REACTIVE: A Rule-based Framework to Process Reactivity

Elsa Tovar[1,2] and María-Esther Vidal[1]

[1] Universidad Simón Bolívar
Caracas, Venezuela
{elsa,mvidal}@ldc.usb.ve
[2] Universidad de Carabobo, Venezuela
eltovar@uc.edu.ve

Abstract. Ontologies have been successfully used to model data and knowledge that conceptualize a particular domain. However, in the real-world, resource properties can be affected by events, and ontology formalisms need to be able to represent the conditions that fire these events as well as their consequences. In this paper we define the framework REACTIVE that provides the basis to specify active ontologies and to process reactivity efficiently. In REACTIVE, active ontologies are specified in ActRDFS and queries are expressed in ActSPARQL; these languages respectively extent RDFS and SPARQL to specify data reactive behavior. To efficiently process reactivity, REACTIVE implements an optimization technique named Intersection of Magic Rewritings (IMR), which is able to identify the minimal set of changes of the resource properties that are fired by a set of events in a query. We have conducted an experimental study for processing reactivity on a variety of datasets, and we have observed that IMR is able to reduce by at least a half the reactive processing time w.r.t. non-optimized queries.

1 Introduction

In the context of the Semantic Web, ontologies are extensively used to conceptualize a particular domain in terms of the static properties that characterize relevant resources. Existing formalisms only allow the representation of static properties, i.e., they express information about data and metadata that do not react when events occur; for example, classes and properties of RDF/RDFS and OWL. However, the values of resource properties may change when certain events are fired. For example, the intensity of an earthquake depends on its focal depth; thus, when the focal depth is less than 70 km, the earthquake is highly destructive and its intensity is very high. Furthermore, when the earthquake intensity increases, a natural disaster occurs and some actions need to be taken in place. From an ontological point of view, earthquake and natural disaster can be modeled as classes, intensity and focus depth are properties, and events that fire the earthquake and the natural disaster are concepts related by a subsumption relationship. However, none of the existing ontological languages are tailored to express properties or relationships between events at the same level as properties and classes; in consequence, reactivity is managed outside of ontologies and the effects of events may not always be used to infer new knowledge. In this paper we present an infrastructure

named REACTIVE (pRocEssing of ACTIVE ontologies) to efficiently processing reactivity. Our approach is comprised of: an ontological formalism name ACTION [24] that is able to express classes, properties and events as concepts to implement active ontologies; a formalization of active ontologies as a deductive database to construct a rule-based framework which supports reactivity using static and active knowledge during the reasoning tasks; a query processing technique to efficiently evaluate the reactive behavior of ontological data [24]; a dialect of RDFS to represent active ontologies; and an extension of SPARQL that provides operators to specify a set of events which can be fired in sequence or in parallel during the reactive processing task. This paper comprises five additional sections. The next section summarizes the related work. In section 3, our approach is described. Section 4 presents extensions of the SPARQL and RDFS languages. The experimental study is reported in section 5. Finally, in section 6, conclusions are pointed out.

2 Related Work

Existing active ontological approaches follow a hybrid representation model, where reactivity is represented by active rules which have the syntactical structure and the semantics of the event-condition-action rules (ECA rules paradigm) [20]. Different approaches that combine XML data with ECA rules to process reactivity have been proposed [1–4, 6, 8–11, 19, 22], and ontologies with ECA rules [13, 21, 25] . However, in these approaches, events are not represented as part of the universe of discourse, and active knowledge is not used in conjunction with static data to infer new knowledge when reactivity is processed. Active knowledge, classes and roles are also treated independently.

In [6], ECA rules are combined with algebras of processes to manage reactive behavior. This approach is a model and architecture for ECA rules that uses heterogeneous event, query, and action languages. Different parts of ECA rules are handled by specific services that implement the respective languages. Rules are specified in a markup language RuleML that expresses simple and complex events, as well as, reactive processing [7]. Active knowledge coded in rules is associated with domain ontologies, but events are not represented as part of the domain. Similarly, in [1] reactive behavior is modeled by using ECA rules, and a foundational ontology is proposed to represent the language needed to specify these rules; in addition, this formalism is used to represent reactive policies in [2]. Although these ECA-based approaches are able to deal with reactivity, active knowledge is not used in conjunction with static data during the query processing and reasoning tasks.

[13] propose an approach to manage data changes over time. This approach consists of a logical framework for representing activities, states, and time within an ontology in First Order Logic and reasoning regarding occurrence of actions by means of intelligent agents. Although this approach categorizes static and active knowledge at the same level, it does not augment the expressive power of the formalism by using the reactive behavior represented in the active knowledge. Recently, an ontology-based information integration approach [25] was presented for distributed sources. Ontologies are used to express information about frequent changes of metadata and overlapping pieces of in-

formation in distributed and heterogeneous sources. In [17, 18] a dynamic logic-based formalism [15] to represent a multi-version ontology reasoning system is described. In this approach different versions of an ontology are represented as spaces of the logic semantic model, and relationships between versions are modeled by sequence-based operators. In a similar way, our ontology framework ACTION can be formalized by using a dynamic logic; however, in order to efficiently manage reactivity, the REACTIVE system does not store the different spaces of an ontology. In addition, REACTIVE provides tailored methods to minimize the changes required to move from one space to the next space when events are fired.

To enhance the expressiveness of ontology languages, hybrid rule-based systems like SWRL, have been defined [16, 21]. In [16], Horn clause rules were added to OWL-DL. However, SWRL extends OWL with the most basic kind of Horn rules where predicates are limited to being OWL classes and properties. Relations between values of properties and events cannot be represented; thus, it is not possible to express how ontological data react to the events that affect them. Furthermore, in hybrid systems, rules are used to augment the knowledge represented in ontologies; but, neither ontologies nor rules are able to represent reactivity or use the changes induced by the reactivity to infer new active knowledge at the ontology level.

In conclusion, although ECA rules are the most widely used approach to process reactive behavior, in general relationships between active rules and ontologies are modeled in an operational way, and active knowledge, classes and roles are treated independently. To overcome this limitation, we propose an alternative processing scenario and an active ontological formalism to model the information required to represent reactivity.

3 Our Approach

3.1 Background

In a previous work, we presented the active ontology formalism called ACTION [24], aiming at enriching the Semantic Web with active specification to express reactive behavior. ACTION formalism represents the active ontology canonical form to which can be traduced any ontological language.

An ACTION ontology is defined as a 7-tuple
$Oa =< C, E, Ps, Pa, F, fr, I >$, where:

- C: a set of classes or basic data types.
- E: a set of events.
- Ps: a set of static properties; each property corresponds to a function from $C \cup E$ to $C \cup E$.
- Pa: a set of active properties; each property corresponds to a function from C to C.
- F: a set of predicates representing instances of the classes, properties and events.
- fr a function, s.t., $fr : F \times Pa \times E \rightarrow F$; fr defines the reactive behavior in Oa.
- I: a set of axioms that describe the properties of the ontological language built-in properties.

The set of static properties Ps is comprised of properties that may induce a hierarchy of events, or a hierarchy of classes, and there will be a deductive rule indicating that the built-in predicate *isSubEventOf* is transitive in the set of axioms I of the ACTION ontology. Similarly, there are some other rules that establish the properties of the predicates *isSubClassOf* and *isSubPropertyOf*. Thus, event, classes and properties are considered as first-class citizens and are undistinguished treated by the reasoning engine. Similar to the approach presented in [23], ACTION ontologies are represented as deductive databases ADOBs. An extensional deductive database for an ACTION ontology Oa is comprised of meta-level predicates that model the knowledge explicitly represented by sets C, E, Ps, Pa, and F, while the intensional component of ADOB is composed of the rules that define the semantics of the knowledge represented in the extensional predicates and modeled by the axioms in I. We have defined a set of meta-level predicates to represent the reactive behavior of the data. Some of the meta-level predicates are as follows: *isEvent(E)* where E is a name event, the built-in predicate *isSubEventOf(E1,E2)* defines that the event name $E1$ is a sub-event of the event name $E2$, and the intensional meta-level predicate *areSubEvents(E2,E1)* is specified by a deductive rule that states that the predicate *isSubEventOf(E1,E2)* is transitive. Furthermore, the meta-level predicate *activeProperty(AP,T,D,R)* defines an active property AP in terms of its type T (not the same as rdf:type), domain D and range R; and the predicate *reactiveBehavior(AP,E1,BE,V)*, specifies the reactive behavior of an active property AP that takes the value V when an event $E1$ occurs and the Boolean expression BE holds. BE is a Boolean expression over the properties in Ps and Pa. As usual, an active ontology query is a rule $q : Q(X) \rightarrow \exists Y B(X, Y)$ where B is a conjunction of predicates. No free variables exist in the ontology, and our approach is based on the Closed-World assumption.

3.2 Motivating Example

Consider the domain of natural disasters. Natural phenomena are caused by nature, e.g., earthquakes, hurricanes and volcanic eruptions, etc. Each of them has its own features, e.g., an earthquake is generated at certain depth of its foci, and it produces a seismic wave whose amplitude expresses the earthquake power; and the wind speed is the main property of a hurricane. Phenomena have different names according to the environment where they occur, e.g., hurricanes are named typhoons in the Pacific Ocean. All these concepts can be modeled by RDF/RDFS, e.g., *NaturalPhenomenon*, *Earthquake*, *Hurricane* and *Typhoon* classes can be defined. Properties such as the intensity of an earthquake and the category of a hurricane can also be expressed. Furthermore, it could be asserted that *Earthquake*, *Hurricane* and *Typhoon* are subclasses of *NaturalPhenomenon*. However, it is not possible to represent that if the depth of an earthquake foci is shallow focus, i.e., if it is less than 70 km depth, then the intensity is very high, the earthquake is very destructive, and it needs to be considered as a natural disaster of great magnitude; thus, particular actions need to be taken, for example, evacuation scale has to be full and recovery fund needs to be equal to 1 Billion.

ACTION static and active properties can be used to differentiate between the characteristics that describe the above described reactive behavior, e.g., foci depth, earthquake intensity, evacuation scale and recovery fund. Events specify when and how the active

properties are affected, e.g., if foci depth is less than 70 km depth, the earthquake intensity is very high and the events of a very destructive earthquake and natural disaster of great magnitude are fired. Finally, hierarchies of events can conceptualize subsumption relationships between events, e.g., the event very destructive earthquake is a sub-event (e1) of the event natural disaster of great magnitude (e3) - e1 and e3 are hexagons in Figure 1.

In order to support the discovery tasks required to identify the effects of a given set of fired events, we have incorporated evaluation techniques into the REACTIVE query and reasoning engine. These techniques are able to identify the minimal set of changes that need to be performed to the active properties, when some events are triggered and they may activate the same changes multiple times. To illustrate this problem consider the hierarchy of events presented in Figure 1, when events e1 and e2 are simultaneously fired, and the effects of the common super-events need to be evaluated several times (events inside the highlighted circle). Properties p1 and p2 (circles) are affected by these events, and in a similar way, all the super-properties are considered several times, (properties inside the highlighted oval). Our proposed optimization technique Intersection of Magic Rewritings (IMR), identifies the events and properties that need to be considered multiple times, and constructs the minimal set of rules that will produce the same result, but that will avoid duplicates.

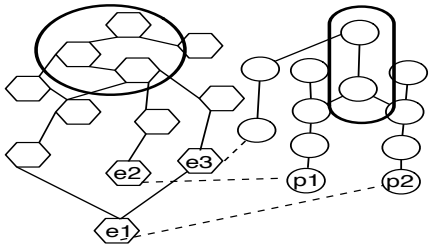


Fig. 1. An **ACTION** ontology

3.3 The REACTIVE Architecture

REACTIVE is a rule-based framework that supports reactive processing of active ontologies. The REACTIVE architecture is showed in Figure 2. Active ontologies are specified in ActRDFS an extension of RDFS, and they are translated into an Active deductive database (ADOB). ActSPARQL queries are decomposed by the Query Decomposer into a traditional SPARQL query, and an Active Query. The SPARQL query will be evaluated by an SPARQL query engine on the ontology produced as the result of processing the reactive behavior fired by the active query. Next, an Active Query Preprocessor receives the Active Query and produces an Adorned Magic Set Rules processing query in three steps. First, it aggregates sub-goals of the Active Query according to the aggregation criteria that will be presented in the next section. After, an Event

Planner generates all super-events of the aggregated sub-goals. The process above implies generating not only the super-events (with no repetitions), but it also checks if the Boolean conditions of the events, do not have active properties. In this case, a reordering of events must be done, due to the fact that if two events affect the same active properties only the last event must be processed. However, if there are active properties in the Boolean conditions, the IMR Query Writer must generate the Adorned Magic Set Rules according to a concurrent study producing a set of rules to process reactivity of a set of events that guarantees termination. Finally, a Rule-based engine evaluates the Adorned Magic Set Rules in conjunction with the rule-based representation of the input ontology, and produces a new static ontology. This static ontology and the SPARQL query are sent to a SPARQL engine, to produce the query answer.

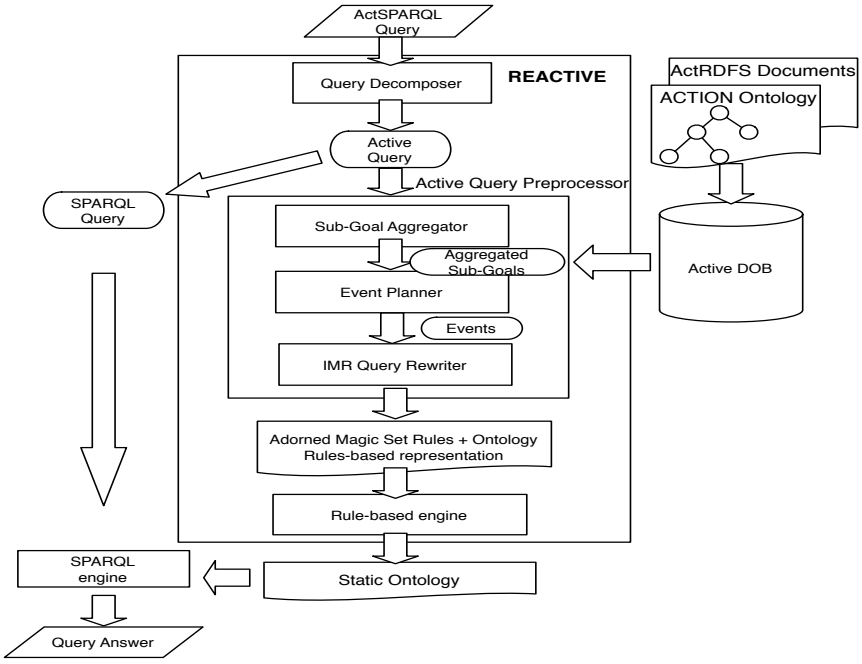


Fig. 2. The REACTIVE Architecture

3.4 Processing Reactivity

We present a general algorithm to process reactive behavior triggered by an event E occurring on concept C [24]. This algorithm receives the minimal model MM of a canonical deductive database ADOB of an ontology Oa, and it determines the individuals in the set C that are affected by the event E and all its super-events. The algorithm is based on the following assumptions:

- if an active property AP is affected by an event E, then all the super-properties of AP are also affected.
- if an event E affects an active property AP when the property P takes the value V, then E affects AP when any sub-property of P has the value V.

To compute the minimal model MM the following predicates are evaluated: *areSubEvents(F, E)*, *areSubProperties(AP, PA)*, *areStatements(Ii, PA, V1)*, *areReactiveBehavior(PA, F, P, BC, V2)*. It must be pointed out that this algorithm alters extensional knowledge but it does not alter metadata or intensional predicates. Thus, there is not possibility to generate inconsistencies between ontological definitions and data produced during the reactive processing. The time complexity of the active algorithm is bound by the time complexity of the transitive closure [12]. Thus, the complexity of the active algorithm is $O(n^3 \times M)$, where n is the number of instances of the predicates $isSubEventOf$, and M is the number of active property predicates to be changed. The number of derived facts polynomially depends on the number and relationships of the events and the same evaluations may be fired by different events (details in [24]). Thus, this enrichment of expressiviness can negatively impact the complexity of the reasoning task implemented by the active algorithm; hence efficient query evaluation techniques are necessary. A naive solution is to follow a bottom-up evaluation. This strategy computes the minimal model and each fact is inferred once but a large number of irrelevant facts may be inferred. On the other hand, top-down evaluation only computes the relevant facts but the same fact can be inferred several times (repeated inferences). Thus, unnecessary inferences can be performed. To overcome these limitations, Magic Set techniques can be applied hence they reduce repeated and unnecessary inferences; magic predicates are introduced to represent bound arguments in the fired events; and supplementary predicates represent sideways-information-passing in the deductive database rules [5].

Our current version of the query engine algorithm is able to process a set of events that affect a particular class. Additionally, we consider that Boolean conditions associated with events can be comprised of static and active properties, i.e., there exists no restriction about the kind of properties that can be used in the Boolean conditions. This characteristic impacts on the complexity of the reactive processing task; this is because an active property $ap1$ affected by an event $e1$ could depend on the changes of another active property $ap2$ which is affected by another event $e2$, when $e1$ and $e2$ are processed simultaneously. Thus, criteria to detect when reactive processing will terminate is required. Finally, we consider that a collection of events on a particular class can be serial, i.e., events occur one after another; and parallel, i.e., events occur *at once*. In this work the IMR method presented in [24] has been extended to process reactivity in these two scenarios.

To deal with serial and parallel sets of events, we have introduced the notion of *processing schedule* that indicates the order in which events that appear in an input active query, must be processed. The rules that define the processing schedule for an active query are as follows:

- If events on class C occur in parallel, and:

- there are no active properties in any Boolean condition associated with the events, then the order in which events appear in the processing schedule can be the same as the order in which they appear in the input active query;
- there are active properties in at least one Boolean condition associated with the events, then the events of the processing schedule have to be partially ordered. Events with static properties in their Boolean conditions must appear in any order at the beginning of processing schedule, but a subset of events with active properties must appear according to the dependencies among the active properties.

– If events on class *C* occur in serial mode, and:
  - there are not active properties in any Boolean conditions associated with the events, then the order in which events appear in the processing schedule must be the same as the order in which they appear in the input active query;
  - there are active properties in at least one Boolean condition associated with the events, then the order of the events must not be altered. If an event in the input active query depends on the change of an active property affected by another event that appears after in the input active query, then the execution of reactive processing is aborted.

  Additionally, before the processing schedule is constructed, a preprocessing of the input active query can be done. This preprocessing consists on the aggregation of the query sub-goals in order to minimize the size of query, according to the following criteria:
  - events that affect a class under the same mode - serial or parallel - are aggregated in only one sub-goal;
  - when the set of events affect a class following the parallel mode, duplicated events are deleted from the input active query;
  - otherwise, sub-goals remain the same as they were in the input query.

4 ActSPARQL and ActRDFS

In this section we present the extensions of SPARQL and RDFS to express and process reactivity. We extend the SPARQL query language with the *when* clause to specify the events that fire reactive process. Table 1 specifies the syntax of the ActSPARQL language.

We illustrate the proposed extension of SPARQL with the following query. It uses two execution modes: serial to represent events that occur is sequence, and parallel for events that occur simultaneously.

```
PREFIX act: < http:// facyt.uc.edu/action1.0/>
PREFIX nd: < http://funvisis.gov.ve/naturalDisaster/ >
SELECT ?rv
WHERE { ?x nd:evacuationScale ?rv }
{{{?e act:event ?o. ?ap act:reactiveBehavior ?e .?ap act:activeProperty ?c .
   Filter (?e = naturalDisaster_ocurrs  &&  ?c=Phenomenan)  }
 SERIAL
  {?e act:event ?o.?ap act:reactiveBehavior ?e .?ap act:activeProperty ?c .
```

```
  Filter (?e = wind_goes_up  &&  ?c=Hurricane)  } }.
{{?e act:event ?o.?ap act:reactiveBehavior ?e.?ap act:activeProperty ?c .
    Filter (?e = tsunamiGenerated  &&  ?c=IndonesiaIslands)  } }
 PARALLEL
 {?e act:event ?o.?ap act:reactiveBehavior ?e.?ap act:activeProperty ?c .
  Filter (?e = shallowEpicenter  &&  ?c=Earthquake) } }}
```

Table 1. The ActSPARQL Syntax (BNF format)

<ActSPARQL query>	::= <SPARQL query><When-clause>
<When-clause>	::= <EventBasicPattern>|
	<EventBasicPattern><Oper><When-clause>
<Oper>	::= SERIAL | PARALLEL
<EventBasicPattern>	::= {<BasicPattern><FILTER>}
<BasicPattern>	::= <VAR> action:event < VAR>.
	<VAR>action:reactiveBehavior <VAR>.
	<VAR> action:activeProperty <VAR>.
<FILTER>	::= Filter (<VAR> = <ClassRDF> && <VAR> = <ClassRDF>) .

The above query retrieves *evacuation scale* of all members of *Phenomenan*, *Hurricane*, *IndonesiaIslands* and *Earthquake* classes after is fired the execution of the reactive processing of the events *naturalDisaster_ocurrs*, *wind_goes_up*, *tsunamiGenerated*, *shallowEpicenter*, respectively. The ActSPARQL engine receives two requests. First, to execute the event *wind_goes_up* on the class *Hurricane* after the execution of the event *naturalDisaster_ocurrs* on the class *Phenomenan*, two sets of rules are evaluated. Second, to execute in parallel, events *tsunamiGenerated*, *shallowEpicenter* on classes *IndonesiaIslands* and *Earthquake*, the reactive process evaluates one set of rules. The query engine interprets the active clause *when* of the query, and processes the reactive behavior of the data in the ontology O. Thus, for example, if the value of the property *evacuationScale* is *Null* before the event *tsunamiGenerated* occurs, then the execution of the *when* clause alters the value of property *evacuationScale* changing this value to *Full*. Once the reactive processing is evaluated on ontology O, the engine returns ontology $O\prime$, which can be queried using any SPARQL query engine.

Additionally, we propose a dialect of RDFS to express ACTION ontologies. Some of the new properties are listed in Table 2. Predicate *isSubEvent* is transitive and its semantics is implemented at the deductive database level as an implicit predicate.

5 Experimental Study

In this section we present the results of our experimental study on the performance of the proposed evaluation techniques. We report on the evaluation time and on the number of derived facts. We compare the bottom-up evaluation of the IMR rewritings to

Table 2. Some of the ActRDFS Property Descriptions

RDFS property	Description
(e action:event)	e is an event of ontology o
(e1 action:isSubEvent e2)	e1 is a sub-event of e2
(p rdf:type activeProperty)	p is an active Property
(p action:activeProperty e)	p is an active of event e
(bc action:BooleanExpression e)	bc is the Boolean expression of event e
(p action:changedTo v)	active property p changes to v

the bottom-up evaluation of the program rewritten by using traditional Magic Sets techniques. We present the performance of the IMR method with and without the reordering of sub-goals of active queries.

5.1 Experiment Configuration

Dataset: The experimental study was conducted on the Lehigh University Benchmark (LUBM) [14]. We have extended the instance generator LUBM to insert active properties and events. The *univ_num* parameter (number of universities to generate) of the generator program, was used to construct the different dataset sizes. Once the instances of the repositories were generated as RDF/RDFS/XML documents, this information was translated into meta-level predicates of ADOB by means of Prologs DCG (Definite Clauses Grammars). To compare IMR (with and without the reordering of sub goals) versus classic magic set evaluation [5], we consider a dataset with three kinds of repositories: small (information of five university), medium (twenty universities) and large (fifty universities). Ten different reactive goals (queries) were posed to each kind of repository; each of them evaluated using classic magic set evaluation and IMR. In Table 3, the generated repositories are described in terms of the number of classes, static and active properties.

Hardware and Software: The experiments were evaluated on a Solaris machine with Sparcv9 1281 MHz processor and 16GB of RAM. The proposed algorithms have been implemented in SWI-Prolog, Version 5.6.54.

Metrics: We report on the following metrics:

- Total Number of Derived Facts (TNDF): the cost of the tasks of reasoning and query evaluation are measured in terms of the number of derived facts needed for the reactive processing. The TNDF corresponds to the size of the minimal model.
- TIME: measures the time in seconds required to achieve the reactive processing.

Transformations: We study the performance of our approach on different active datasets. We apply two kinds of transformations to the event hierarchy of the documents: *AddBind* and *Bind*.

AddBind Transformation: each *AddBind* transformation Ti adds three $isSubEventOf$ relationships to the event hierarchy generating a more densed hierarchy. These transformations augment intersections between events.

Bind Transformation: each *Bind* transformation Ti adds one *reactiveBehavior* relation to three events which are randomly chosen, and increases the number of active properties that are affected by these events. *Bind* transformations increase the complexity of the Boolean condition to be evaluated in the reactive processing.

Table 3. DataSet Description

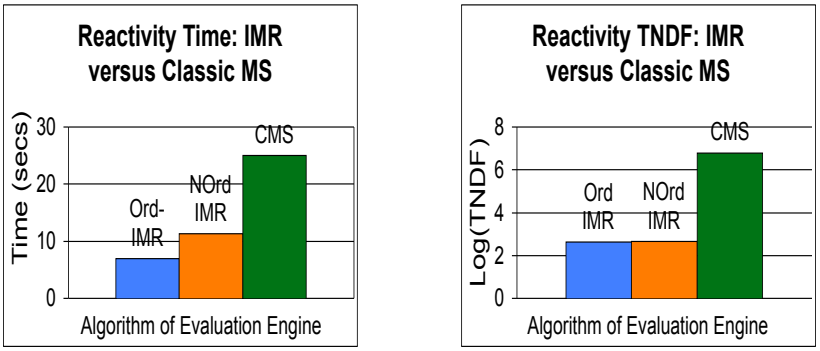
#Univ	#Class-Inst	# Prop-Inst	#ActProp-Inst	MB
5	91,408	325,429	69,441	33.5
20	414,194	1,305,736	278,876	151.0
50	978,764	4,212,872	741,150	379.9

5.2 Results

We considered ten queries. Each query consists of three to nine sub-goals, one to ten events affecting one class, one to five different affected classes in the query, and one random execution scheduler per sub-goal - serial or parallel. Figure 3 shows the average TIME and TNDF for the bottom-up evaluation of ten input programs with Classical Magic Sets rewritings (CMS) that represent the reactive processing versus the bottom-up evaluation of the IMR rewriting of the same programs. In Figure 3(a) we observe that non-ordering sub-goals IMR method (NOrdIMR) speeds up the tasks of reasoning and query evaluation by 55% w.r.t. CMS, while the ordering sub-goal IMR method (OrdIMR) reduces evaluation time by 72% w.r.t. CMS. Figure 3(b) also shows that both versions of IMR outperform CMS by four orders of magnitude (TNDF). The reason is that the IMR method avoids duplicate inferences when it processes a set of events. In contrast, the classic Magic Sets method makes all inferences for each event of the set.

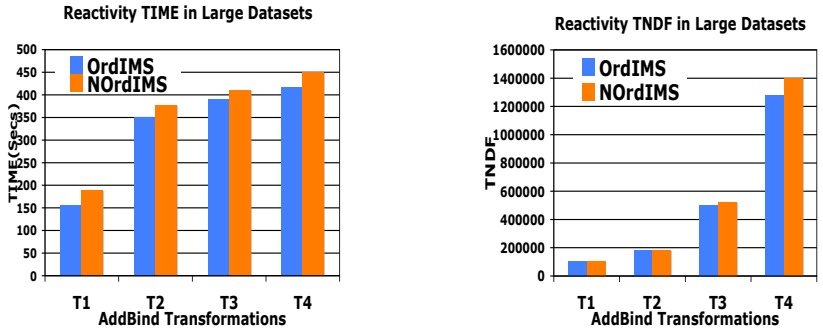
Figures 4(a) and 4(b) respectively compare values of TIME and TNDF for the bottom-up evaluation of Magic Sets rewritings (ordered versus non-ordered sub-goals) of ten queries when AddBind transformations are applied. In Figure 4(a) we observe that OrdIMR always requires less TIME to process reactivity. This is because this method reduces the number of query sub-goals, i.e., original sub-goals are aggregated. In Figure 4(b) values of TNDF are similar in both techniques because AddBind transformations do not increase the number of events.

Figure 5 compares values of TIME and TNDF for ten queries when Bind transformations were applied. In Figure 5(a) we observe that the ordering sub-goal IMR method requires, in average, 10% less TIME to process reactivity. The difference between TNDF in both techniques is 5% (Figure 5(b)); this may be because, Bind transformations augment the number of active properties affected by each event, and the event hierarchy is always the same. Thus, the ordering sub-goal IMR method consumes less TIME and TNDF because it processes queries with a small number of sub-goals; however, the difference is small because the number of events remains the same.



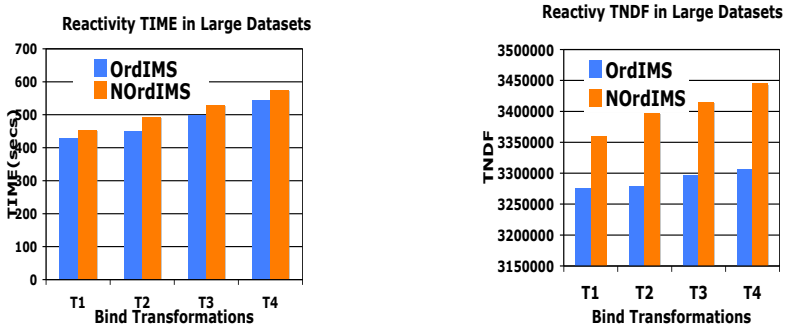
(a) (b)

Fig. 3. Cost of ordering sub-goals IMR method(OrdIMR), non-ordering sub-goals IMR method(NOrdIMR), Classical Magic Sets (CMS): a) TIME (secs) b) TNDF(log-scale)



(a) (b)

Fig. 4. Cost of ordering sub-goals IMR method(OrdIMR) and non-ordering sub-goals IMR method-AddBinds Transformations:a) TIME-secs; b) TNDF



(a) (b)

Fig. 5. Cost of ordering sub-goals IMR method(OrdIMR) and non-ordering sub-goals IMR method(NOrdIMR)-Bind Transformations:a) TIME-secs; b) TNDF

6 Conclusions and Future Work

ACTION ontologies extend the expressiveness of ontological languages in order to incorporate events as first-class concepts, and make use of traditional deductive reasoning tasks to manage reactivity. In order to efficiently process reactive behavior the REACTIVE architecture was proposed. This platform integrates ActRDFS and an extension of SPARQL namely ActSPARQL that allows expressing a set of events that occur in sequence or in parallel; in addition, an evaluation engine to efficiently manage the reactive behavior of ontological data was developed. Results of the conducted empirical study indicate that our approach is an alternative - declarative - way to express and process reactivity. In the future we plan to integrate REACTIVE to existing SPARQL query engines.

References

1. J. J. Alferes and R. Amador. r^3- a foundational ontology for reactive rules. In *OTM Conferences (1)*, pages 933–952, 2007.
2. J. J. Alferes, R. Amador, P. Kärger, and D. Olmedilla. Towards reactive semantic web policies: Advanced agent control for the semantic web. In *International Semantic Web Conference (Posters & Demos)*, 2008.
3. J. Bailey. Transformation and reaction rules for data on the web. In *ADC '05: Proceedings of the 16th Australasian database conference*, pages 17–23, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.
4. J. Bailey, G. Papamarkos, A. Poulovassilis, and P. T. Wood. An Event-Condition-Action Language for XML. In *Web Dynamics*, pages 223–248. 2004.

5. F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic Sets and Other Strange Ways to Implement Logic Programs. In *PODS*, pages 1–15, 1986.
6. E. Behrends, O. Fritzen, W. May, and F. Schenk. Embedding Event Algebras and Process for ECA Rules for the Semantic Web. *Fundam. Inform.*, 82(3):237–263, 2008.
7. Y. Biletskiy, D. Hirtle, and O. Vorochek. Toward the Identification and Elimination of Semantic Conflicts for the Integration of RuleML-based Ontologies. In *CSWWS*, pages 135–142, 2006.
8. A. Bonifati, S. Ceri, and S. Paraboschi. Active rules for XML: A new paradigm for E-services. *VLDB J.*, 10(1):39–47, 2001.
9. A. Bonifati and S. Paraboschi. Active XQuery. In *Web Dynamics*, pages 249–274. 2004.
10. D. Braga, A. Campi, D. Martinenghi, and A. Raffio. ActiveXQBE: A Visual Paradigm for Triggers over XML Data. In *EDBT Workshops*, pages 865–875, 2006.
11. F. Bry, M. Eckert, H. Grallert, and P.-L. Patranjan. Evolution of Distributed Web Data: An Application of the Reactive Language XChange. In *ICDE*, pages 1517–1518, 2007.
12. E. Cohen. Estimating the Size of the Transitive Closure in Linear Time. In *FOCS*, pages 190–200, 1994.
13. M. Fox and M. Gruninger. On Ontologies and Enterprise Modelling. *The AI Magazine*, pages 109–121, 1997.
14. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.*, 3(2-3):158–182, 2005.
15. D. Harel, D. Kozen, and J. Tiuryn. Dynamic logic. In *Handbook of Philosophical Logic*, pages 497–604. MIT Press, 1984.
16. I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3C Member Submission, 21 May 2004. Available at http://www.w3.org/Submission/SWRL/.
17. Z. Huang, S. Schlobach, F. van Harmelen, N. Casellas, and P. Casanovas. Dynamic aspects of opjk legal ontology. In *Computable Models of the Law, Languages, Dialogues, Games, Ontologies*, pages 113–129. 2008.
18. Z. Huang and H. Stuckenschmidt. Reasoning with multi-version ontologies: A temporal logic approach. In *International Semantic Web Conference*, pages 398–412, 2005.
19. M. Levene and A. Poulovassilis. Special issue on Web dynamics. *Computer Networks*, 50(10):1425–1429, 2006.
20. M. Morgenstern. Active Databases as a Paradigm for Enhanced Computing Environments. In *VLDB*, pages 34–42, 1983.
21. M. O'Connor, R. Shankar, and A. Das. An Ontology-Driven Mediator for Querying Time-Oriented Biomedical Data. In *CBMS '06: Proceedings of the 19th IEEE Symposium on Computer-Based Medical Systems*, pages 264–269, Washington, DC, USA, 2006. IEEE Computer Society.
22. A. Poulovassilis, G. Papamarkos, and P. T. Wood. Event-Condition-Action Rule Languages for the Semantic web. In *EDBT Workshops*, pages 855–864, 2006.
23. E. Ruckhaus, E. Ruiz, and M. Vidal. Query Evaluation and Optimization in the Semantic Web. *TPLP*, 2008.
24. E. L. Tovar and M.-E. Vidal. Magic Rewritings for Efficiently Processing Reactivity on Web Ontologies. In *OTM Conferences (2)*, pages 1338–1354, 2008.
25. W. Xing, O. Corcho, C. Goble, and M. Dikaiakos. Active Ontology: An Information Integration Approach for Highly Dynamic Information Sources. In *European Semantic Web Conference 2007 (ESWC-2007)*, Innsbruck, Austria, June 2007. Poster.