xml" />
   <with><BoundVariable><name>T</name></BoundVariable></with>
   <having><Parameter><is rdf:resource="#text" />
     <boundTo>T</boundTo>
   </Parameter></having>
 </Expression></solve>
 <using><Substitution>
   <binding><Variable><name>T</name>
     <literal><![CDATA[&lt;?xml version="1.0"?&gt;
&lt;!DOCTYPE eSearchResult PUBLIC "-//NLM//DTD eSearchResult, 11 May 2002//EN"
 "http://www.ncbi.nlm.nih.gov/entrez/query/DTD/eSearch_020511.dtd"&gt;
&lt;eSearchResult&gt;
 &lt;Count&gt;43&lt;/Count&gt;
 &lt;RetMax&gt;0&lt;/RetMax&gt;
 &lt;RetStart&gt;0&lt;/RetStart&gt;
 &lt;IdList&gt;
 &lt;/IdList&gt;
 &lt;TranslationSet&gt;
   &lt;Translation&gt;
     &lt;From&gt;pubmed&lt;/From&gt;
     &lt;To&gt;(&amp;quot;pubmed&amp;quot;[MeSH Terms] OR pubmed[Text Word])&lt;/To&gt;
   &lt;/Translation&gt;
 &lt;/TranslationSet&gt;
 &lt;TranslationStack&gt;
   &lt;TermSet&gt;
     &lt;Term&gt;ontology[All Fields]&lt;/Term&gt;
     &lt;Field&gt;All Fields&lt;/Field&gt;
     &lt;Count&gt;2114&lt;/Count&gt;
     &lt;Explode&gt;Y&lt;/Explode&gt;
   &lt;/TermSet&gt;
   &lt;TermSet&gt;
     &lt;Term&gt;&amp;quot;pubmed&amp;quot;[MeSH Terms]&lt;/Term&gt;
     &lt;Field&gt;MeSH Terms&lt;/Field&gt;
     &lt;Count&gt;3529&lt;/Count&gt;
     &lt;Explode&gt;Y&lt;/Explode&gt;
   &lt;/TermSet&gt;
   &lt;TermSet&gt;
     &lt;Term&gt;pubmed[Text Word]&lt;/Term&gt;
     &lt;Field&gt;Text Word&lt;/Field&gt;
     &lt;Count&gt;4582&lt;/Count&gt;
     &lt;Explode&gt;Y&lt;/Explode&gt;
   &lt;/TermSet&gt;
   &lt;OP&gt;OR&lt;/OP&gt;
   &lt;OP&gt;GROUP&lt;/OP&gt;
   &lt;OP&gt;AND&lt;/OP&gt;
   &lt;OP&gt;GROUP&lt;/OP&gt;
 &lt;/TranslationStack&gt;
 &lt;QueryTranslation&gt;ontology[All Fields] AND
  (&amp;quot;pubmed&amp;quot;[MeSH Terms] OR pubmed[Text Word])
 &lt;/QueryTranslation&gt;
&lt;/eSearchResult&gt;]]></literal></Variable></binding>
 </Substitution></using>
</Evaluate>
```

Example 3.3.16 XPath Queries. *To obtain specific values from the ESearch returned XML document we resort to XPath expressions.*

This example uses a copy of an eSearchResult document actually returned by PubMed and performs three different XPath queries on it:

- *it retrieves the total Count of elements satisfying the search (to retrieve all of them –if more than 20– this number must be included in the ESearch query);*
- *it retrieves all the Id's actually returned;*
- *it builds a search term that is the disjunction of all these Id's.*

```
<Evaluate
 rdf:about="http://dummy.nop/queries/esearch-xml"
 xmlns="http://rewerse.net/I5/NS/2006/r3#"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/xpath"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <using><Substitution>
   <binding><Variable><name>Qry</name><literal
     >/eSearchResult/Count/string()</literal></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>Qry</name><literal
     >/eSearchResult/IdList/Id/string()</literal></Variable></binding>
 </Substitution></using>
 <using><Substitution>
   <binding><Variable><name>Qry</name><literal
     >string-join(for $i in /eSearchResult/IdList/Id/string()
      return concat($i,'[uid]'),'+OR+')</literal></Variable></binding>
 </Substitution></using>
 <solve><Expression><is rdf:resource="#opaque" />
   <with><BoundVariable><name>Qry</name></BoundVariable></with>
   <having><Parameter><is rdf:resource="#literal" />
     <boundTo>Qry</boundTo>
   </Parameter></having>
   <having><Parameter><is rdf:resource="#document" />
     <literal rdf:parseType="Literal">
<eSearchResult xmlns="">
 <Count>43</Count>
 <RetMax>5</RetMax>
 <RetStart>0</RetStart>
 <IdList>
   <Id>17727721</Id>
   <Id>17620146</Id>
   <Id>17488846</Id>
   <Id>17473321</Id>
   <Id>17403693</Id>
 </IdList>
 <TranslationSet>
   <Translation>
     <From>pubmed</From>
     <To>("pubmed"[MeSH Terms] OR pubmed[Text Word])</To>
   </Translation>
 </TranslationSet>
 <TranslationStack>
   <TermSet>
     <Term>ontology[All Fields]</Term>
     <Field>All Fields</Field>
     <Count>2114</Count>
     <Explode>Y</Explode>
```

```
   </TermSet>
   <TermSet>
     <Term>"pubmed"[MeSH Terms]</Term>
     <Field>MeSH Terms</Field>
     <Count>3529</Count>
     <Explode>Y</Explode>
   </TermSet>
   <TermSet>
     <Term>pubmed[Text Word]</Term>
     <Field>Text Word</Field>
     <Count>4582</Count>
     <Explode>Y</Explode>
   </TermSet>
   <OP>OR</OP>
   <OP>GROUP</OP>
   <OP>AND</OP>
   <OP>GROUP</OP>
 </TranslationStack>
 <QueryTranslation>ontology[All Fields] AND ("pubmed"[MeSH Terms] OR pubmed[Text Word])</QueryTranslation>
</eSearchResult>
   </literal></Parameter></having>
 </Expression></solve>
</Evaluate>
```

Reactive Rules. Having explained (and set) the types of XChange events involved, what is to be stored (and queried) in the Prova rulebase , and the usage of some PubMed utilities, we can now illustrate the reactive rules used in this scenario. For easing the reading of this document, we begin by describing all the rules in a compact notation, and only then detail each of them:

- Rule for Added PubMed Entry

```
on new_pubmed(ID)
 if not(pubmed(ID, _)), relevant(W),
    xpath(concat('(',$W,')+AND+(',$ID,'[uid])'))^Q,
    xpath(/eSearchResult/IdList/Id/string(),
       text2xml(eSearch(Q, 1)))^ID
   raise search_result(entry(ID), terms(*term(W)))
```

- Rules for Relevant Term

```
on new_relevant(W)
 if not(relevant(W))
   raise found_relevant(W)
on found_relevant(W)
 do assert(relevant(W))
 if xpath(/eSearchResult/Count/string(),
       text2xml(eSearch(W, 0)))^N,
    xpath(/eSearchResult/IdList/Id/string(),
       text2xml(eSearch(W, N)))^ID
   raise search_result(entry(ID), terms(*term(W)))
```

- Rule for Fetching PubMed Entry

```
on search_result(entry(ID), TERMS)
 if pubmed(ID, S)
   raise search_xresult(entry(ID), summary(S), TERMS)
 if not(pubmed(ID, _)), eFetch(ID)^S
   raise search_xresult(entry(ID), summary(S), TERMS)
```

- Rules for Processing Search Results

```
on search_xresult(entry(ID), summary(S), ...)
 if not(pubmed(ID, _)) do assert(pubmed(ID, S))
on search_xresult(entry(ID), terms(*term(W)), ...)
 do assert(relevant(WORD, ID))
on search_xresult(entry(ID), summary(S), terms(*term(W)))
 raise new_pubmed_relevant(entry(ID), summary(S), term(W))
```

Example 3.3.17 Rule for Added PubMed Entry. *Upon a new PubMed entry: - search for relevant terms in it and - raise the search_result.*

```
on new_pubmed(ID)
 if not(pubmed(ID, _)), relevant(W),
    xpath(concat('(',$W,')+AND+(',$ID,'[uid])'))^Q,
    xpath(/eSearchResult/IdList/Id/string(),
       text2xml(eSearch(Q, 1)))^ID
   raise search_result(entry(ID), terms(*term(W)))
```

```
<register
 xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
 <subject>http://dummy.nop/rules/new_pubmed</subject>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     new_pubmed { var ID }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>not(pubmed(ID, _))</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:query>
```

```
<eca:query>
  <Expression><is rdf:resource="prova#opaque" />
    <having><Parameter><is rdf:resource="prova#literal" />
      <literal>relevant(WORD)</literal>
    </Parameter></having>
    <with><Variable><name>WORD</name></Variable></with>
  </Expression>
</eca:query>

<eca:query>
  <Expression><is rdf:resource="xpath#opaque" />
    <with><BoundVariable><name>ID</name></BoundVariable></with>
    <with><BoundVariable><name>WORD</name></BoundVariable></with>
    <having><Parameter><is rdf:resource="xpath#literal" />
      <literal>concat('(',$WORD,')+AND+(',$ID,'[uid])')</literal>
    </Parameter></having>
    <boundTo>SEARCH</boundTo>
  </Expression>
</eca:query>
<eca:query>
  <Expression><is rdf:resource="http#vget" />
    <with><BoundVariable><name>SEARCH</name><rename>=term</rename></BoundVariable></with>
    <with><BoundVariable><name>one</name><rename>=retmax</rename>
      <literal>1</literal></BoundVariable></with>
    <having><Parameter><is rdf:resource="http#vuri" />
      <literal rdf:parseType="Literal"
      >http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed</literal>
    </Parameter></having>
    <boundTo>LST_</boundTo>
  </Expression>
</eca:query>
<eca:query>
  <Expression><is rdf:resource="util#text2xml" />
    <with><BoundVariable><name>LST_</name></BoundVariable></with>
    <having><Parameter><is rdf:resource="util#text" />
      <boundTo>LST_</boundTo>
    </Parameter></having>
    <boundTo>LST</boundTo>
  </Expression>
</eca:query>
<eca:query>
  <Expression><is rdf:resource="xpath#opaque" />
    <with><BoundVariable><name>LST</name></BoundVariable></with>
    <having><Parameter><is rdf:resource="xpath#literal" />
      <literal>/eSearchResult/IdList/Id/string()</literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="xpath#document" />
      <boundTo>LST</boundTo></Parameter></having>
    <boundTo>ID</boundTo>
  </Expression>
</eca:query>

<eca:action>
  <Expression><is rdf:resource="xchange#raise"/>
    <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
    search_result { entry { var ID }, terms { all term { var WORD } } }
    </literal></Parameter></having>
    <with><BoundVariable><name>ID</name></BoundVariable></with>
    <with><BoundVariable><name>WORD</name></BoundVariable></with>
```

```
   </Expression>
 </eca:action>
</eca:rule>

</register>
```

Example 3.3.18 Rules for Relevant Term. *Upon a new relevant term:*

- *search for PubMed entries containing it and*
- *raise the search_result for each entry found.*

```
on new_relevant(W)
 if not(relevant(W))
   raise found_relevant(W)
on found_relevant(W)
 do assert(relevant(W))
 if xpath(/eSearchResult/Count/string(),
       text2xml(eSearch(W, 0)))^N,
    xpath(/eSearchResult/IdList/Id/string(),
       text2xml(eSearch(W, N)))^ID
   raise search_result(entry(ID), terms(*term(W)))
```

```
<register
 xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
 <subject>http://dummy.nop/rules/new_relevant</subject>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     new_relevant { var WORD }
     </literal></Parameter></having>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>not(relevant(WORD))</literal>
     </Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
   </Expression>
 </eca:query>

 <eca:action>
   <Expression><is rdf:resource="xchange#raise" />
     <having><Parameter><is rdf:resource="xchange#event" />
       <literal>found_relevant { var WORD }</literal>
```

```
    </Parameter></having>
    <with><BoundVariable><name>WORD</name></BoundVariable></with>
  </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
  <Expression><is rdf:resource="xchange#detect"/>
    <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
    found_relevant { var WORD }
    </literal></Parameter></having>
    <with><Variable><name>WORD</name></Variable></with>
  </Expression>
 </eca:event>

 <eca:action>
  <Expression><is rdf:resource="prova#opaque" />
    <having><Parameter><is rdf:resource="prova#literal" />
      <literal>assert(relevant(WORD))</literal>
    </Parameter></having>
    <with><BoundVariable><name>WORD</name></BoundVariable></with>
  </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
  <Expression><is rdf:resource="xchange#detect"/>
    <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
    found_relevant { var WORD }
    </literal></Parameter></having>
    <with><Variable><name>WORD</name></Variable></with>
  </Expression>
 </eca:event>

 <eca:query>
  <Expression><is rdf:resource="http#vget" />
    <with><BoundVariable><name>WORD</name><rename>=term</rename></BoundVariable></with>
    <with><BoundVariable><name>zero</name><rename>=retmax</rename>
      <literal>0</literal></BoundVariable></with>
    <having><Parameter><is rdf:resource="http#vuri" />
      <literal>http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed</literal>
    </Parameter></having>
    <boundTo>CNT_</boundTo>
  </Expression>
 </eca:query>

 <eca:query>
  <Expression><is rdf:resource="util#text2xml" />
    <with><BoundVariable><name>CNT_</name></BoundVariable></with>
    <having><Parameter><is rdf:resource="util#text" />
      <boundTo>CNT_</boundTo></Parameter></having>
    <boundTo>CNT</boundTo>
  </Expression>
 </eca:query>
 <eca:query>
  <Expression><is rdf:resource="xpath#opaque" />
    <with><BoundVariable><name>CNT</name></BoundVariable></with>
```

```
    <having><Parameter><is rdf:resource="xpath#literal" />
      <literal>/eSearchResult/Count/string()</literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="xpath#document" />
      <boundTo>CNT</boundTo></Parameter></having>
    <boundTo>N</boundTo>
  </Expression>
</eca:query>
<eca:query>
  <Expression><is rdf:resource="http#vget" />
    <with><BoundVariable><name>WORD</name><rename>=term</rename></BoundVariable></with>
    <with><BoundVariable><name>N</name><rename>=retmax</rename></BoundVariable></with>
    <having><Parameter><is rdf:resource="http#vuri" />
      <literal rdf:parseType="Literal"
      >http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed</literal>
    </Parameter></having>
    <boundTo>LST_</boundTo>
  </Expression>
</eca:query>

<eca:query>
  <Expression><is rdf:resource="util#text2xml" />
    <with><BoundVariable><name>LST_</name></BoundVariable></with>
    <having><Parameter><is rdf:resource="util#text" />
      <boundTo>LST_</boundTo>
    </Parameter></having>
    <boundTo>LST</boundTo>
  </Expression>
</eca:query>
<eca:query>
  <Expression><is rdf:resource="xpath#opaque" />
    <with><BoundVariable><name>LST</name></BoundVariable></with>
    <having><Parameter><is rdf:resource="xpath#literal" />
      <literal>/eSearchResult/IdList/Id/string()</literal>
    </Parameter></having>
    <having><Parameter><is rdf:resource="xpath#document" />
      <boundTo>LST</boundTo></Parameter></having>
    <boundTo>ID</boundTo>
  </Expression>
</eca:query>

<eca:action>
  <Expression><is rdf:resource="xchange#raise"/>
    <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
    search_result { entry { var ID }, terms { all term { var WORD } } }
    </literal></Parameter></having>
    <with><BoundVariable><name>ID</name></BoundVariable></with>
    <with><BoundVariable><name>WORD</name></BoundVariable></with>
  </Expression>
</eca:action>
</eca:rule>

</register>
```

Example 3.3.19 Rule for Fetching PubMed Entry. *Upon a search result for a PubMed entry:*

- *fetch the entry details and*

- *complete the search result with them.*

```
on search_result(entry(ID), TERMS)
 if pubmed(ID, S)
   raise search_xresult(entry(ID), summary(S), TERMS)
 if not(pubmed(ID, _)), eFetch(ID)^S
   raise search_xresult(entry(ID), summary(S), TERMS)
```

```
<register
 xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
 <subject>http://dummy.nop/rules/fetching</subject>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_result { entry { var ID }, var TERMS -> terms {{ }} }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
     <with><Variable><name>TERMS</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression>
     <is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>pubmed(ID, PUB)</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
     <with><Variable><name>PUB</name></Variable></with>
   </Expression>
 </eca:query>

 <eca:query>
   <!-- truncate PUB to avoid a possible XChange problem related to long strings -->
   <!-- somehere between 80 and 120 XChange starts to crash :-(( -->
   <Expression><is rdf:resource="xpath#opaque" />
     <with><BoundVariable><name>PUB</name></BoundVariable></with>
     <having><Parameter><is rdf:resource="xpath#literal" />
       <literal>concat(substring(\$PUB,1,80),'...')</literal>
     </Parameter></having>
     <boundTo>S</boundTo>
   </Expression>
 </eca:query>
 <eca:action>
   <Expression><is rdf:resource="xchange#raise"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_xresult { entry { var ID }, summary { var S }, var TERMS } }
     </literal></Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
     <with><BoundVariable><name>S</name></BoundVariable></with>
     <with><BoundVariable><name>TERMS</name></BoundVariable></with>
   </Expression>
```

```
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_result { entry { var ID }, var TERMS -> terms {{ }} }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
     <with><Variable><name>TERMS</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression>
     <is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>not(pubmed(ID, _))</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:query>

 <eca:query>
   <Expression><boundTo>PUB</boundTo>
     <is rdf:resource="http#vget" />
     <having><Parameter><is rdf:resource="http#vuri" />
       <literal>http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name><rename>=id</rename></BoundVariable></with>
     <with><BoundVariable><name>mode</name><rename>=retmode</rename>
       <literal>text</literal></BoundVariable></with>
     <with><BoundVariable><name>type</name><rename>=rettype</rename>
       <literal>abstract</literal></BoundVariable></with>
   </Expression>
 </eca:query>

 <eca:query>
   <!-- truncate PUB to avoid a possible XChange problem related to long strings -->
   <!-- somehere between 80 and 120 XChange starts to crash :-(( -->
   <Expression><is rdf:resource="xpath#opaque" />
     <with><BoundVariable><name>PUB</name></BoundVariable></with>
     <having><Parameter><is rdf:resource="xpath#literal" />
       <literal>concat(substring(\$PUB,1,80),'...')</literal>
     </Parameter></having>
     <boundTo>S</boundTo>
   </Expression>
 </eca:query>
 <eca:action>
   <Expression><is rdf:resource="xchange#raise"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_xresult { entry { var ID }, summary { var S }, var TERMS }
     </literal></Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
     <with><BoundVariable><name>S</name></BoundVariable></with>
     <with><BoundVariable><name>TERMS</name></BoundVariable></with>
   </Expression>
 </eca:action>
```

```
</eca:rule>

</register>
```

Example 3.3.20 Rules for Processing Search Results. *Upon a complete search result for a PubMed entry with details:*

- *cache the details if needed,*
- *mark contained relevant terms and*
- *signal relevant terms.*

```
on search_xresult(entry(ID), summary(S), ...)
 if not(pubmed(ID, _)) do assert(pubmed(ID, S))
on search_xresult(entry(ID), terms(*term(W)), ...)
 do assert(relevant(WORD, ID))
on search_xresult(entry(ID), summary(S), terms(*term(W)))
 raise new_pubmed_relevant(entry(ID), summary(S), term(W))
```

```
<register
 xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
 <subject>http://dummy.nop/rules/processing</subject>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_xresult {{ entry { var ID }, summary { var PUB } }}
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
     <with><Variable><name>PUB</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression>
     <is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>not(pubmed(ID, _))</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:query>

 <eca:action>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>assert(pubmed(ID, PUB))</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
```

```
    <with><BoundVariable><name>PUB</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_xresult {{ entry { var ID }, terms {{ term { var WORD } }} }}
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:action>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>assert(relevant(WORD, ID))</literal>
     </Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     search_xresult { entry { var ID }, summary { var PUB }, terms {{ term { var WORD } }} }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
     <with><Variable><name>PUB</name></Variable></with>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:action>
   <Expression><is rdf:resource="xchange#raise"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     new_pubmed_relevant { entry { var ID }, summary { var PUB }, term { var WORD } }
     </literal></Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
     <with><BoundVariable><name>PUB</name></BoundVariable></with>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

</register>
```

Example 3.3.21 Rules for Dropped PubMed Entry. *Cleanup Prova rulebase and signal any dropped relevant terms for each PubMed entry.*

```
on not_pubmed(ID)
```

```
 do retract(pubmed(ID, _))
 if relevant(W, ID)
   raise not_pubmed_relevant(entry(ID), term(W))
on not_pubmed_relevant(entry(ID), term(W))
 do retract(relevant(W, ID))
```

```
<register
 xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
 <subject>http://dummy.nop/rules/not_pubmed</subject>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_pubmed { var ID }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:action>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>retract(pubmed(ID, _))</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_pubmed { var ID }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>relevant(WORD, ID)</literal>
     </Parameter></having>
     <with><Variable><name>WORD</name></Variable></with>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:query>

 <eca:action>
   <Expression><is rdf:resource="xchange#raise"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_pubmed_relevant { entry { var ID }, term { var WORD } }
     </literal></Parameter></having>
```

```
    <with><BoundVariable><name>WORD</name></BoundVariable></with>
    <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_pubmed_relevant { entry { var ID }, term { var WORD } }
     </literal></Parameter></having>
     <with><Variable><name>ID</name></Variable></with>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:action>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>retract(relevant(WORD, ID))</literal>
     </Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

</register>
```

Example 3.3.22 Rules for Irrelevant Term. *Cleanup Prova rulebase and signal any dropped relevant terms for each PubMed entry.*

```
on not_relevant(W)
 do retract(relevant(W))
 if relevant(W, ID)
   raise not_pubmed_relevant(entry(ID), term(W))
 if relevant(W, ID), ~(relevant(X, ID), ~equals(X, W))
   do retract(pubmed(ID, _))
```

```
<register
 xmlns:eca="http://www.semwebtech.org/eca/2006/eca-ml"
 xml:base="http://rewerse.net/I5/NS/r3/2005/eval/"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 >
 <subject>http://dummy.nop/rules/not_relevant</subject>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_relevant { var WORD }
     </literal></Parameter></having>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
```

```
 </eca:event>

 <eca:action>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>retract(relevant(WORD))</literal>
     </Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_relevant { var WORD }
     </literal></Parameter></having>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>relevant(WORD, ID)</literal>
     </Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
     <with><Variable><name>ID</name></Variable></with>
   </Expression>
 </eca:query>

 <eca:action>
   <Expression><is rdf:resource="xchange#raise"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_pubmed_relevant { entry { var ID }, term { var WORD } }
     </literal></Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

<eca:rule xmlns="http://rewerse.net/I5/NS/2006/r3#">
 <eca:event>
   <Expression><is rdf:resource="xchange#detect"/>
     <having><Parameter><is rdf:resource="xchange#event"/><literal rdf:parseType="Literal">
     not_relevant { var WORD }
     </literal></Parameter></having>
     <with><Variable><name>WORD</name></Variable></with>
   </Expression>
 </eca:event>

 <eca:query>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>and(relevant(WORD, ID), not(and(relevant(W, ID), not(equals(W, WORD)))))</literal>
     </Parameter></having>
     <with><BoundVariable><name>WORD</name></BoundVariable></with>
```

```
    <with><Variable><name>ID</name></Variable></with>
   </Expression>
 </eca:query>

 <eca:action>
   <Expression><is rdf:resource="prova#opaque" />
     <having><Parameter><is rdf:resource="prova#literal" />
       <literal>retract(pubmed(ID, _))</literal>
     </Parameter></having>
     <with><BoundVariable><name>ID</name></BoundVariable></with>
   </Expression>
 </eca:action>
</eca:rule>

</register>
```

Testing. Now that you have all your rules in place (assuming you have posted everything up to this point), go back to the "Event Interface" examples and use the first 4 examples.

Chek the detected events in the "dumpster" and query the Prova rulebase with the examples available in "Prova Queries".

For proving a fresh rulebase and detectors, the example also includes some cleanup and reset utilities.

Cleanup and Reset. Reset Prova rulebase and cleanup all rules and event detectors.

Just post everything (in order) and you will get a clean start.

Example 3.3.23

Resetting Prova Rulebase.

```
<request>
 <subject>http://dummy.nop/prova/reset</subject>
 <opaque>and([
   retractall(pubmed(_, _)),
   retractall(relevant(_)),
   retractall(relevant(_, _))
 ])</opaque>
</request>
```

Uninstall Prova Predicates. *Cleanup rules for Prova utility predicates.*

```
<deregister><subject>http://dummy.nop/rules/prova</subject></deregister>
```

Uninstall Rules for Added PubMed Entry, for Relevant Term, for Fetching PubMed Entry, for Processing Search Results, for Dropped PubMed Entry, for Irrelevant Term.

```
<deregister><subject>http://dummy.nop/rules/new_pubmed</subject></deregister>
<deregister><subject>http://dummy.nop/rules/new_relevant</subject></deregister>
<deregister><subject>http://dummy.nop/rules/fetching</subject></deregister>
<deregister><subject>http://dummy.nop/rules/processing</subject></deregister>
<deregister><subject>http://dummy.nop/rules/not_pubmed</subject></deregister>
<deregister><subject>http://dummy.nop/rules/not_relevant</subject></deregister>
```

Uninstall Detect Relevant PubMed Entry, and Irrelevant PubMed Entry.

```
<deregister><subject>http://dummy.nop/rules/new_pubmed_relevant</subject></deregister>
<deregister><subject>http://dummy.nop/rules/not_pubmed_relevant</subject></deregister>
```

3.3.2 Possible Extensions

Recall that the main purpose of the example here is the illustration of how to build a rulebase, and to test it as a demo. But this example suggests several ways to expand this scenario, which can be tried.

One such extension resides in the generation of new_pubmed events directly from PubMed. Since PubMed does not, at present, provide any wait of detecting changes, not even an RSS feed, such generation could only rely on periodically pooling E-Search (in a continuous-queries style). This pooling could be easily written with an ECA rule, triggered by some clock event. Of course, for a demo such automatic generation of new entries is not adequate, since it would require the demo to actualy wait for real addition of entries in PubMed.

Another possible extension is to, instead of raising output event, have some other way of signalling the user, e.g. by sending emails. One main reason for not having it in the demo has to do with malicious use that could generate many mail messages with origin in our demo server.

Yet another, more interesting, extension, is the generation of new_relevant events from GeneOntology, and making inference on that ontology for relevance of one terms given others. This would require a way to detect differences in GO, possibly optimised with some http:if-modified-since. Eventually an eca:rule may be written which must use appropriate GO URLs and possibly resort to some XML diff utility.

Chapter 4

Future Work

Being a research prototype, $\boldsymbol{r^3}$ constitutes an actual moving target, with periodic stable releases. The release described in this report, version 0.20, is one such stable releases. However, the most up-to-date information about $\boldsymbol{r^3}$ is to be found at the $\boldsymbol{r^3}$ site (`http://rewerse.net/I5/r3/`). This release, including its documentation and test examples will be available to the outside after the life of the project, and support for it will continue to be given.

A new version of the $\boldsymbol{r^3}$ ontology [2] (version 0.50) is already available, pointing the future direction of r3 (beyond the timeframe of REWERSE). Such future is expected to profit from close cooperation with the industry, namely with RuleCore [50] together with the SSRG group at Skövde, towards WIDER [10] perspectives on evolution and reactivity on the Semantic Web. Given the complementary nature of $\boldsymbol{r^3}$, MARS [44] and XChange [18], it is quite important that the integration effort between the 3 prototypes continues to be pursued.

The foundational and structural nature of the $\boldsymbol{r^3}$ ontology seems appropriate for describing rules in general (not withstanding proper extension/generalization whenever needed). For instance, it would be quite simple to provide a definition (particularly, at an algebraic level) of the Prolog language and write heterogenous Prolog rules using the $\boldsymbol{r^3}$ vocabulary. Although our present work, and the work proposed in the context of REWERSE workpackage I5, is not focused on rules in general, reactive rules very often use other types of rules, namely to model relationships of a more deductive nature (e.g. higher level constructions based on lower level ones, or events "derived" from actions). As such, it would be quite interesting (both from a general and reactive perspective) to collaborate with other communities towards achieving an $\boldsymbol{r^2}$ ontology (viz. a foundational ontology for Resourceful Rules). Also, considering the extension of the $\boldsymbol{r^3}$ ontology to higher abstraction levels (e.g. rule languages, domain/application languages, event languages, event algebras, process algebras) would be interesting to pursue.

Regarding ECA languages, the $\boldsymbol{r^3}$ ontology implicitly extends and generalizes the previously proposed general ECA framework of [5, 26, 27] (e.g. solution constraints and rule constructions) and calls for a revision of the ECA-ML markup [26] and of the general framework. Any revision of the ECA-ML markup should: consider a homogenous markup for logical variables, clearly identify syntactic sugar constructions (e.g. using XPath expressions to initialize variables), and be based upon the formal specification of the (revised) framework. This is in accordance with the ontology proposed in [7]. Also, given the $\boldsymbol{r^3}$ ontology (as an adequate abstract syntax for the framework), such a (fully) formal specification of the framework is possible to achieve without resorting to general markup guidelines and principles. Once the revised general framework is

formally specified, the $\boldsymbol{r^3}$ prototype is to be upgraded accordingly, thus supporting the final version of the ontology.

The $\boldsymbol{r^3}$ approach tries to maximize separation of concerns and to focus as much as possible on a model for reactive rule evaluation, taking into consideration (but not focusing on) the specificities of the different component languages. For instance, Semantic Web event detection/propagation is a concern for event engines (viz. atomic event detectors/matchers, event brokers or algebraic event composers); as much as Semantic Web transactions are a concern for action engines. $\boldsymbol{r^3}$ is expected to work with different event (and action) engines, regardless of the specific event detection/propagation (and transaction management) algorithms/architectures they use. $\boldsymbol{r^3}$ is not expected to enforce any particular algorithm/architecture at this level, which, of course, also means that $\boldsymbol{r^3}$ does not propose any particular solution for these research issues. $\boldsymbol{r^3}$ simply does not try to solve these issues. Instead, $\boldsymbol{r^3}$ tries to integrate external research results, by defining how they should work together and by providing an adequate environment for testing these results. It is our stance that $\boldsymbol{r^3}$ may provide a useful test bed for integration of related languages like, for instance, $\boldsymbol{XChange^{EQ}}$ [19] and GeTS [33]. Regarding such research areas/issues, and related work, $\boldsymbol{r^3}$ takes a cooperative stand; hoping to become a useful scenario where different external results (either within or outside REWERSE) may be integrated and experimented.

Bibliography

[1] José Júlio Alferes and Ricardo Amador. r3: Towards a foundational ontology for reactive rules. In *4th European Semantic Web Conference (ESWC)*, 2007. Poster at `http://www.eswc2007.org/posters.cfm`.

[2] José Júlio Alferes and Ricardo Amador. $\boldsymbol{r^3}$ - a foundational ontology for reactive rules. In *Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 4803/4804 of *Lecture Notes on Computer Science*. Springer, 2007.

[3] José Júlio Alferes, Ricardo Amador, Erik Behrends, Mikael Berndtsson, François Bry, Gihan Dawelbait, Andreas Doms, Michael Eckert, Oliver Fritzen, Wolfgang May, Paula-Lavinia Pătrânjan, Loc Royer, Franz Schenk, and Michael Schroeder. Specification of a Model, Language and Architecture for Reactivity and Evolution. Deliverable I5-D4, Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa, 2005.

[4] José Júlio Alferes, Ricardo Amador, Erik Behrends, Michael Eckert, Oliver Fritzen, Wolfgang May, Paula-Lavinia Pătrânjan, and Franz Schenk. A first prototype on evolution and behaviour at the XML-Level. Deliverable I5-D5, Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa, 2006.

[5] José Júlio Alferes, Ricardo Amador, Erik Behrends, Michael Eckert, Oliver Fritzen, Wolfgang May, Paula-Lavinia Pătrânjan, and Franz Schenk. Completion of the prototype scenario. Deliverable I5-D7, Rewerse project, 2007.

[6] José Júlio Alferes, Ricardo Amador, and Wolfgang May. A general language for Evolution and Reactivity in the Semantic Web. In *Principles and Practice of Semantic Web Reasoning (PPSWR)*, volume 3703 of *Lecture Notes on Computer Science*, pages 101–115. Springer, 2005.

[7] José Júlio Alferes, Ricardo Amador, and Wolfgang May. Reactive rule ontology: RDF/OWL level. Deliverable I5-D6, Rewerse project, 2007.

[8] José Júlio Alferes, Mikael Berndtsson, François Bry, Michael Eckert, Nicola Henze, Wolfgang May, Paula-Lavinia Pătrânjan, and Michael Schroeder. Use-cases on evolution. Deliverable I5-D2, Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa, 2005.

[9] José Júlio Alferes and Gastón E. Tagni. Implementation of a Complex Event Engine for the Web. In *Proceedings of First International Workshop on Event-driven Architecture, Processing and Systems, Chicago, USA (18th September 2006)*, 2006.

[10] Ricardo Amador and José Júlio Alferes. Web Integrated Development tools for Evolution and Reactivity (WIDER). http://www.ricardoamador.com/research/program.aspx, 2005. PhD Proposal.

[11] Jürgen Angele, Harold Boley, Jos de Bruijn, Dieter Fensel, Pascal Hitzler, Michael Kifer, Reto Krummenacher, Holger Lausen, Axel Polleres, and Rudi Studer. Web Rule Language (WRL). http://www.w3.org/Submission/2005/SUBM-WRL-20050909/.

[12] Dave Beckett. Turtle - Terse RDF Triple Language. http://www.dajobe.org/2004/01/turtle/.

[13] Erik Behrends, Oliver Fritzen, Wolfgang May, and Franz Schenk. Combining ECA Rules with Process Algebras for the Semantic Web. In *Proceedings of Second International Conference on Rules and Rule Markup Languages for the Semantic Web, Athens, Georgia, USA (10th–11th November 2006)*, 2006.

[14] Erik Behrends, Oliver Fritzen, Wolfgang May, and Daniel Schubert. An ECA Engine for Deploying Heterogeneous Component Languages in the Semantic Web. In *Proceedings of Workshop Reactivity on the Web, Munich, Germany (31st March 2006)*, LNCS, 2006.

[15] Tim Berners-Lee. Putting the Web back in Semantic Web - talk. http://www.w3.org/2005/Talks/1110-iswc-tbl/, November 2005.

[16] Harold Boley, Benjamin Grosof, Michael Sintek, Said Tabet, and Gerd Wagner. *RuleML Design*. RuleML Initiative, http://www.ruleml.org/, 2002.

[17] Harold Boley and Michael Kifer. RIF Core Design. Working Draft, W3C, http://www.w3.org/TR/2007/WD-rif-core-20070330, March 2007.

[18] F. Bry and P.-L. Pătrânjan. Reactivity on the Web: Paradigms and Applications of the Language XChange. In *20th ACM Symp. Applied Computing*. ACM, 2005.

[19] François Bry and Michael Eckert. Rule-Based Composite Event Queries: The Language $\boldsymbol{XChange^{EQ}}$ and its Semantics. In *RR'07*, volume 4524 of *LNCS*. Springer, 2007.

[20] Sharma Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data & Knowledge Engineering*, 14(1):1–26, 1994.

[21] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[22] Foundation for Intelligent Physical Agents. FIPA ACL message structure specification. Technical Report SC00061G, http://www.fipa.org, Dec. 2002.

[23] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

[24] International Organization for Standardization. *Common Logic (CL): a framework for a family of logic-based languages*, volume ISO/IEC FDIS 24707. ISO, http://common-logic.org/, May 2007.

[25] Aditya Kalyanpur, Daniel Pastor, Steve Battle, and Julian Padget. Automatic Mapping of OWL Ontologies into Java. In *Proceedings of Sixteenth International Conference on Software Engineering and Knowledge Engineering (SEKE), Banff, Canada, (20th–24th June 2004)*, 2004.

[26] Wolfgang May, José Júlio Alferes, and Ricardo Amador. Active rules in the Semantic Web: Dealing with language heterogeneity. In *International Conference on Rules and Rule Markup Languages for the Semantic Web (RuleML)*, volume 3791 of *Lecture Notes on Computer Science*, pages 30–44. Springer, 2005.

[27] Wolfgang May, José Júlio Alferes, and Ricardo Amador. An ontology- and resources-based approach to evolution and reactivity in the Semantic Web. In *Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 3761 of *Lecture Notes on Computer Science*, pages 1553–1570. Springer, 2005.

[28] Wolfgang May, José Júlio Alferes, and François Bry. Towards generic query, update, and event languages for the Semantic Web. In *Principles and Practice of Semantic Web Reasoning (PPSWR)*, volume 3208 of *Lecture Notes on Computer Science*, pages 19–33. Springer, 2004.

[29] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[30] Object Management Group. *Business Process Definition Metamodel (BPDM)*. OMG, 2003.

[31] Object Management Group. *Unified Modelling Language (UML) 2.0 Superstructure*. OMG, 2004. http://www.omg.org/cgi-bin/doc?ptc/2004-10-02.

[32] Object Management Group. *Semantics of Business Vocabulary and Business Rules (SBVR)*. OMG, 2006. http://www.omg.org/cgi-bin/doc?dtc/2006-03-02.

[33] Hans Jürgen Ohlbach. Implementation: GeTS - A Specification Language for Geo-Temporal Notions. Deliverable A1-D10-1, Rewerse project, 2005.

[34] Silvie Spreeuwenberg and Rick Gerrits. Business Rules in the Semantic Web, Are There Any or Are They Different? In *Proceedings of Summer School Reasoning Web 2006, Lisbon, Portugal (4th–8th September 2006)*, volume 4126 of *LNCS*, pages 152–163. REWERSE, 2006.

[35] Apache Axis. `http://ws.apache.org/axis/`.

[36] Apache Axis2. `http://ws.apache.org/axis2/`.

[37] DB2 universal database. `http://www.ibm.com/db2/`.

[38] Haskell - A purely functional language. `http://haskell.org/`.

[39] Jastor - Typesafe, Ontology Driven RDF Access from Java. `http://jastor.sourceforge.net/`.

[40] JavaServer Pages Technology. `http://java.sun.com/products/jsp`.

[41] Java Servlet Technology. http://java.sun.com/products/servlets.

[42] Jena Semantic Web framework for Java. http://jena.sourceforge.net/.

[43] Java message service (JMS). http://java.sun.com/products/jms/.

[44] MARS: Modular Active Rules for the Semantic Web. http://www.dbis.informatik.uni-goettingen.de/MARS/. Databases and Information Systems Group - DBIS, Institute for Informatics, Georg-August-Universität Göttingen.

[45] Web ontology language (OWL). http://www.w3.org/2004/OWL/.

[46] Pellet OWL Reasoner. http://www.mindswap.org/2003/pellet/.

[47] Resourceful Reactive Rules (r^3). http://rewerse.net/I5/r3/. Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa.

[48] Resource description framework (RDF). http://www.w3.org/RDF/.

[49] RDF/XML syntax specification (revised). http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[50] ruleCore: The Active Rule Engine. http://www.rulecore.com/.

[51] SAXON - The XSLT and XQuery Processor. http://saxon.sourceforge.net/.

[52] Simple object access protocol (SOAP). http://www.w3.org/TR/soap.

[53] Web services addressing (WS-Addressing). http://www.w3.org/Submission/ws-addressing/.

[54] Web services description language (WSDL) 1.1. http://www.w3.org/TR/wsdl.

[55] XML Schema. http://www.w3.org/XML/Schema/.

[56] W3C XML query (XQuery). http://www.w3.org/XML/Query/.

[57] XQuery 1.0 and XPath 2.0 Functions and Operators. http://www.w3.org/TR/xpath-functions/.

[58] XSL transformations (XSLT) 2.0. http://www.w3.org/TR/xslt20/.

Acknowledgements

This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).