

r^3 : A foundational ontology for reactive rules ^{*}

José Júlio Alferes and Ricardo Amador

Centro de Inteligência Artificial - CENTRIA, Universidade Nova de Lisboa
{jja, ra}@di.fct.unl.pt

Abstract. In this paper we present the r^3 ontology, a foundational ontology for reactive rules, aiming at coping with language heterogeneity at the rule (component) level. This (OWL-DL) ontology is at a low (structural) abstraction level thus fostering its extension. Although focusing on reactive rules (reactive derivation rules not excluded), the r^3 ontology defines a vocabulary that allows also for the definition of rule (component) languages to model other types of rules like production, integrity, or logical derivation rules.

1 Introduction

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 WSDL2 (as a semantically extensible service model) provide the natural underlying concepts.

The Semantic Web does not have any central structure, neither topologically nor thematically, rather it 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 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 it is really unlikely that there will be a unique language for the latter throughout the Web.

Heterogenous Reactivity. In this setting, *reactivity* and its formalization as *Event-Condition-Action (ECA) rules* provide a suitable common model because they provide

^{*} This research has been funded by the European Commission within the 6th Framework Programme project REVERSE number 506779.

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 which has a well-understood meaning: when an event (atomic or composite, the latter possibly using some event algebra for 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 [16, 17] we have proposed an ontology-based approach for describing (reactive) behavior and evolution in the Web, following the ECA paradigm. This work also defines a global architecture and general markup principles for a modular framework capable 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. The interested reader is referred to [2, 1] for additional details on the present state of this work.

Semantic Web Events. The notion of event is core to ECA rules, and in a reactive model of behaviour for the Web it needs to be freed of limitations introduced by more specific settings (e.g. active databases). Events in the Web cannot be restricted to the realization of a specific set of actions (e.g. insert, update and delete). Instead a *Web Event* is to be understood as the actual perception by a reactive system of an(y) external (or otherwise uncontrolled) occurrence, that may or may not be the result of a known action.

Logically, an event may be perceived as a temporary (non-persistent) assertion, resulting in the evolution of the knowledge base, as described in [4], through concrete (re)actions (or active deductions) that may generate new persistent assertions, invalidate existing assertions or cause additional externally perceivable occurrences (i.e. events). The notion of non-persistence of an event is of utmost importance for the Web given the humongous number of events perceivable in such a global system. This global nature also precludes any solution based on indiscriminated broadcast of events; systems interested in particular kinds of events have to express their interest to specialized event brokers. The latter may to some extent persist historical event information, but reactive rule engines have to be free of such a burden.

In a distributed environment formed of autonomous nodes, like the (Semantic) Web, ECA rules can not react to actual occurrences, only (more or less reliable) perceptions of those occurrences are generally available, and even those may sometimes go unnoticed. Nevertheless, using an eclectic mix of deductive and reactive rules, and based on different lower level perceptions, one may achieve a symbolic definition of higher level events that (fully) abstract and mimic (to the extent of the knowledge they represent)

the actual occurrences, possibly even compensating for unperceived ones through alternative or implicit perceptions; thus allowing to shift the focus to *Semantic Web Events* (like *a book has been bought online*), eventually abstracting away the intricacy of Web Events (just try to imagine how many different ways exist to perceive that a book has been bought online).

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 rules themselves. In what concerns the latter, most of the current proposals are based on XML markups (e.g. [7]); eventually relying on specific abstract syntax for defining rule semantics (e.g. [8, 14]). Markup-based approaches, as such, seem to ignore the fact that rules do not only operate on the Semantic Web, but are themselves part of it. In general (ECA) rules and their components must be communicated between different nodes, and may themselves be subject to being queried and updated, especially if one wants to reason about evolution, leading to a Semantic Web capable of dynamic behaviour according to behaviour policies. 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 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 r^3 takes a third hypothesis: ontologies, in OWL-DL, provide a suitable tool for describing language heterogeneity.

Present State. Although the examples included in [16] use “syntactical” languages in XML term markup –ECA-ML– to describe ECA rule components, as stated there, *also languages using a semantical, e.g., OWL-based representation (which have to be developed) can be used*; thus leading to fully embrace the approach proposed in [17]. To further experiment with both approaches, namely syntactic and semantic, two REVERSE WG15¹ sub-projects, aiming at developing prototypes of the proposed general ECA framework, were launched: MARS [22] and r^3 [23]. Currently, both prototypes are functional, available online, and eventually integrable through appropriate syntactic/markup transformations. The MARS project is now also evolving into the semantic level taking a flexible approach, not restricted to OWL-DL; future integration of the two prototypes is to be pursued at this semantic/ontology level. The interested reader may find additional details on both prototypes in [2, 1].

In this paper, results of the r^3 project on defining an (OWL-DL) foundational ontology for reactive rules are presented. 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 not excluded, and fruitful synergies are expected with the MARS project

¹ REVERSE WG15: Evolution and Reactivity - <http://reverse.net/I5/>

which is following an higher level approach. Although focusing on reactive rules, the r^3 ontology defines a vocabulary allowing for the definition of rule and rule component languages to model also other types of rules.

Related Work. To the best of our knowledge, to the present there are only three ontology proposals for describing rules: SWRL [13], WRL [5] and SBVR [18]. 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 results. About SBVR, it is worth mentioning, that it is targeted to describe business rules in general with an emphasis on human understanding [21] (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. [8, 7], 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) [14], in what concerns language heterogeneity, is probably the standardization effort closest to the spirit of 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.

Structure of the Paper. We start (in section 2) by introducing the r^3 ontology from a rule “taxonomy” point of view. In the following sections we detail the r^3 ontology explaining and illustrating² how to define different languages (in section 3), and how to use these languages to define heterogenous rules (in section 4). We end the paper with some conclusion and future directions of the work.

The r^3 OWL-DL ontology available at <http://rewerse.net/I5/NS/2007/r3/r3.owl> constitutes the only complete and formal definition of the r^3 ontology.

² Examples illustrating RDF models use Turtle, omit prefix declarations, and assume the r^3 namespace as the empty (‘:’) prefix. The complete set of examples presented here may also be found in RDF/XML at <http://rewerse.net/I5/NS/2007/r3/odbase07.owl>.

For the sake of readability, we have chosen to present it here using UML2 diagrams. These diagrams formally define (to the extent possible) the r^3 ontology. The explanatory text that accompanies them is neither a formal definition of the r^3 ontology, nor a substitute for the UML2 diagrams. As such, careful observation of the diagrams is required for full understanding of the work presented here.

2 An Ontology for Reactive Rules

An ontology for *reactive rules* restricted to different forms of *active rules* would be of limited expressivity in practice. Conditions used in ECA rules are quite often defined resorting to *logical derivation rules* (e.g. deductive rules that define intensional relations). Also, some form of *reactive derivation rules* is imperative so that symbolical events and actions with higher semantic value may be defined/derived; thus allowing reactive rules to actually express behaviour on a semantic level and not only basic low level reactions. Furthermore, integrity of a reactive system is frequently hard to express and maintain on a rule by rule basis (viz. using post-conditions): global *integrity rules* provide an additional orthogonal perspective and can be used together or independently of reactive rules allowing the detection of invalid states or reactions. Given all this, the r^3 ontology, although aiming at describing reactive behaviour, includes all these different kinds of rules, as shown in figure 1 where abstract rules (further detailed in section 4) are partitioned according to their components.

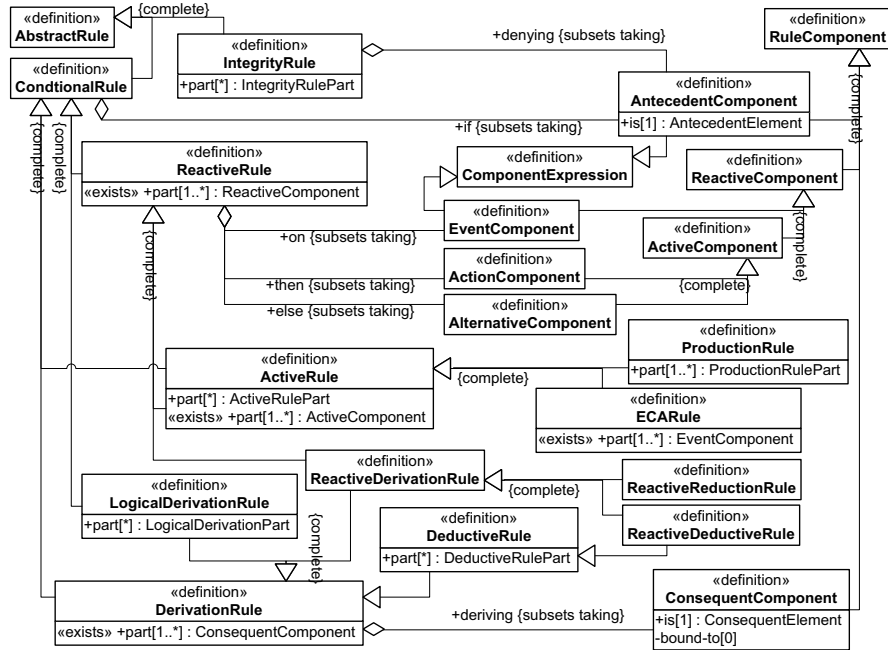


Fig. 1. Abstract Rules

Abstract *rule components* are partitioned into consequent, antecedent and reactive, i.e. event or (trans)action, components. At this foundational level, such partition is not

based on the contents of those components, but rather on a meta-level declaration (described in section 3) of what the constituent elements of a specific rule language are, and their roles in it. Such structural definition does not exclude further restriction on the contents of those components, vis-à-vis to specific rule (component) languages, i.e. extension of the r^3 ontology is possible so that semantic coherence is maintained between the contents of the rule components and the declared nature of the associated language elements (e.g. ensuring that the content of an event component is actually an event specification).

Besides the rule components, in figure 1, also *rule parameters* are permitted, cf. figure 2, accounting for modelling semantic variations (e.g. rule priority and defeasibility) that should not have an impact on the general semantics of the different kinds of abstract rules (wrt. classification of figure 1).

Focusing on the structure of rules, and rule languages, themselves, instead of focusing on the structure of each of the rule components (or on the actual semantics induced by their contents), results in a layered approach that allows the r^3 ontology to distinguish the different types of rules independently of the specific languages used in their components³; thus separating the semantics of rules from the semantics of their components. For instance, careful analysis of figures 1 and 2 conveys that:

- an active rule is an abstract rule that has at least one action component but no consequent component, provided all its antecedents are condition components;
- among active rules, ECA rules are distinguished from production rules according to the presence or absence of an event component;
- a derivation rule is a rule required to have at least one consequent component and optionally taking other antecedent (viz. necessity) components;
- a deductive rule is a derivation rule with symbolic consequents (i.e. views) based only on side-effect free⁴ components (e.g. conditions).

Notice that the r^3 ontology generalizes active rules with *alternative components* (i.e. “else”-actions). Alternative components are usually considered syntactic sugar expressible with the use of negated conditions, but given their usefulness in practice and the heterogenous nature of r^3 we believe it is important to consider them. ECA rules with an alternative component (ECAA) have a clear operational semantics⁵ and facilitate the modelling of workflows [15]. Further examples may be found in [11]. Regarding production rules, we are not aware of any formalization for alternative components and as such (and given that the main focus of our future work will be on ECA rules and their derivation variants), we have chosen to restrict the r^3 ontology, for now, to their most usual form (viz. if-then, cf. OMG’s PRR).

³ Naturally, this component-based approach has limitations if applied to the description of arbitrary logical rules (e.g. FOL formulas, in general, do not adhere to this component structure); nevertheless we believe it to be expressive enough to describe what is commonly understood as rules; not excluding general *formulas* as shown in figure 11.

⁴ Remember that without proper extension, as mentioned before, the r^3 ontology does not enforce that the content of, e.g., a condition component is actually side-effect free. It simply declares that it must be so.

⁵ Given an event occurrence if the condition has no solutions, perform an alternative action.

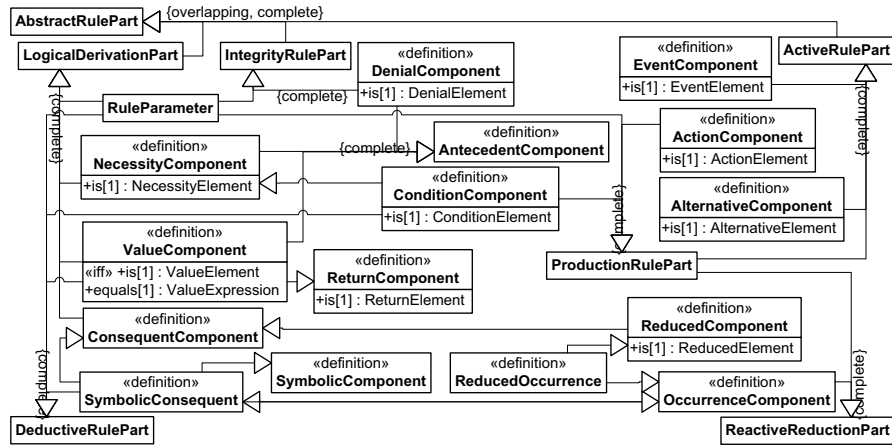


Fig. 2. Rule Parts

Among *derivation rules*, the r^3 ontology distinguishes between reactive and logical derivation rules depending, respectively, on the presence or absence of a reactive component. A *reactive derivation rule*, optionally under given conditions, allows higher level symbolic events or actions (viz. occurrences, cf. figure 3) to be derived from, or reduced to, other (atomic, composite or symbolic) events or actions. [2] identifies three kinds of reactive derivation rules, namely those that, under some conditions derive events from events (ECE), actions from actions (ACA) and events from actions (ACE). ECE rules have a purely deductive nature and, loosely speaking, they define views over events. ACA rules have a more operational nature and might be seen as reduction rules, rewriting higher level symbolic actions into lower level ones (similarly to instead-triggers in active databases). The intuitive idea underlying ACE rules is to declaratively express that when an action is executed (and some conditions are verified) some events occur as a derived consequence, and it may be realized through proper extension of ACA rules⁶. Usually, events derived from actions will include values that are only reliably known during the action evaluation (e.g. old and new in a database update action/event). Given all this, we propose, as introduced in figure 1 and further detailed in figure 3, to partition reactive derivation rules into *reactive deductive rules* (viz. ECE) and *reactive reduction rules* (viz. ACA/ACE).

Reactive derivation rules are the subject of ongoing work and further discussion about them is not in the scope of this paper. Nevertheless, it must be stressed that reactive derivation rules are mostly uncharted territory in what concerns the (Semantic) Web. To the best of our knowledge, the only published proposal relating to this matter concerns a recent evolution of the language XChange [10], which includes reactive deductive rules⁷. As such, the proposal contained in figure 3 is introduced here mainly as a matter of completeness of the presented ontology and is to be understood as a pre-

⁶ See e.g. [3] for a formalization of a declarative reactive rule language with derivation rules, and where ACE and ACA rules are not distinguishable.

⁷ We strongly distinguish a reactive deductive rule that derives events (viz. occurrences); from an ECA rule performing an action (e.g. sending a message) which may induce the occurrence of events. Resorting to reactive reduction rules, an implementation of $XChange^{EQ}$ may not need to be bound to specific (more or less ubiquitous) protocols (viz. HTTP and SOAP as suggested in [10]).

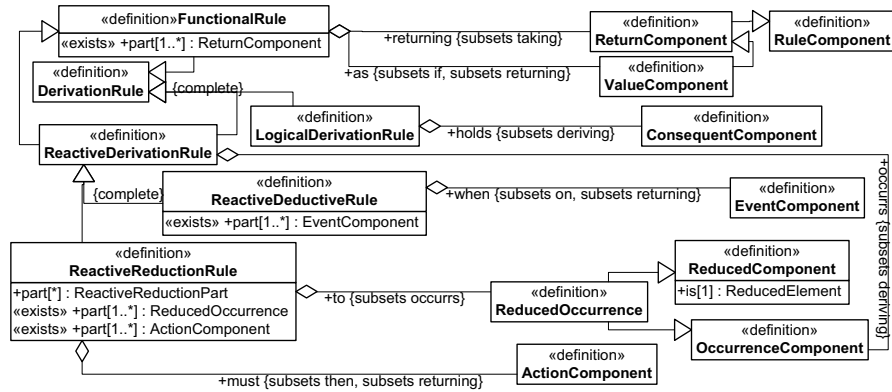


Fig. 3. Derivation Rules

liminary contribution to this open research area, requiring future validation given the foundational nature of the r^3 ontology.

3 Defining Reactive Rule Languages

Mainly, the r^3 ontology at the current foundational (and structural) level aims at providing a Semantic Web transparent abstract syntax for reactive rule-based systems. Rule component languages are assumed to follow a term structure, using a set of functors, and functor items. Such language items are described using a meta-level of the ontology. As shown in figure 4, functors are partitioned into *language constructs* and *language symbols*, and their items are distinguished between *parameters* and *construct components*. Recursively, functor items themselves are also language symbols, i.e. functors.

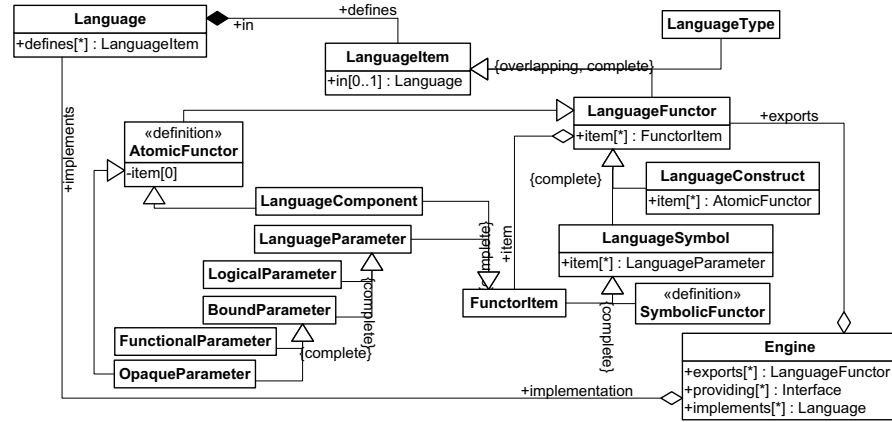


Fig. 4. Languages

The actual operational implementation of the items of a language is to be exported by some engine. More precisely, engines evaluate constructions based on language constructs, and derive symbolic terms based on language symbols.

Language components are atomic symbols (and can only be included, as items, in a language construct). Language parameters are further distinguished between logical (i.e. input/output) and bound (i.e. input) parameters. Among input parameters, opaque parameters are distinguished from purely functional parameters. Appropriate

sub-properties (viz. takes, digs, uses and binds) are introduced, in figure 5, to facilitate the declaration of functor items.

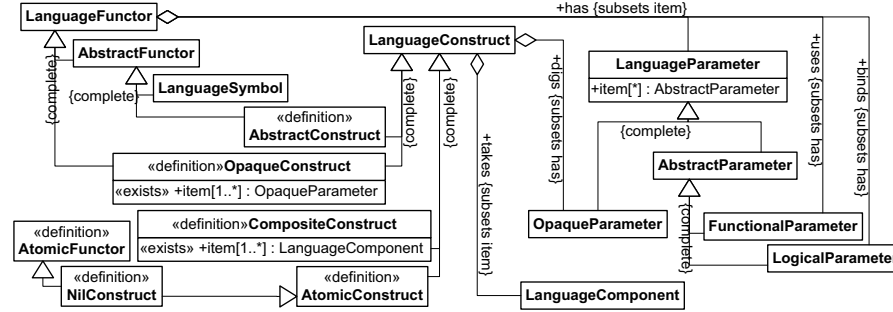


Fig. 5. Language Functors

Declaratively, a functor actually represents the set of functors formed by all its ground instances (wrt. its parameters). Operationally, the semantics of a language functor (viz. construct) can not be realized unless all its input parameters are known, i.e. bound to actual values. Opaque parameters can only be used in so called opaque constructs. They account for non-atomic parameters whose values are expressed using textual or markup (sub-)languages that hide their actual structure away. A textual template where variable references are to be substituted, a snippet of code written in some scripting language that is to be interpreted, or even the literal source of a database trigger, as further detailed below, are all examples of opaque parameters.

Abstract functors, that include only abstract –functional or logical– parameters (besides components, in case of language constructs), do not require the explicit declaration of the involved variables unless for very specific cases (e.g.: quantifying variables or scoping variables implicitly quantified, and aggregators or solution modifiers).

Language constructs, cf. figure 6, are partitioned into rule, rule package and formula constructs; distinguishing native and abstract rule constructs. Only one subset of formula constructs is identified, viz. universal or existential quantifiers, but others are not excluded: e.g. conjunction, disjunction, conditionals and negation in its different variants.

Opaque formulas are allowed, and opaque native rules are restricted to purely parametric ones, i.e. no rule elements are allowed. Native rules allow the modelling of rule constructs that use textual languages that may not follow a term structure. A native rule (e.g. a database trigger) may have some functional parameters (e.g. database name, user name and password), whose atomic values are transparently used. But it is actually expressed in opaque parameters (e.g. trigger source) which hide away considerable semantic value by using “internal” (sub-)languages.

The full semantics of an opaque construct is not accessible without full knowledge of the (sub-)languages actually used inside the (consequently, non-atomic) values of such opaque parameters. In fact, the semantics of an opaque construct is known only to an engine that, cf. figure 4, exports it or implements its associated language and opaque (sub-)languages. Usually these engines will only define the semantics of such an opaque construct in operational terms, meaning that the only form of knowing it is to submit, at runtime, an actual construction to the evaluation interface provided by the engine. Nevertheless, static analysis may sometimes be possible as long as a translation

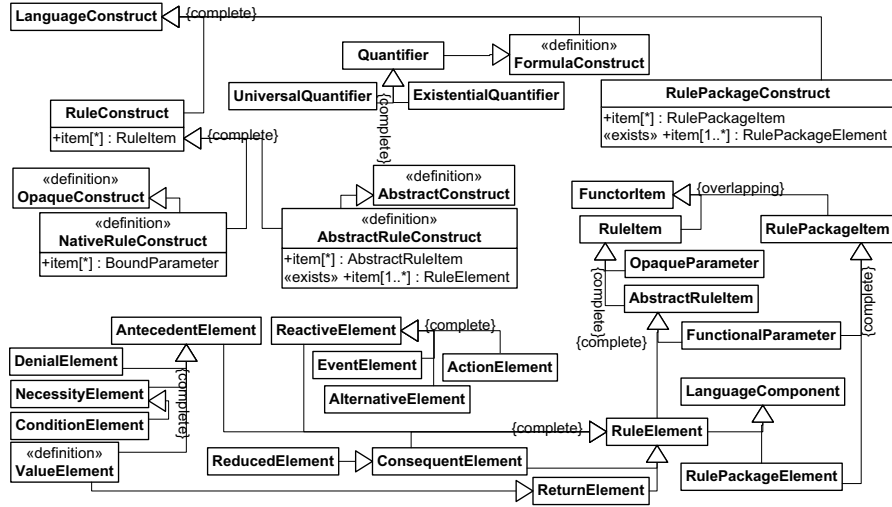


Fig. 6. Language Constructs

interface is provided by the engine for parsing an opaque construction into an abstract one.

Example 1. For an illustrative example of an r^3 language definition⁸ we resort to the current RIF Core proposal [8], more precisely to its subset dedicated to Horn rules (viz. `rif:horn`):

```

rif:ruleset a :RulePackageConstruct; :in rif:horn;
  :takes rif:rule, rif:rest.
rif:horn :defines rif:rule, rif:rest.
rif:fact a :RuleConstruct; :in rif:horn;
  :takes rif:atomic.
rif:atomic a :ConsequentElement; :in rif:horn.
rif:implies a :RuleConstruct; :in rif:horn;
  :takes rif:if, rif:then.
rif:if a :ConditionElement; :in rif:horn.
rif:then a :ConsequentElement; :in rif:horn.

```

Figure 4 includes two kinds of language items, namely functors and types. Language types, as shown in figure 7, may be literal types (e.g. lexical XML Schema types identified by an URI) or symbolic types.

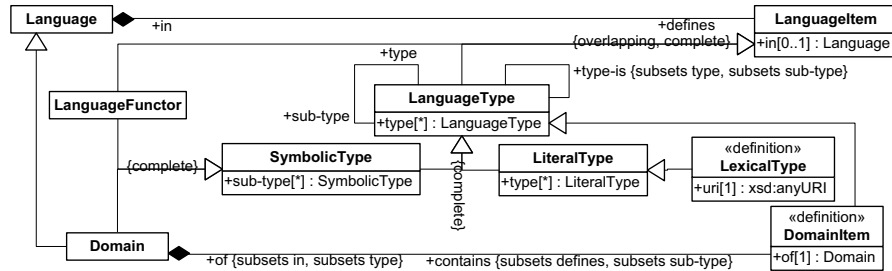


Fig. 7. Language Types

Every language functor implicitly defines a symbolic type; if not a functor a symbolic type is said to be a domain and is implicitly defined by a language.

⁸ The presented definition is a partial one, namely the universal quantifier is omitted as currently it expresses only implicit quantification at the rule level, and cardinality restrictions (on the item property) should be present, in order to close the definition of the included individuals.


```

eca:rule a :RuleConstruct; :in eca:ml;
  :takes eca:event, eca:condition, eca:action.
eca:event a :EventElement; :in eca:ml.
eca:condition a :ConditionElement; :in eca:ml.
eca:action a :ActionElement; :in eca:ml.
eca:native a :RuleConstruct; :in eca:ml;
  :uses eca:lang; :digs eca:source.
eca:opaque a :FormulaConstruct; :in eca:ml;
  :uses eca:lang; :digs eca:literal.
eca:ml :defines eca:lang, eca:source, eca:literal.

event:sequence a :Operator; :of event:algebra;
  :takes event:first, event:next.
event:algebra :contains event:first, event:next.

travel:booking-place a :SymbolicFunction; :of travel:domain;
  :binds travel:client, travel:flightnr, travel:seat.
travel:flight-info a :SymbolicFunction; :of travel:domain;
  :uses travel:flight; :binds travel:date, travel:origin, travel:destination.
travel:flightnr :of travel:domain; :type travel:flight.
travel:domain :contains
  travel:client, travel:flight, travel:seat,
  travel:date, travel:origin, travel:destination.

rental:request-quotation-for-flight a :SymbolicFunction; :in rental:application;
  :uses rental:client, rental:flight.
rental:get-client a :SymbolicFunction; :in rental:application;
  :uses rental:client;
  :binds rental:client-name, rental:favorite-class, rental:max-price.
rental:get-available-cars a :SymbolicFunction; :in rental:application;
  :uses rental:office, rental:date;
  :binds rental:car, rental:car-class, rental:price.
rental:client :in rental:application; :type travel:client.
rental:flight :in rental:application; :type-is travel:flight.
rental:application :defines
  rental:client-name, rental:favorite-class, rental:max-price,
  rental:car, rental:car-class, rental:price.

mail:send a :FormulaConstruct; :in mail:library;
  :uses mail:from, mail:to, mail:subject, mail:body.
mail:address :in mail:library;
  :sub-type mail:from, mail:to, rental:client, travel:client.

mail:library :defines mail:from, mail:to, mail:subject, mail:body.
text:join a :FormulaConstruct; :in text:library;
  :digs text:template; :uses text:separator.
text:replace a :FormulaConstruct; :in text:library;
  :digs text:template.
text:library :defines text:template, text:separator.

```

4 Defining Reactive Rule Constructions

The term languages modelled on the meta-level of the r^3 ontology, described in section 3, are used on a coding level to build coding resources that ultimately will define rule sets describing reactive rule-based systems. Coding resources, cf. figures 9 and 10, provide the foundations for a generic term structure (that is later used to define rules), and are partitioned between coding values (structured or not) and structure parts / constraints. Also distinguishable are coding variables, viz. references or declarations, and coding structures.

A coding structure is a language functor (or a hi-functor) gathering several structure parts (parameters or components), possibly further restricted with a set of constraints

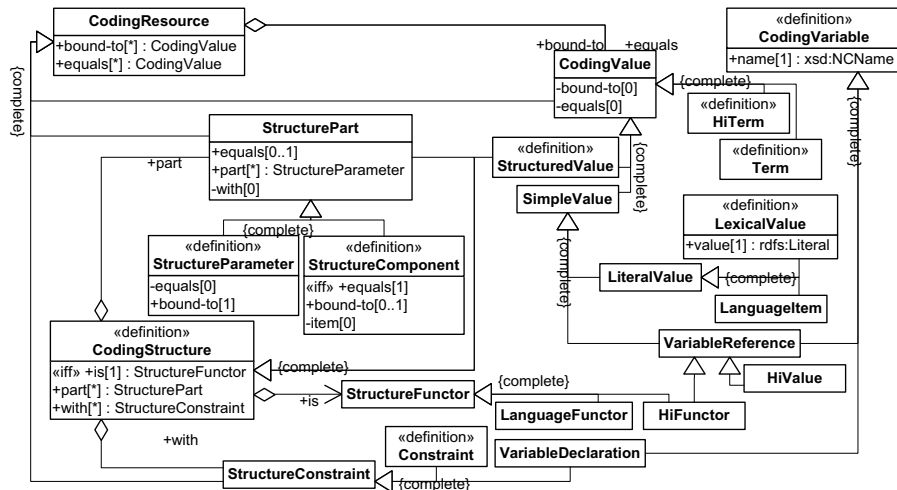


Fig. 9. Coding Resources

or variable declarations. A structure parameter is bound to a coding value; whereas a structure component, as further constrained in figure 10, equals a structured value (or a hi-value) whose returned value (if there is one) may still be bound to a coding value.

A hi-functor or a hi-value is a variable reference used in-place of a functor or of a component content, resp., which is to be understood in HiLog [12]. From the declarative point of view, r^3 will go no further than CL [14], i.e. its logical expressiveness is restricted to FOL. The semantics of a non-declarative, operational, “hi-construction” will be impossible to determine unless it is instantiated with a valid construction.

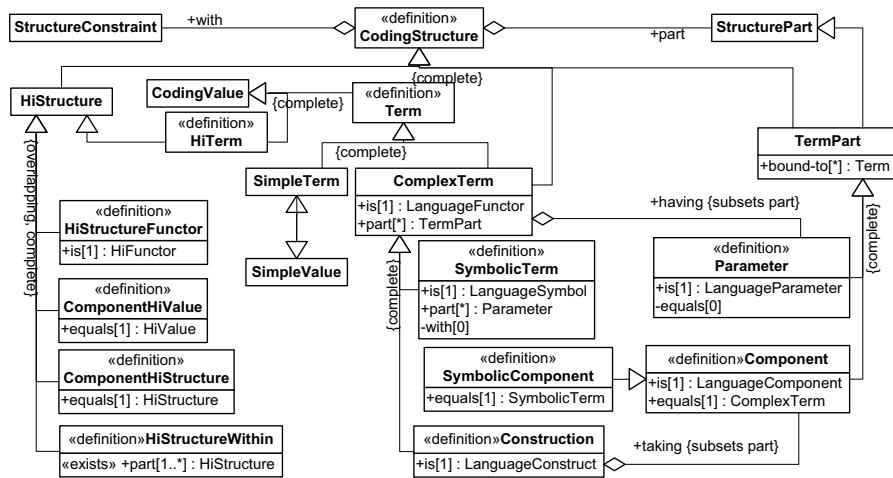


Fig. 10. Coding Terms

A coding value is either a term or a hi-term, where a term may be simple or complex. A simple term (i.e. a simple value cf. figure 10) is a literal value or a variable reference. A complex term is a coding structure, one that is a language functor and one that, as implied by the definitions in figure 10, is free of any hi-functors or hi-values. It is either a symbolic term (i.e. a language symbol without any components, constraints or

ues can only be used in the evaluation of the construction immediately containing the submitted construction, unless they are bound to some variable, thus extending the output substitutions. Submission of a rule set to an appropriate engine results in the rule set being loaded/activated.

A construction may further restrict its results with a set of structure constraints (viz. constraints or variable declarations, cf. figure 9). According to figure 12, some of the constraints are to be enforced during the evaluation; while others, namely post-conditions, are ensured to hold for every result of the construction. Among the former, pre-requisites are to be enforced, upon invocation, before the actual evaluation starts.

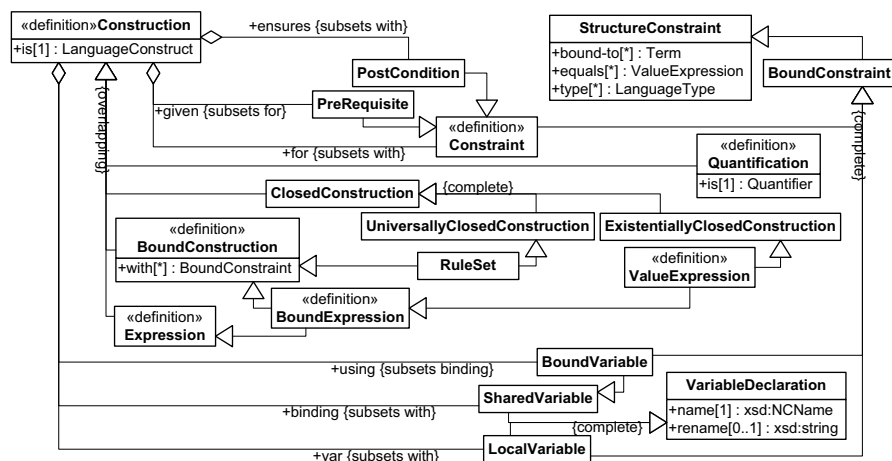


Fig. 12. Variables and Constraints

Each structure constraint defines a *constraint domain* as the intersection of its constituent domains: all possible instantiations of a particular term (it is bound to); all values returned by a value expression (it equals); or values of a particular language type. If a structure constraint has no constituent domains it trivially succeeds. Otherwise, its constraint domain must not be empty, and in case of a variable declaration it further constrains the domain of the declared variable¹¹. Constraints are a generalization of the test component proposed in [16] for ECA rules (which is seen here as a constraint that equals the test expression).

Any construction defines a scope where local variables may be declared. The communication between a sub-construction and its ancestor/sibling constructions is achieved through shared variables. Some of those shared variables may be required to be bound and a construction cannot be evaluated unless all of them are actually bound to specific values. Notice that, for efficiency, the constituents of a constraint domain, namely those based on value expressions, are expected to be incrementally evaluated thus anticipating empty constraint domain detection and variable restriction.

All variables referenced in a construction, if not explicitly declared, are implicitly declared as shared (or bound if referenced only by parameters of such nature). Furthermore, a construction implicitly inherits all the shared variables (implicit or explicit) of

¹¹ Additionally, a variable declaration may `rename` its variable, in order to support different naming conventions used by different opaque (sub-)languages.

its descendant sub-constructions. Finally, all variables shared by bound constructions are required to be bound variables: explicit declarations are restricted accordingly in figure 12.

Although scoping of variables is allowed for any construction, their actual quantification is only achieved through explicit use of quantifiers, that quantify and scope their local variables, or through implicit quantification as induced by closed constructions, that quantify all the variables referenced within and scope them if needed (with the possible exception of bound variables they explicitly declare).

Example 4. Concluding this section, below we illustrate the usage of the r^3 ontology to define a simplified ECA rule that, as a result of some client booking a flight and requesting a car rental quotation, sends him the quotation for the cars available at the date of the flight. For this we resort to the r^3 languages included in the examples of section 3.

```

_:quotation-request-rule :is eca:rule;
:on  [:is eca:event;      :equals _:request-for-quotation];
:if  [:is eca:condition; :equals _:available-quotation];
:then [:is eca:action;    :equals _:send-quotation].

_:request-quotation :is event:sequence;
:taking [:is event:first; :equals [
  :is travel:booking-place;
  :having [:is travel:client; :bound-to [:name "Mail"]];
  :having [:is travel:flightnr; :bound-to [:name "Flight"]]]];
:taking [:is event:next; :equals [
  :is rental:request-quotation-for-flight;
  :having [:is rental:client; :bound-to [:name "Mail"]];
  :having [:is rental:flight; :bound-to [:name "Flight"]]]].
_:available-quotation :for _:price-ok; :is rif:and;
:taking [:is rif:some; :equals _:flight-info];
:taking [:is rif:other; :equals [:is rif:and;
  :taking [:is rif:some; :equals _:get-client];
  :taking [:is rif:other; :equals _:get-available-cars]]].
_:get-client :is rental:get-client;
:having [:is rental:client; :bound-to [:name "Mail"]];
:having [:is rental:client-name; :bound-to [:name "Client"]];
:having [:is rental:favorite-class; :bound-to [:name "Class"]];
:having [:is rental:max-price; :bound-to [:name "Max-Price"]].
_:get-available-cars :is rental:get-available-cars;
:having [:is rental:office; :bound-to [:name "To"]];
:having [:is rental:date; :bound-to [:name "Date"]];
:having [:is rental:car; :bound-to [:name "Car"]];
:having [:is rental:car-class; :bound-to [:name "Class"]];
:having [:is rental:price; :bound-to [:name "Price"]].
_:flight-info :is travel:flight-info;
:having [:is travel:flight; :bound-to [:name "Flight"]];
:having [:is travel:date; :bound-to [:name "Date"]];
:having [:is travel:origin; :bound-to [:name "From"]];
:having [:is travel:destination; :bound-to [:name "To"]].

_:price-ok :equals _:check-price; :bound-to [:value "true"].
_:check-price :is eca:opaque;
:using [:name "Price"], [:name "Max-Price"]; :rename "MaxPrice";
:having [:is eca:lang; :bound-to [:value "http://www.w3.org/XPath"]];
:having [:is eca:literal; :bound-to [:value "$Price <= $MaxPrice"]].
_:send-quotation :is mail:send;
:var [:name "Text"; :equals _:quotation-message];
:having [:is mail:from; :bound-to [:name "Rental-Mail"]];
:having [:is mail:to; :bound-to [:name "Mail"]];
:having [:is mail:subject; :bound-to [:value "Car Rental Quotation"]];
:having [:is mail:body; :bound-to [:name "Text"]].
_:quotation-message :is text:replace;

```

```

:var [:name "Priced-Cars";
:equals [:is text:join;
:using [:name "Car"], [:name "Price"];
:having [:is text:template; :bound-to [:value "|Car|=|Price|"];
:having [:is text:separator; :bound-to [:value ", "]]];];
:using [:name "Client"], [:name "Flight"], [:name "Date"], [:name "To"];
:having [:is text:template; :bound-to [:name "QuotationTemplate"]].

```

5 Conclusion and Future Work

The r^3 ontology provides a foundation to describe both reactive rules and reactive rule languages. The latter is accomplished on a meta-level where not only rule languages themselves but also other supporting languages used in rule components can be described. The former is provided following a term-based compositional approach that allows not only the use of different languages for different rule components (e.g. event, condition, action), but also the composition of different languages in a single component possibly using algebraic languages, thus accounting for language heterogeneity.

The r^3 ontology recognizes that full expressivity of reactive behavior can not be achieved without the help of derivation rules. As such it includes a proposal for reactive derivation rules (complementing the more traditional active rules, viz. ECA and production rules) and also logical derivation rules. Additionally, it recognizes the importance of reliability for reactive systems and contemplates global integrity rules.

Furthermore, the r^3 ontology contributes for the clarification of concepts through formal definition of an RDF vocabulary for describing rule-based reactive behaviour; notably establishing a clear distinction between reactive, active and deductive rules, consistently with the terminology traditionally used in the Active Databases field. As obvious as it may seem, it should be stressed that the formal definition of such a (Semantic Web transparent) vocabulary is not to be confused with the formalization of a semantics for reactive rule languages; the former is the subject of this paper and provides the basis (as an abstract syntax) for the definition of the latter (which is out of scope here).

Last, it is worth mentioning that the work here presented builds upon previous concrete proposals of the r^3 ontology that provided the basis for the current implementation of the r^3 prototype [1] available online [23]. In this prototype a previous version of the r^3 ontology has already been used to model several component languages. Namely, the languages Xcerpt for queries, XChange for events and actions of updates of XML data, Prova, XQuery/XPath and HTTP. This work actually lead to the inclusion of these Languages (as fully functional Engines) in the current version of the r^3 prototype. Details about the previous definition and implementation of these Languages may be found in [1]. This experimental work on defining Languages has brought relevant contributions for the r^3 ontology and must continue to be pursued and extended to other languages like, e.g.: XSLT, the algebraic language for actions described in [6], the *XChange^{EQ}* [10] or ruleCore [20] event languages, or even the SPARQL algebra.

At the current foundational level, the r^3 ontology mainly defines a Semantic Web transparent abstract syntax for reactive rules. As usual for most abstract syntax, it allows invalid constructions without a defined semantics (e.g. infinite terms). This abstract syntax must be validated against and complemented with a formal semantics definition.

Achieving such a formal definition, together with the re-implementation of the r^3 prototype according to it, constitutes our major goal for the immediate future.

The future work on the r^3 semantics and prototype is to be focused mainly on ECA and reactive derivation rules, although not discarding consistent integration with the other types of rules (particularly the evolution of RIF [8] is to be followed as closely as possible). A very important matter needs to be carefully considered: a solution grouping mechanism is mandatory for actions (as careful analysis of example 4 shows). Whether such grouping is achieved through the use of grouped aggregations, or resorting to solution modifiers like project and distinct, or left out for action languages is still an open matter under discussion at the time of this writing. As such we have chosen not to include any proposal on this issue here. On the other hand, types and hi-terms are a matter to be taken conservatively and may be postponed until a stable version of the prototype is available.

Considering the extension of the r^3 ontology to higher abstraction levels (e.g. rule languages, domain/application languages, event languages, event algebras, process algebras) is also to be pursued in close cooperation with the MARS project [22]. Furthermore, the r^3 ontology implicitly extends and generalizes the previously proposed general ECA framework [2, 16, 17] (e.g. derivation rules, solution constraints and rule constructions) and calls for a revision of the ECA-ML markup [16] and of the general framework. Any revision of the ECA-ML markup should: consider an homogenous markup for logical variables, and be based upon the formal specification of the (revised) framework. Given the r^3 ontology (as an adequate abstract syntax for the framework), such a (fully) formal specification should now be possible to achieve without resorting to general markup guidelines and principles.

Although not directly related, given its ontology based approach, to current textual or markup based proposals for concrete Web ECA languages (e.g. [9, 19, 10]), r^3 undoubtedly aims at modelling most (if not all) of them. As such, given a formal semantics of r^3 , actual demonstration of its adequacy to this purpose is also to be pursued.

References

1. José Júlio Alferes, Ricardo Amador, Erik Behrends, François Bry, Michael Eckert, Tiago Franco, Oliver Fritzen, Hendrik Grallert, Tobias Knabke, Ludwig Krippahl, Wolfgang May, Paula-Lavinia Pătrânjan, Franz Schenk, and Daniel Schubert. Completion of the prototype scenario. REVERSE deliverable I5-D7, CENTRIA, Universidade Nova de Lisboa, <http://reverse.net/deliverables.html>, March 2007.
2. José Júlio Alferes, Ricardo Amador, Erik Behrends, Oliver Fritzen, Tobias Knabke, Wolfgang May, Franz Schenk, and Daniel Schubert. Reactive rule ontology: RDF/OWL level. REVERSE deliverable I5-D6, CENTRIA, Universidade Nova de Lisboa, <http://reverse.net/deliverables.html>, April 2007.
3. José Júlio Alferes, Federico Banti, and Antonio Brogi. An Event-Condition-Action Logic Programming Language. In *JELIA'06*, volume 4160 of *LNAI*. Springer, 2006.
4. José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira. An evolvable rule-based e-mail agent. In *EPIA'03*, volume 2902 of *LNCS*. Springer, 2003.
5. 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). Submission, W3C, <http://www.w3.org/Submission/2005/SUBM-WRL-20050909/>, September 2005.
6. Erik Behrends, Oliver Fritzen, Wolfgang May, and Franz Schenk. Combining ECA Rules with Process Algebras for the Semantic Web. In *RuleML'06*. IEEE, 2006.
 7. Harold Boley, Benjamin Grosf, Michael Sintek, Said Tabet, and Gerd Wagner. *RuleML Design*. RuleML Initiative, <http://www.ruleml.org/>, August 2006.
 8. Harold Boley and Michael Kifer. RIF Core Design. Working Draft, W3C, <http://www.w3.org/TR/2007/WD-rif-core-20070330>, March 2007.
 9. Angela Bonifati, Daniele Braga, Alessandro Campi, and Stefano Ceri. Active XQuery. In *ICDE'02*. IEEE, 2002.
 10. François Bry and Michael Eckert. Rule-Based Composite Event Queries: The Language XChangeEQ and its Semantics. In *RR'07*, volume 4524 of *LNCS*. Springer, 2007.
 11. François Bry, Michael Eckert, Paula-Lavinia Pătrânjan, and Inna Romanenko. Realizing Business Processes with ECA Rules: Benefits, Challenges, Limits. In *PPSWR'06*, volume 4187 of *LNCS*. Springer, 2006.
 12. Weidong Chen, Michael Kifer, and David Scott Warren. HILOG: A Foundation for Higher-Order Logic Programming. *Journal of Logic Programming*, 15(3):187–230, 1993.
 13. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosf, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. Submission, W3C, <http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/>, May 2004.
 14. 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.
 15. Gerhard Knolmayer, Rainer Endl, and Marcel Pfahrer. Modeling processes and workflows by business rules. In *Business Process Management, Models, Techniques, and Empirical Studies*, pages 16–29. Springer, 2000.
 16. Wolfgang May, José Júlio Alferes, and Ricardo Amador. Active rules in the Semantic Web: Dealing with language heterogeneity. In *RuleML'05*, volume 3791 of *LNCS*. Springer, 2005.
 17. Wolfgang May, José Júlio Alferes, and Ricardo Amador. An ontology- and resources-based approach to evolution and reactivity in the Semantic Web. In *ODBASE'05*, volume 3761 of *LNCS*. Springer, 2005.
 18. Object Management Group. *Semantics of Business Vocabulary and Business Rules (SBVR)*. OMG, <http://www.omg.org/cgi-bin/doc?dtdc/2006-08-05>, August 2006.
 19. George Papamarkos, Alexandra Poulouvassilis, and Peter T. Wood. Event-condition-action rules on RDF metadata in P2P environments. *Computer Networks: The International Journal of Computer and Telecommunications Networking*, 50(10):1513–1532, 2006.
 20. Marco Seiriö and Mikael Berndtsson. Design and implementation of an eca rule markup language. In *RuleML'05*, volume 3791 of *LNCS*. Springer, 2005.
 21. Silvie Spreeuwenberg and Rick Gerrits. Business Rules in the Semantic Web, Are There Any or Are They Different? In *Summer School Reasoning Web 2006*, volume 4126 of *LNCS*. Springer, 2006.
 22. MARS: Modular Active Rules for the Semantic Web. <http://reverse.net/I5/MARS/>. DBIS, Institute for Informatics, Georg-August-Universität Göttingen.
 23. Resourceful Reactive Rules (r^3). <http://reverse.net/I5/r3/>. CENTRIA, Universidade Nova de Lisboa.

A Meta-level Extension Examples

In figure 1, multiple occurrences of the same rule component may seem to be allowed in a rule. This is not the case. Each component of any construction (cf. a meta-level described language construct that takes a precise set of language components) must be included once and only once. Generally, most practical language definitions include some forms of syntactic sugar. Among the most common forms are optional/multiple parameters/arguments. These forms of syntactic sugar very often hide major semantic decisions. In an eclectic heterogenous context like the one proposed, the “sweetness” of these syntactic forms may be misleading and hinder the semantic understanding of a set of constructions. For instance, if an ECA rule is written taking several condition or action components, the actual meaning of such constructions remains unknown or opaque unless a particular composing 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 probably be to interpret them as a disjunction). Appropriate extension of the meta-level of the r^3 ontology, as presented in figure 13, may provide enough expressivity for the most “annoying” situations without disregarding semantic transparency.

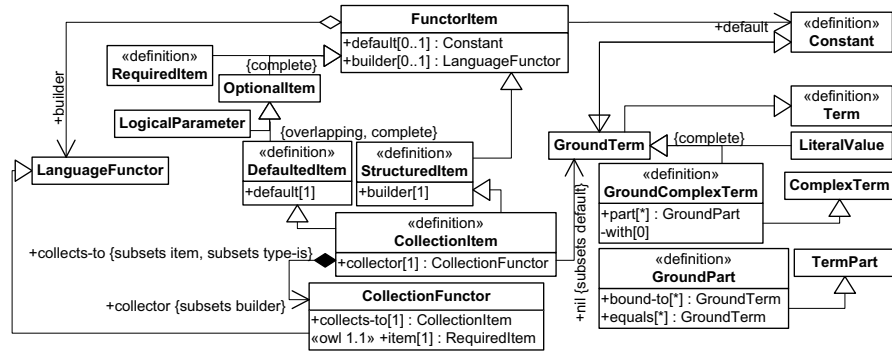


Fig. 13. Language (not so Syntactical) Details

Optional items (parameters or components) are modelled as usual with the introduction of `default` constants that are to be used if the item is omitted. Logical parameters are optional even without a default, in which case they are to be understood as free (i.e. unbound) if omitted.

Multiple items are considered a specialization of a more general concept: structured items. A *structured item* is always built using a specific `builder` and most often it induces syntactic simplifications when defining markup languages (i.e. the structured item or its `builder` are omitted, as long as ambiguity issues are not introduced). A *collection item* (named after `rdf:parseType="Collection"`) has a recursive nature given by its `collector` (usually a binary associative one). This collector “consumes” a single instance of a (required) item and further collects additional instances in the collection item itself (substituted by a required `default -or nil-` constant, which stands for the empty collection, if there are no further instances to collect).

Example 5. To illustrate some of these issues, consider the `eca:ml` (partial) definition in example 3 with the following extension of `eca:condition` as an optional component structured as a (possibly empty) collection of queries followed by an optional test.

```

eca:condition
  :builder eca:collect-and-test; :default eca:true.
eca:collect-and-test a :FormulaConstruct; :in eca:ml;
  :takes eca:collect, eca:test.
eca:collect a :ConditionElement; :in eca:ml;
  :collector eca:query-collect; :nil eca:true.
eca:test a :ConditionElement; :in eca:ml;
  :default eca:true.
eca:query-collect a :FormulaConstruct; :in eca:ml;
  :takes eca:query, eca:collect.
eca:query a :ConditionElement; :in eca:ml;
  :default eca:true.
eca:true a :NilConstruct; :in eca:ml.

```

Notice in example 5 that proper extension of the r^3 ontology would be required for a full characterization of `eca:collect-and-test` and `eca:query-collect` as conjunction constructs and of `eca:true` as a true constant. Such a characterization could be obtained, for instance, as depicted in figure 14.

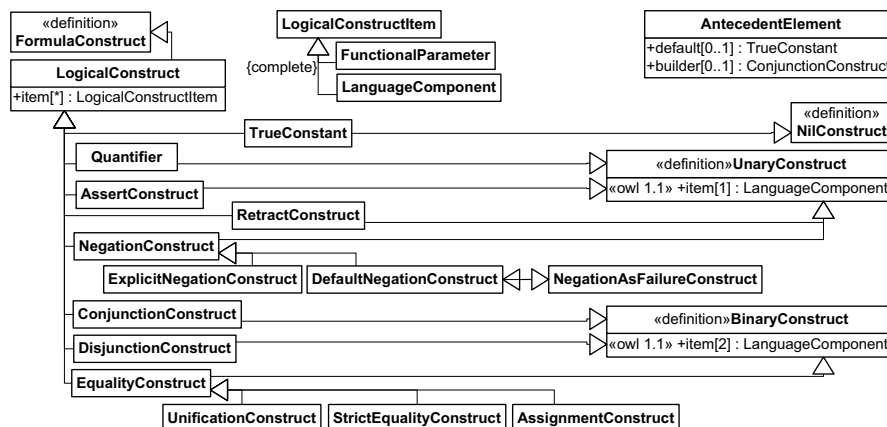


Fig. 14. Extending Formula Constructs

Some of the `eca:ml` language items, included in examples 3 and 5, 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 [16] 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, the ECA-ML markup only includes the "leaf" components `eca:query` and `eca:test`. 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 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 r^3 languages, it is worth stressing that strict conformance to OWL-DL must be observed at all times. The language meta-level of the r^3 ontology may lead to, sometimes tempting, misuses that may introduce undecidability issues. For instance, an ontology that imports the 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 `LanguageItems`) 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 its `LanguageItems`). Such an ontology (or its namespace, e.g. `rif:`) should never be mistaken for the `Language` itself (e.g. `rif:core`). 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 (`LexicalType` suggests one possible way around this, viz. extend the `LanguageType` class with appropriate data - or annotation - properties that identify the associated OWL class).