An Active Domain Node Architecture for the Semantic Web

Franz Schenk and Wolfgang May

Institut für Informatik, Universität Göttingen,
{schenk,may}@informatik.uni-goettingen.de

Abstract. We present an architecture for *application nodes for the Semantic Web* (SWAN). The underlying principle in SWAN is the specification of actions and events as dynamic aspects of the application. This complements the framework *Modular Active Rules for the Semantic Web* (MARS), where the communication between services is based on the notions of events and (requests of) domain-level actions. Such a model allows to define workflows on the ontology level. While MARS offers the service infrastructure needed for processing the workflow, SWAN is an architecture for applications in a rule-driven environment. Basically, SWAN consists of a hybrid OWL/F-Logic knowledge base, augmented with active rules. Using SWAN, only a set of rules is needed in order to deploy a new application. A prototype implementation of the architecture exists that shows the flexibility and applicability of its concepts.

1 Introduction

The World Wide Web as we know and use it today makes a steadily growing amount of services available. Often, the problem is not that a service for a specific purpose does not exist. Rather, it is hard to know where it exists or how to get a machine-readable specification of its interface. This is one of the promises of the Semantic Web: An interaction of services based on the notions of the domain ontology rather than on the syntactic specifications of service calls.

Several approaches (e.g., OWL-S [10], WSMO [14], WSDL-S [15]) try to fill this gap by giving a vocabulary for the specifcation of the properties and semantics of Web services. Hereby, Web Services for specific purposes can be discovered if descriptions according to a separate *service ontology* vocabulary are used.

In contrast, in our approach, the *tasks*, i.e., the *actions* themselves are part of the ontology. An action request is an instance of a certain class (which is a subclass of Action) with its properties. It is represented by a (small) RDF graph that references resources in the domain as its properties/parameters. Similarly, *events* are also part of the domain ontology. The communication between services according to such an ontology does not need additional service descriptions, but is fully declarative.

Structure of the Paper. The domain ontology meta model underlying our approach will be discussed in the next section, together with a short overview

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part II, LNCS 5871, pp. 1095–1102, 2009.

of the overall MARS infrastructure that provides the global service framework. Section 3 introduces the architecture of SWAN and its components, which is the main contribution of this paper. Finally, a look on related work is given in Section 4, followed by conclusions in Section 5, which completes this paper.

2 Ontologies with Actions and Events

Active Rules and Global Architecture. In *event-driven architectures*, large parts of the specification of the behaviour in the application domain are based on active rules, i.e., *Event-Condition-Action (ECA) rules*: Upon an event, evaluate a condition (that can include queries) and, if satisfied, execute an action. The *Modular Active Rules for the Semantic Web* (MARS) framework [2] provides a Web Service infrastructure for ECA rules. Each component of a rule (event, condition or action) can be specified in an arbitrary component sublanguage as long as a language service is available. The overall MARS infrastructure for processing active rules consists of autonomous nodes of different types:

- Language nodes: they support domain-independent specification languages, such as ECA rules, composite event specification formalisms (e.g. SNOOP event algebra [4]), query languages, or process languages (e.g. CCS process algebra [13]).
- Infrastructure nodes: they provide infrastructure as mediators between language nodes and domain nodes, such as domain brokers that implement a portal functionality for a given domain based on its ontology, and *Language and Service Registries* [5] that serve for finding language or domain nodes.
- Domain nodes: they carry out the real "businesses", e.g., airlines and car rental companies. Domain nodes are able to answer queries, to execute actions of the domain, and they emit events of the domain. SWAN offers an architecture for this kind of nodes.

Domain Ontologies. Domain ontologies (e.g. for travelling) define *static notions* for the description of concepts, their properties, and their relationships. In our approach, also the *dynamic notions* of actions and events are modeled as classes in the domain ontology. Having events as first-order citizens of the ontology especially allows for a declarative, event-driven specification of behaviour. Additionally, having actions and events in the ontology allows to correlate preconditions, postconditions etc. in a suitable meta language.

Concrete instances of actions and events are RDF graphs containing parameters that usually refer to URIs in the domain. These instances are *volatile* as they do not exist in any knowledge base state, but as information sent between services. The execution of an action instance by a node changes the internal state of the node's knowledge base. Such changes are –as usual– visible in an indirect way by *pull communication* when users or other services state queries against it, and additionally become visible as *events* by *push communication*.

Example 1 (Actions and Events in the Travel Ontology). *The travel ontology defines concepts of the travel domain. For instance, concepts of flights are specified by both the* general *flight connections (e.g. OF123 from Frankfurt (FRA) to Lisbon (LIS)) and concrete flights for a certain day (e.g. OF123 on November 1, 2009) that can be booked (we use Turtle syntax and slightly abuse prefix notation with hierarchical local parts in the presentation):*

```
travel:airlines/OF a travel:Airline.
travel:Flights/OF123 a travel:Flight;
   travel:from travel:airports/FRA; travel:to travel:airports/LIS;
   travel:departure "13:50";  travel:arrival "15:00";
   travel:operatedBy travel:airlines/OF.
travel:Flights/perDay/OF123/20091101 a travel:concreteFlight;
   travel:hasFlightNo travel:Flights/OF123;  travel:date "2009-11-01";
   travel:airplane travel:airlines/OF/planes/ikarus.
travel:airlines/OF/planes/ikarus a travel:Airplane;
   travel:name "Ikarus"; travel:type travel:Airplanes/Airbus/A333.
travel:Airplanes/Airbus/A333 travel:maxSeats 199.
travel:Action rdfs:subClassOf mars:Action.
travel:DoFlightBooking rdfs:subClassOf travel:Action ;
   mars:belongs-to-domain  travel: .
```

The following RDF graph is an instance of the action travel:DoFlightBooking, which requests a booking for customer John Doe for flight OF123 on November 1, 2009:

```
 [ a travel:DoFlightBooking;  travel:passenger persons:john-doe;
   travel:flight travel:Flights/OF123; travel:date "2009-11-01" ].
```

Event types in the travel ontology are for instance travel:FlightDelayed (for a concrete flight, with information about the reason and the expected delay), or, as shown in this example, travel:FlightBooked (for a concrete flight and a person):

```
travel:event rdfs:subClassOf mars:Event .
travel:FlightBooked rdfs:subClassOf travel:TravelBooked .
```

Assume that the travel:DoFlightBooking action becomes executed by the OntoFlight airline service. As a result, it raises a travel:FlightBooked event:

```
 [ a travel:FlightBooked;  travel:passenger persons:john-doe;
   travel:flight travel:Flights/OF123; travel:date "2009-11-01" ].
```

An event is local at the application (e.g., the airline service). Additionally, it can be *raised* in order to make the event visible to the application domain. Hereby, it can be used for triggering further tasks in a workflow.

Domain Brokering and Global Communication. The inter-service communication consists only of actions, events (and queries) on the ontology level. Neither actions nor events are addressed explicitly to specific services. Rather, they are distributed by domain brokers, which are part of the infrastructure. Basically, domain brokering is a matching of action or event names with the information

about services. This information comprises the kinds of actions they support, or which *classes* of events they are interested in. Domain brokering is described in more detail in [1].

Communication by XML Messages. A close look on actions and events reveals that they have a janus face: from one point of view they are notions, which carry the meaning of dynamic aspects of an application domain. In this respect they are best represented as RDF graphs *within* services. But, with regard to the communication *between* services, they are, at the same time, simply messages. As messages, they become serialised to XML, using fixed, application specific DTDs. This transformation ensures that also classical (XML-based) Web Services can be recipients of these messages. In many cases, it is not only more efficient but simply the only possibility to handle events or actions on a purely syntactical level.

The following example illustrates how actions, events and active rules can be combined.

Example 2. *Consider again the booking for John Doe for flight OF123 on November 1, 2009. An instance of the action class travel:DoFlightBooking is submitted as an RDF/XML fragment "into the application domain" (i.e., to an appropriate domain broker). All application nodes that are known to support that action (like OntoFlight) will receive a copy. The recipient has to translate the action into a local update on its knowledge base (this translation is one of the main contributions of this paper, see Section 3). Afterwards, a travel:FlightBooked event is raised by the OntoFlight application node (sent to the domain broker).*

ECA rules may now react upon this event. For instance, there is the car rental company OntoRent (another application domain node), which offers car rental services to persons at the destination city of their travel, provided that OntoRent has a branch there. OntoRent has registered the following ECA rule at the the ECA rule evaluation service (only given in its abstract form here):

ON travel:FlightBooked(...)
WHEN *OntoRent has a branch at the destination of the flight and cars are available on that date*
DO travel:PrereserveCar(...) *(and send offers to customer)*

The ECA rule engine will react upon the detection of the travel:FlightBooked event and evaluate the condition of the ECA rule. If the condition is satisfied, the resulting action part of the rule will be emitted for execution. Via the domain broker, OntoRent will be a recipient of the action request. The successful execution of that action (a preliminary reservation of a car) at the car rental domain node will in turn cause travel:CarPrereservation events to be raised for the purpose of notifying the customer about pre-reservations of cars at the destination of their travel. The customer can react on them, e.g., having a rule registered that collects all prereservations and chooses the least expensive offer for actual reservation.

So far, the domain ontology, which describes events, actions, and concepts of the application domain, and the MARS service infrastructure for domain brokering

and ECA rule execution have been explained. For an event-driven Semantic Web service infrastructure, an application node architecture that translates domain-level actions into knowledge base updates and emits events in return is needed. SWAN provides such an architecture.

3 The Swan Web Node Architecture

The contribution of this paper is the SWAN architecture for a *Semantic Web Application Node*. SWAN is a Web Service architecture for applications in an event-driven environment. As described above, in this environment, there are no explicit service calls, rather the execution of (high-level) actions are requested. These requests do not specify where (by which service) or how (by which update procedure) the action is actually implemented. The matching of actions is done by the domain brokers based on the shared vocabulary of a domain ontology. Actions are communicated to those services that are known to support a specific action.

SWAN Components. Internally, SWAN uses an OWL knowledge base for the local representation of its state. This knowledge base can be queried (in SPARQL), and there is a local update language RDFU (with well-defined semantics) for executing updates on the RDF graph level. Moreover, local active rules in form of knowledge base triggers are available for maintaining knowledge base integrity and update completion. Additionally, triggers are used for the *raising* of events to the application domain, making changes to the local state globally visible. For example, after the execution of a travel:DoFlightBooking action the domain node can raise a travel:FlightBooked event (as shown in Example 1). Furthermore, OWL reasoning is supplemented by an F-Logic [9] rule-based engine to provide a hybrid reasoning mechanism. Details about these knowledge base components can be found in [11] [12].

Much in contrast to other hybrid reasoning systems for the Semantic Web (where different reasoning systems are combined *continuously*), SWAN combines knowledge base triggers with hybrid reasoning into an *on-demand* hybrid reasoning system. Triggers allow for precise specifications of those changes in the knowledge base where a hybrid reasoning process should be initiated. The rationale behind this design is that rule-based completions are often needed only for small parts of the knowledge base. Hereby, a distinction is made between *background facts* (like the flight schedule) and *dynamic facts* (like the actual bookings). Depending on the trigger specifications, hybrid reasoning can be initiated exactly in situations where changes on *background facts* occur. Hereby, the overall performance of the application node improves significantly because the additional reasoning service is only used when actually needed.

ACA: Mapping Actions to Knowledge Base Updates. Central to the concept of SWAN is its integration into an event-driven framework. This is realised by mapping of high level actions to knowledge base updates by the use of ACA rules.

ACA rules provide a modularising layer between the global ontology and the potentially different internal technologies of the Semantic Web services. They allow to use high-level notions for a declarative specification of application domain behaviour that are then implemented locally.

Incoming action requests are translated by a rule-based mapping mechanism, called *ACA (Action-to-Action) wrapper*, into RDFU update operations. In the current implementation, either XQuery or XSLT can be used to express ACA rules in SWAN. In both cases, the "input document" to the XSLT or XQuery engine, respectively, is the RDF/XML fragment[1].

The condition part of the ACA rule can be specified in form of an rdfu:condition element. The query attribute of the condition element will be evaluated as a SPARQL query on the local knowledge base. The result is bound to variables which can be used in the RDFU update element. If no result tuples are returned the condition does not hold and no update will be performed. Note that variables are denoted by '?' in SPARQL while in XQuery the '$' sign is used for variables. The substitution of variables is handled by the ACA rule engine.

Example 3. *Consider again the domain action* travel:DoFlightBooking *from Example 1, which will be translated by the following ACA rule into an* RDFU *update statement:*

```
let $booking in //travel:DoFlightBooking,
    $flight := $booking/travel:flight,
    $person := $booking/travel:passenger,
    $date := $booking/travel:date
return
  <rdfu:condition rdfu:query="
        SELECT ?connection WHERE{{
           ?connection travel:hasFlightNo $flight.
           ?connection travel:date $date.
           ?connection travel:hasNoOfBookings ?bookings.
           ?connection travel:airplane ?plane.
           ?plane travel:type ?planeType.
           ?planetype travel:maxSeats ?capacity.
        FILTER (?bookings &lt; ?capacity).}}">
    <rdfu:insert>
      <rdf:subject rdf:about="{$connection}"/>
      <rdf:predicate rdf:about=
             "http://www.semwebtech.org/domains/2006/travel#hasBooking"/>
      <rdf:object rdf:about="{$person}"/>
    </rdfu:insert>
 </rdfu:condition>
```

The let clause binds DoFlightBooking *elements, which represent the incoming high-level actions. Parameters of the action are bound to XQuery variables. The condition ensures both the existence of the flight and the availability of seats. The*

[1] Recall that we expect the fragment to conform to a certain DTD such that its structure can be assumed to be fixed.

result of the SPARQL query (if there is any) is bound to the variable ?connection. If the condition is satisfied, the included RDFU update actions are executed on the knowledge base.

4 Related Work

The SWAN and MARS architectures use ECA rules for the definition of reactive behaviour of an application domain. These rules are conceptually on an abstract level where no implementation details of the application domain have to be provided. This is different from conventional web services, which are invoked by procedure calls or application-specific commands for data storage or data manipulation. In our work, the concepts of domain ontologies are used in order to specify the *meaning* of an action or an event. It is left to the application how to translate and execute these specifications. A similar level of abstraction can be found, for example, in the object-oriented programming language Smalltalk [6]. Another form of event-based application-independent infrastructures can be found with Event-Notification-Services (ENS) and Publish-Subscribe systems [3,7]. Most of these approaches try to tackle the technical problems of event processing, while abstract concepts are rare. For example, in [8] the authors propose a meta-service for event notification. Here, events can be specified completely independent from the application domain services that will eventually process the subscriptions. This can be compared to the use of ACA rules in SWAN, where abstract action specifications are transformed into knowledge base updates of the application. In SWAN, though, the mapping is realised by the application domain node and not by a meta-service.

5 Conclusions

SWAN *architecture.* We presented a rule-based architecture for application nodes in the Semantic Web. The architecture realises an active OWL knowledge base, which exhibits a number of distinct features novel to this kind of knowledge management application: ontologies are extended with notions of actions and events that describe the activities and visible effects in the domain. Communication between services is completely based on these notions. Global, reactive behaviour is then preferably specified by ECA (Event-Condition-Action) rules. Domain actions are expressed in terms of the domain ontology instead of explicit update commands of a specific application service. The actual execution of domain actions then relies on the definition of translation rules, called ACA rules, mapping domain actions to internal knowledge base updates. Moreover, low-level triggers can be used for the raising of events which fully integrates the domain node into an event-driven environment. The behaviour of SWAN nodes is thus specified completely in a rule-based way.

As an orthogonal functionality, SWAN combines the use of an OWL reasoner with a rule-based F-Logic reasoner for supplementary deductions. The hybrid reasoning also profits from the active features of SWAN as it can be initiated on-demand by using knowledge base triggers.

Advantage of Rule-Based Specification. In our proposal for a Semantic Web service architecture, a new service can be quickly instantiated (and also reconfigured and finetuned). It is sufficient to create an instance of a SWAN node template and to initialise it by registering the sets of knowledge base triggers, transformation rules, and logical derivation rules along with the application node ontology. The rules completely determine the behaviour of the application node. Even more, since for a fixed ontology there is a large set of common ACA rules, triggers, and static axioms, a preconfigured SWAN node can be taken and *incrementally* completed with more specific behaviour. The flexibility and applicability of this concept has been proven with a protoype implementation.

References

1. Behrends, E., Fritzen, O., Knabke, T., May, W., Schenk, F.: Rule-Based Active Domain Brokering for the Semantic Web. In: Marchiori, M., Pan, J.Z., de Marie, C.S. (eds.) RR 2007. LNCS, vol. 4524, pp. 250–268. Springer, Heidelberg (2007)
2. Behrends, E., Fritzen, O., May, W., Schenk, F.: Embedding Event Algebras and Process Algebras in a Framework for ECA Rules for the Semantic Web. Fundamenta Informaticae (82), 237–263 (2008)
3. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and Evaluation of a Wide-Area Event Notification Service. ACM TOCS 19, 332–383 (2001)
4. Chakravarthy, S., Mishra, D.: Snoop: An Expressive Event Specification Language for Active Databases. Data Knowledge Engineering 14(1), 1–26 (1994)
5. Fritzen, O., May, W., Schenk, F.: Markup and Component Interoperability for Active Rules. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 197–204. Springer, Heidelberg (2008)
6. Goldberg, A., Robson, D. (eds.): Smalltalk: The Language. Addison Wesley, Reading (1989)
7. Hinze, A., Voisard, A.: A Parameterized Algebra for Event Notification Services. In: TIME, p. 61. IEEE Computer Society, Los Alamitos (2002)
8. Jung, D., Hinze, A.: A Meta-service for Event Notification. In: Meersman, R., Tari, Z. (eds.) OTM 2004. LNCS, vol. 3290, pp. 283–300. Springer, Heidelberg (2006)
9. Kifer, M., Lausen, G.: F-Logic: A higher-order language for reasoning about objects, inheritance and scheme. In: SIGMOD, pp. 134–146 (1989)
10. Martin, D., Paolucci, M., McIlraith, S.A., Burstein, M., McDermott, D., McGuinness, D.L., Parsia, B., Payne, T.R., Sabou, M., Solanki, M., Srinivasan, N., Sycara, K.: Bringing semantics to Web services: The OWL-S approach. In: Cardoso, J., Sheth, A.P. (eds.) SWSWPC 2004. LNCS, vol. 3387, pp. 26–42. Springer, Heidelberg (2005)
11. May, W., Schenk, F., Kattenstroth, H.: Combining OWL with F-Logic Rules and Defaults. In: ALPSWS, CEUR Proc. vol. 287, pp. 60–75 (2007)
12. May, W., Schenk, F., von Lienen, E.: Extending an OWL Web Node with Reactive Behavior. In: Alferes, J.J., Bailey, J., May, W., Schwertel, U. (eds.) PPSWR 2006. LNCS, vol. 4187, pp. 134–148. Springer, Heidelberg (2006)
13. Milner, R.: Calculi for Synchrony and Asynchrony. In: Theoretical Computer Science, pp. 267–310 (1983)
14. Roman, D., de Bruijn, J., Mocan, A., Lausen, H., Domingue, J., Bussler, C.J., Fensel, D.: WWW: WSMO, WSML, and WSMX in a Nutshell. In: Mizoguchi, R., Shi, Z.-Z., Giunchiglia, F. (eds.) ASWC 2006. LNCS, vol. 4185, pp. 516–522. Springer, Heidelberg (2006)
15. Web Service Semantics (WSDL-S) W3C Submission, `http://www.w3.org/Submission/WSDL-S/`