[Semantic Web] Services: Comprehension and Composition

Reto Krummenacher, Barry Norton, and Adrian Marte

Semantic Technology Institute, University of Innsbruck, Austria
firstname.lastname@sti-innsbruck.at

Abstract. The combination of semantic technology and Web Services in form of 'Semantic Web Services' has been oriented towards extending the WS-* stack with ontology-based descriptions. We re-dub these 'Semantic [Web Services]' to underline their emphasis on Web Services. The same time, there is a strong movement away from this stack - for which the 'Web' part is little more than branding - towards a RESTful model of services. The Linked Open Data (LOD) initiative is a prominent adopter of this approach and exposes many datasets via SPARQL endpoints and dedicated RESTful services. Our '[Semantic Web] Services' model accommodates WS-* stack-based services, RESTful services and LOD endpoints with service descriptions based on SPARQL to aid their comprehension as consumers and producers of RDF data. Along the way, we show how the existing link between service messaging and the semantic viewpoint, commonly called 'lifting and lowering', is usually unduly restricted to ontology-based classification and misses how the effect of a service contributes to the knowledge of its consumer. Our SPARQL-based approach helps in the composition of services as processes that enhance the Semantic Web by making such knowledge available.

1 Introduction

Thus far in the literature the combination of semantic technologies and Web services to form 'Semantic Web Services' has been oriented towards extending the WS-* stack (based on the XML-based languages WSDL and SOAP, and built on to form UDDI and BPEL) with ontology-based descriptions [1–3]. Their claimed benefits are centred on an increased degree of automation in high-level tasks such as the discovery, composition and mediation of services that are a priori already explicit aims of the Web Service approach. We re-christen these 'Semantic [Web Services]' to underline their emphasis on the Web services worldview. At the same time, however, there is a strong movement away from this underlying stack for which the 'Web' naming is acknowledged as little more than branding towards a resourceful ('RESTful' [4]) model of services [5]. The RESTful model adopts the Web resource model of HTTP and eschews the WS-* stack languages as cumbersome and redundant. Among the prominent adopters of this approach is the Linked Open Data community who, meanwhile, has been seeking to increase the amount of Semantic Web-style content, in the form of RDF, online. As well as dedicated RESTful services, Linked Open Data (LOD)[1] exposes many datasets via passive SPARQL endpoints.

[1] http://www.linkeddata.org

Our approach of '[Semantic Web] Services' (abbreviated [SW]S, not SWS) combines WS-* stack-based services, RESTful services and LOD endpoints, and makes service descriptions based primarily on SPARQL that show how each service is a consumer and producer of RDF data, and thereby services the Semantic Web. To this end, we discuss how existing links between service messaging (usually XML-based) and the semantic viewpoint, commonly referred to as 'lifting and lowering' [6], are usually unnecessarily restricted to ontology-based classification and thus miss to identify how the effect of a service contributes to the knowledge of its consumer.

In this paper we propose a SPARQL-driven approach to service composition that fosters the comprehension of process creation and execution, and that facilitates services' contribution to the Semantic Web in which resulting knowledge is eventually exposed. The paper first takes a look at the state-of-the-art and thus the "traditional" way of composing Web Services, in Section 2, and thereby Semantic [Web Services], in Section 3. In contrast to these naturally XML-driven approaches to the creation of processes, we present in Section 4 our semantic-minded approach: services are described by means of SPARQL constructs as being first RDF consumers and then RDF producers. As such composition can be fully done at the semantic level, and the outcome of a service invocation directly contributes to the Semantic Web, thus the name [Semantic Web]Service composition. After the presentation of the basic concepts and technicalities of our approach, we depict in Section 5 a first implementation and discuss some insights gained from initial experimentation with WS-* stack-based and RESTful services, as well as Linked Open Data. The paper is concluded with Section 6.

2 Web Service Comprehension and Composition

Around 1998 Microsoft began collaborating on a scheme to use W3C's XML standard, which was perceived as open and independent, in form of *XML-RPC*[2] to facilitate language- and vendor-neutral communication between software across the Internet – extending the existing notion of Remote Procedure Call (RPC). With the publication of the *XML Schema* specification, which added a typing mechanism to XML definitions, this evolved to become the *Simple Object Access Protocol* (SOAP). With buy-in also from IBM, among others, SOAP version 1.1 was submitted to the W3C. The reuse of Web technologies to achieve a common agreement (both motivated to reinforce the vendor-neutrality and openness of the approach) are part of the reason the resulting software services are de facto referred to as *Web Services*. An important point was that, in contrast to the Common Object Model (COM) and the Common Object Request Broker Architecture (CORBA), which Microsoft and IBM had respectively previously pursued, the notion of an identifiable stateful object, to which procedures applied as methods, was dropped. Indeed, SOAP is no longer an acronym, avoiding the original mention of objects, for this reason. Version 1.2 of SOAP,[3] a W3C recommendation continued the buy-in from competing companies in the form of support from Sun and Oracle.

[2] `http://www.xmlrpc.com/spec`
[3] `http://www.w3.org/TR/soap12/`

In order to aid the *comprehension* of Web Services, XML was also used as the vendor-neutral basis for the description of SOAP v1.1 services in the W3C submission of the Microsoft- and IBM-led *Web Services Description Language* (WSDL). WSDL allows the collection of signatures for procedures, or *operations*, into abstract interfaces. Operations are described, according to their input and output messages, using XML Schema. In a further appeal to the Web standards movement, WSDL interfaces are identified by URIs. Although WSDL version 1.1 attained only the status of a member submission it received widespread tool support in industry. The follow-up submission – originally version 1.2, renamed WSDL 2.0 – is published as a W3C recommendation.[4] Again WSDL drops the notion of an identifiable, stateful object and binds interfaces to static, stateless *endpoints* (URLs).

In order to further aid comprehension of Web Service descriptions, building on WSDL, Universal Description, Discovery and Integration (*UDDI*) was standardised, not by W3C but by OASIS, the Organization for the Advancement of Structured Information Standards.[5] UDDI provides a registry of service descriptions and so brokers the services offered by providers against the requirements of businesses on the basis of supported interfaces, persistently identified by URI and described in WSDL. Service descriptions are also brokered on the basis of a 'yellow pages' by which service descriptions are organised into taxonomies, similar to product catalogues. In addition, UDDI allows meta-data to be attached to service descriptions. In the first instance Quality of Service (QoS) meta-data can be used to find the most fitting services to non-functional requirements. In the second instance 'white pages' provide information on the providers of services, and 'green pages' further technical details on their provision.

In order to allow *composition* of Web Services, WSDL was again built on primarily to overcome differences between Microsoft and IBM's former approaches. Microsoft's XLANG described workflow-oriented compositions with *block-oriented* control flow, while IBM's Web Services Flow Language (WSFL) described compositions via graph-oriented control flows. Together they defined the Business Process Execution Language (BPEL), which as WS-BPEL is now an OASIS standard. BPEL allows the definition of a new Web Service which is realised by executing a workflow over existing services. Control flow between the component processes can be defined in a hybrid of the XLANG and WSFL styles; i.e., in block-oriented or graph-oriented style. Dataflow, on the other hand, is achieved solely by assigning the messages resulting from executing service operations to mutable variables, deconstructing these messages using XPath and copying parts to new variables forming messages for the execution of further operations or output from the composite service. Decision points in the control flow are similarly resolved using (boolean valued) XPath expressions over the contents of mutable variables. This approach, therefore, requires a detailed understanding of the physical representation of the message structures, and implicitly relies on the declaration and management of named variables to encode implicit knowledge based on the relationship between data in these variables, and the Web Services from which they were obtained.

[4] `http://www.w3.org/TR/wsdl20/`
[5] `http://www.oasis-open.org/committees/uddi-spec/`

3 Semantic [Web Service] Comprehension and Composition

The approach we describe as 'Semantic [Web Services]' is that foregoing work which has primarily concentrated on applying ontology-based semantics to enhance the stack defined in the previous section to increase comprehension and, to some degree, composition. OWL-S[6] is an ontology for the description of Web Services that has been primarily applied to SOAP services with WSDL descriptions. The OWL-S Service Profile increases on the amount of static information available about a Web Service from that associated with WSDL and UDDI in the form of OWL descriptions, thereby allowing OWL-based reasoning. In particular the taxonomic classification of services that UDDI adds to WSDL is captured via so-called 'service categories', and the input and output messages of service operations – or more precisely, in combination with OWL-S' Service Grounding, and the WSDL 1.1 bindings, the 'parts' of messages – are mapped onto OWL concepts. We note that this is a direct map, which can be supplied with a transformation between the physical representation, for instance in XML, and the semantic representation in OWL. In other words, the concepts attached to parameters express what is communicated by the messages *in themselves*, not what implicit knowledge can be inferred from executing the service, in particular in relationship between the knowledge represented in the input.

The OWL-S Process Model allows a composite Web Service behaviour to be derived from 'atomic processes' by which existing service functionality, primarily SOAP operations, have been encoded. This process model defines control-flow in a strictly block-oriented fashion, but dataflow is specified in a graph-oriented manner, i.e., by connecting parameters of atomic and contained composite processes. This assumes, however, that outputs, once transformed to semantic representations, directly fulfil the required inputs. By extension it means that implicit knowledge, based on the relationships among that which has been derived from physical messages, is extremely difficult to manage. This may be one reason the process model is revised in the proposal for OWL-S 1.2 to include local variables.[7] This is not subject, however, to tool support and we believe this programming-oriented solution is not the best solution to what is fundamentally a knowledge management challenge. In order to express conditions for the decision points in processes, the OWL languages are insufficient and SWRL is recommended, though this non-standard language has limited tool support and adoption potential from the communities composing services.[8]

The Web Services Modeling Ontology (WSMO, [7]) is a similar, but wider-ranging, attempt to extend Web Services via ontology-based modelling and reasoning. 'Grounding' within service models similarly allow a correspondence between physical messages, primarily the WSDL description of SOAP messages, and ontology concepts. 'Lifting' and 'lowering' are the respective terms for the transformation between physical message representations and the semantic form, and the reverse.

Towards further comprehensibility, rather than mixing an ontology language and a separate rule language, as in OWL-S, the WSMO meta-model is implicitly included in

[6] http://www.w3.org/Submission/OWL-S/
[7] http://www.ai.sri.com/daml/services/owl-s/1.2/
[8] http://www.w3.org/Submission/SWRL/

a family of ontology languages, called WSML for Web Service Modeling Language, which includes the ability to define rules. OWL-S aim to allows the definition, using SWRL, of conditions and effects. WSMO, as captured in WSML, makes two refinements to this. First conditions are split into preconditions and (external) assumptions, and effects into postconditions and (external) effects. Secondly, WSML allows the declaration of *shared variables* which can be used to document implicit links between inputs and outputs. With regard to the approach outlined in this paper, however, the issue is that, while potential useful for discovery and automated composition, this implicit knowledge is not communicated as *capabilities*, where these are contained, are wholly separate from *choreographies*, documenting communications and their grounding.

There are two exiting approaches to composition using WSMO. The first is the possibility for a service to be declared with an included executable orchestration. An ontologized form of abstract state machines [8] has been proposed for the definition of such orchestrations [9]. These build on the rule-like logical expressions of the WSML syntax to form implicitly stateful behaviours. The control flow, in terms of propagation of control between states, is thus not explicit, either in block- or graph-oriented form, but implicit in rule-oriented form. The second approach extends BPEL to form BPEL4SWS (BPEL for Semantic Web Services), wherein 'Semantic [Web Services]' are composed in BPEL processes. A semantic form for the dataflow in BPEL4SWS has been considered [10], wherein WSML rules are used to transform between the lifted forms of messages, but this still uses BPEL's mutable variables to disambiguate, and represent implicit knowledge between, these. In both cases a knowledge of WSML, a non-standard language, is needed to form compositions. This situation will be eased through the standardization of the Rule Interchange Format (RIF)[9], which with WSML for syntactic compatibility. Still, the '[Semantic Web] Service' approach presented here offers a more immediately accessible alternative for its intended audience, who is more familiar with linked data-associated technology than rules.

The W3C submissions of both OWL-S and WSMO, while not leading to recommendations of either model, have lead to the definition, and subsequent recommendation, of SAWSDL, in full 'Semantic Annotations for WSDL and XML Schema'.[10] SAWSDL does not define any specific ontological service model at all, but allows 'model references' from parts of the WSDL service schema into arbitrary ontological models. Since, however, model references are shown to explicitly apply to WSDL messages, and 'lifting and lowering schema' references are also provided, it is clear that SAWSDL preserves the simplistic view that the concepts targeted are mere representations of the messages and are not extended, as in our approach, to the implicit knowledge associated with to service execution. One response to the publication of SAWSDL has been to view it as an extra (dual) link to the existing models; another, that of WSMO-Lite [11], has been to produce a reduced ontological model in even closer accordance. Since the 'minimal service model' ontology associated with WSMO-Lite is close to our requirements, we choose to reuse this, even though we are deliberately not specific to SAWSDL.

[9] `http://www.w3.org/2005/rules/`
[10] `http://www.w3.org/TR/sawsdl/`

4 [Semantic Web] Service Composition

As well as the limitations with respect to communicating implicit knowledge associated with service invocation, and the use of non-standard languages for description, the above described approaches fall down in the developing Semantic Web context with respect to two increasingly common types of service. Firstly, there is a significant rejection of the WS-* stack in the RESTful services movement. In REST arbitrary operations are not overloaded (usually over HTTP POST) on single endpoints in SOAP RPC style, rather the HTTP operations (GET, PUT, POST and DELETE) are applied to resources addressed via individual URIs. Secondly, many Linked Data services are simply abstracted to SPARQL endpoints. Here the concern is not with formalising the (generic) messages and (unnecessary) lifting and lowering, but the data that is allowed. Neither are these issues orthogonal since there are an increasing number of RESTful, and indeed also SOAP services, that directly produce and consume RDF.[11]

For all of these reasons we propose a model for the comprehension and composition of [Semantic Web] Services that are viewed primarily as RDF 'prosumers', which may be realised by SOAP or RESTful services or by SPARQL endpoints, which are viewed and combined using SPARQL, and whose composition is open to the inclusion of further Linked Data sources that, at any point in a process, may extend the execution's knowledge base.

4.1 Basic Concepts

As we discussed previously in this paper, Semantic [Web Services] traditionally rely on ontology-based classifications for the typing of input and output concepts, and for the formalization of conditions and effects. Although, the more recently emerging lightweight approaches to service descriptions such as SAWSDL still use conceptual links for the annotations of services. In particular the descriptions of input and output messages are reduced to the linkage to a concept in an external ontology.

A core concept of our composition approach is the fact that [Semantic Web] Services take RDF graphs as 'input message' and produce RDF as output too. This concept is schematically depicted on Figure 1. No matter what type of service is bound to the process, at the level of the [Semantic Web] Service RDF is consumed and produced, and all communication is conducted at the semantic level. The description of the input and output, respectively, is then no longer given by linking some value to some concept in the ontology, but rather by a graph pattern which precisely describes the content of the expected input graph (condition), respectively the guaranteed output graph (effect). A graph pattern that describes the input to the GeoNames 'search' RESTful service[12] is shown in Table 4, as part of the semantic service description of the example process in Section 5.

Having the corresponding RDF data available, online or in a process-specific semantic space (termed *Process Space* on Figure 1), allows for the creation of SPARQL

[11] Witness the recent acceptance as an Apache Incubator project of Clerezza: `http://incubator.apache.org/clerezza/`.

[12] `http://sws.geonames.org/search`

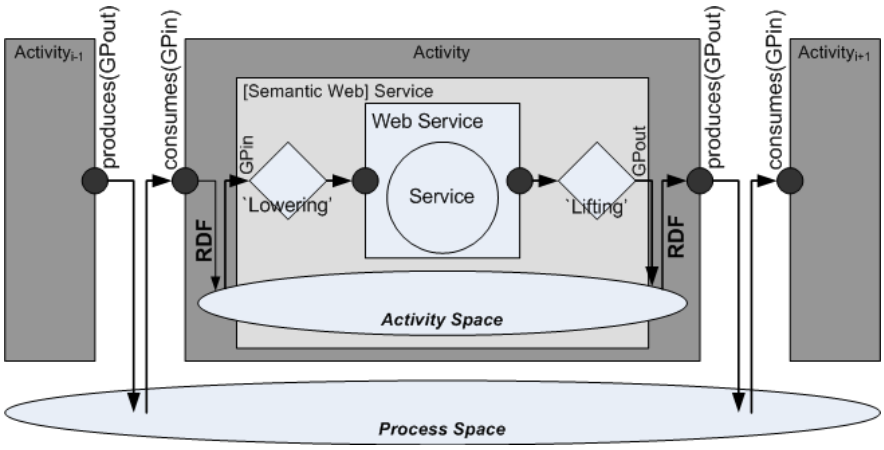

Fig. 1. [Semantic Web] Service composition

CONSTRUCT queries that fetch and prepare the required RDF statements. A semantic space is a semantic-minded evolution of tuplespaces into a blackboard-style communication and coordination platform for the realization of collaborative problem-solving activities [12]. The fundamental principle on which computation over semantic spaces is based, is the fact that participating agents or services do not communicate with each other directly, but instead collaboration is guided by the coordinated access to and the (concurrent) modification of a shared set of RDF statements. In the simplest case, the one depicted in Figure 1, all the required and produced RDF data is shared in a process-owned space against which the SPARQL queries are executed. Generalizing this setting, without altering any of the core concepts, the input data could also be derived by more complex CONSTRUCT queries over public data sources, or by means of Semantic Web Pipes [13] that aggregate data from various sources on the Web or the LOD initiative.

The use of semantic spaces as shared and persistent data management layer has thus several advantages. Processes, i.e. service compositions can be executed in time and reference decoupled manners. Subsequent services do not directly invoke each other, nor do they have to know of their precise existence. Execution is governed by coordinated access to the shared knowledge, and not by invocation. In tuplespaces systems [14], coordination is realized by means of blocking operations that only fire once a required piece of information is available in the shared space. Blackboard system rely on a coordinator agent that passes a token along to all participators [15]. In the scope of semantic spaces, publish-subscribe style notification services that exploit graph patterns as subscriptions are investigated as a means to trigger the invocation of a service [16].

As a direct consequence of this implicit invocation of services within a process, there is no need to ship data around between [Semantic Web] Services. In fact, any service selects precisely the data from the space that is required, and as such, the input to one service is a priori independent of the output of the predecessor. In other words, the execution of a process does not really require the specification of data flow.

The last missing piece to [Semantic Web] Service composition relates to the fact that most Web Services do still not accept or produce RDF data. Although more and more service rely on RDF directly, or REST principles for invocation, XML is still the predominant format for data transfer over the Internet. This is even the case for services that are internally manipulating RDF data, such as for example the GeoNames services that operate over Linked Data. In order to invoke a Web Service, it is thus still necessary to transform from RDF to the expected data format of the service implementation, and to map the output of the service back into RDF. A [SW]S does consequently not only specify in its description what graph patterns it consumes and produces, respectively, but also how the input RDF is 'lowered' to the expected data format of the service, respectively how the output of the service is 'lifted' back to RDF. Again, if services consume RDF, 'lowering' and 'lifting' are obsolete and the input message – RDF data – is directly fed from the activity space to the service. In all other cases the lowering box prepares the appropriate call. This can be done by help of a simple SELECT query that binds values to the query variables in a URL, or by more heavyweight XSLT or XSPARQL transforms for XML-based services;[13] e.g., services expecting SOAP messages. The different types of lowering are illustrated in the example of Section 5.

4.2 Process Representation

The fundamental characteristic of processes that form compositions over '[Semantic Web] Services' is that a knowledge-based view is taken on the behaviour, and knowledge is directly built up within 'spaces' from which it can be projected out to different participants. Each prototypical use made of a [SW]S, including Linked Data endpoints as well as Web Services and RESTful services with lifting and lowering, is represented as an *activity*. Each activity has a space, in which the activity's knowledge (triples) is maintained during its lifetime. Triples from each activity's space are selected and augmented, to represent implicit knowledge; i.e., to infer triples in the process space. Finally triples from the process space can be selected and augmented to make the process available as an [SW]S.

In order to arrange this behaviour processes must represent control flow between activities. Our approach is agnostic to the choice between block- and graph-oriented control flow. It is also supportive of data-driven approaches to control, wherein control may be according to some global scheme[14], and local execution order is derived from 'readiness' judged according to the satisfaction of input conditions given by the state of the current process space.

For the purposes of illustration, however, we present a simple block-oriented representation of processes with explicit control flow. One reason to do this is to demonstrate the use of SPARQL ASK queries to define conditions that affect the process control and to show that this is more expressive than a data-driven 'mash-up' approach to the com-

[13] `http://www.w3.org/Submission/2009/01/`

[14] For instance: all activities are executed once before the process completes; they are divided into two classes where 'sources' are each executed once and others are iterated in an interleaved fashion until no further activity is ready. These data-driven schemes, and several block- and control-oriented process definitions are surveyed in [17]

Table 1. Simple language for processes composing [SW]S (in N3)

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix act: <http://www.example.org/SWServices/Activities#> .
@prefix proc: <http://www.example.org/SWServices/Process#> .

proc:Process rdf:type rdfs:Class .

proc:Sequence rdfs:subClassOf proc:Process .

proc:hasFirst rdf:type rdf:Property ;
              rdfs:domain proc:Sequence ;
              rdfs:range act:Activity .

proc:hasNext rdf:type rdf:Property ;
             rdfs:domain proc:Sequence ;
             rdfs:range proc:Process .

proc:Choice rdfs:subClassOf proc:Process .

proc:hasBranch rdf:type rdf:Property ;
               rdfs:domain proc:Choice ;
               rdfs:range proc:Branch .

proc:ConditionalBranch rdfs:subClassOf proc:Branch .

proc:hasCondition rdf:type rdf:Property ;
                  rdfs:domain proc:ConditionalBranch ;
                  rdfs:range act:sparqlLiteral .

proc:asserts rdf:type rdf:Property ;
             rdfs:domain proc:Branch ;
             rdfs:range act:sparqlLiteral .

proc:chosesFor rdf:type rdf:Property ;
               rdfs:domain proc:Branch ;
               rdfs:range proc:Process .

proc:While rdfs:subClassOf proc:Process .

proc:iterates rdf:type rdf:Property ;
              rdfs:domain proc:While ;
              rdfs:range proc:ConditionalBranch .

proc:consumes rdf:type rdf:Property ;
              rdfs:domain proc:Process ;
              rdfs:range act:sparqlLiteral .

proc:produces rdf:type rdf:Property ;
              rdfs:domain proc:Process ;
              rdfs:range act:sparqlLiteral .

act:sparqlLiteral rdfs:subClassOf rdfs:Literal .
```

position of services as RDF consumers/producers, but still natural to the community to whom we intend to appeal.

Table 2. Activities in [SW]S processes (in N3)

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix wsl: <http://www.wsmo.org/wsmo-lite#> .
@prefix act: <http://www.example.org/SWServices/Activities#> .

act:Activity rdf:type rdfs:Class .

act:performs rdf:type rdf:Property ;
             rdfs:domain act:Activity ;
             rdfs:range wsl:Service .

act:consumes rdf:type rdf:Property ;
             rdfs:domain act:Activity ;
             rdfs:range act:sparqlLiteral .

act:produces rdf:type rdf:Property ;
             rdfs:domain act:Activity ;
             rdfs:range act:sparqlLiteral .
```

Table 1 illustrates an RDFS-based approach to the definition of such processes, using the definition of activities in Table 2. The sequence is the fundamental means of composition as it includes 'singleton' sequences with only an activity and no 'next part'. In general this allows, as do the other process subclasses, composition of processes (and thereby activities). Choices and loops are both based on branches; a choice is expected to have exactly one non-conditional (default) branch and any number of conditional ones, based on SPARQL ASK queries, whereas a loop is expected to have exactly one such conditional branch only. Each branch is allowed to assert further knowledge, via a CONSTRUCT query, into the process space.

Activities, in this and potential other [SW]S process languages, perform a service which will be described using the WSMO-Lite lightweight service model. Before carrying out the lowering, to derive the input message for the invocation, the declared 'consume' CONSTRUCT is carried out over the process space, and any required external Linked Data sources, populating the activity's space. After the invocation the returned message is lifted and added to the local activity space, the 'produce' CONSTRUCT is then carried out over the activity's space in order to derive triples that are injected into the process' space. In this way, implicit knowledge about the links between the input and output, as well as context related to the authority (i.e., the service), can be included in the activity's contribution to the process' knowledge base.[15]

[15] The activity space thus enables the maintenance of state across the execution of a service, which, for example, is modelled in WSMO as 'shared variables' that connect the pre-condition to the effect of a service capability.

5 Evaluation

In order to evaluate our conceptual approach, we have carried out a proof-of-concept implementation. To illustrate the implementation we designed a hypothetical scenario that relies on the composition of three services: a real existing RESTful service that produces RDF data, a real one that returns an XML message and a locally implemented service that relies on SOAP messages. We chose not to use an existing service only to avoid real-life side effects. A formal specification of our evaluation process, according to the schema in Section 4.2, is given in Table 3. The process composes three activities that represent each of the three services.

Table 3. Process formalization

```
@prefix act: <http://www.example.org/SWServices/Activities#> .
@prefix proc: <http://www.example.org/SWServices/Process#> .
@prefix geo: <http://ws.geonames.org/> .
@prefix sparql: <http://www.w3.org/TR/rdf-sparql-query/#> .

_:p1 rdf:type proc:Sequence;
   proc:hasFirst _:a1; proc:hasNext _:x;
   proc:consumes "CONSTRUCT ..."^^sparql:construct .
_:a1 act:performs geo:search;
   act:consumes
     "PREFIX geo: <http://www.geonames.org/ontology#>
      CONSTRUCT {?x geo:name ?o1 .}
      WHERE {?x geo:name ?o1 .}"^^sparql:construct;
   act:performs
     "PREFIX wgs84: <http://www.w3.org/2003/01/geo/wgs84_pos#>
      CONSTRUCT {?x wgs84:lat ?lat;
                    wgs84:long ?lng .}
      WHERE  {?x wgs84:lat ?lat;
                 wgs84:long ?lng .}"^^sparql:construct .
_:x proc:hasFirst _:a2; proc:hasNext _:ch .
_:a2 act:performs geo:findNearByWeatherXML;
   act:consumes "CONSTRUCT ..."^^sparql:construct;
   act:performs "CONSTRUCT ..."^^sparql:construct .
_:ch proc:hasBranch _:cbr, _:br .
_:cbr proc:hasCondition "ASK ..."^^sparql:ask;
   proc:chosesFor _:p2;
   proc:asserts "CONSTRUCT ..."^^sparql:construct .
_:br proc:chosesFor _:p2;
   proc:asserts "CONSTRUCT ..."^^sparql:construct .
_:p2 proc:hasFirst _:a3 .
_:a3 act:performs <http://localhost:8080/axis/YP.jws>;
   act:consumes "CONSTRUCT ..."^^sparql:construct;
   act:performs "CONSTRUCT ..."^^sparql:construct .
```

The first service is from GeoNames. It takes the name of some geographic feature, and a return type as query variables: `http://ws.geonames.org/search?type=rdf&name=Innsbruck`. By putting the type on RDF, the service returns RDF/XML, rather than some proprietary XML document, that can be written directly back into the activity space without the need for lifting. One possible semantic description of this service is shown in Table 4. There are references given to the condition on the input of the service, the effect of the output, and a link to a lowering schema mapping; there is not need for a lifting schema, as the service natively returns RDF. In our SPARQL-minded approach, the condition and effect of a service are given by graph patterns, as they are defined by the SPARQL specification.[16] In fact, the graph patterns that are part of the service description relate directly to the consumes and produces part, respectively, of the activity description – compare, for example, the value of the condition in Table 4 with the consumes statement of the first activity in Table 3.

The lowering for the 'search' service is very simple in the given case. All that is needed is the name of the location to *search* which becomes part of the service URL as query string. Hence, the sole variable binding of the SPARQL SELECT query "`SELECT ?name WHERE {?x geo:name ?name .}`" becomes the ?name variable of the service URL. The SELECT query is executed against the activity space which is populated by the aforementioned CONSTRUCT query.

Table 4. Lightweight service description for GeoNames' search service

```
@prefix sparql: <http://www.w3.org/TR/rdf-sparql-query/#> .
@prefix wsl: <http://www.wsmo.org/wsmo-lite#> .
@prefix geo: <http://ws.geonames.org/> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .

geo:search rdf:type wsl:Service;
    sawsdl:modelReference _:cond, _:effect;
    sawsdl:loweringSchemaMapping
                  <http://localhost:8080/search.sparql> .
_:cond rdf:type wsl:Condition;
    rdf:value "?x geo:name ?name ."^^sparql:GraphPattern .
_:effect rdf:type wsl:Effect;
    rdf:value "?x wgs84:lat ?lat;
                  wgs84:long ?lng ."^^sparql:GraphPattern .
```

The second service is GeoName's 'findNearByWeatherXML' service.[17] It takes latitude and longitude coordinates as input and returns the observations of the nearest weather station as an XML document, including temperature, humidity, clouds, atmospheric pressure, and wind direction and speed amongst other things. The service description is similar to the one presented in Table 4, and we spare it here. The same

[16] `http://www.w3.org/TR/rdf-sparql-query/`
[17] `http://ws.geonames.org/findNearByWeatherXML`

accounts for the lifting and lowering, which is more interestingly presented in the context of the SOAP-based service (Table 6).

The third service that is invoked depends on the weather information returned by GeoName's weather service. For this reason, the process contains a 'choice' construct over the knowledge in the process space that was asserted by the preceding activities. The choice is implemented by means of an ASK query that is exemplified in Table 5 by means of a temperature threshold.[18] Consequently, either the SOAP-based yellow pages service is invoked to query the phone number of a local restaurant, or under better weather conditions the contacts of some tennis club in the chosen city. As both branches of the choice trigger the same activity that wraps the service `http://localhost:8080/axis/YP.jws`, the difference in execution is given by the asserted triples of the branches. As stated above, the conditioned branch causes the request for a restaurant's contact information, and hence the type of business to look for that has to be communicated to the yellow pages service is "Restaurant", otherwise "Tennis".

Table 5. The 'choice' construct: ASK condition and CONSTRUCT-based assertion

```
PREFIX wth: <http://www.example.org/weather#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

ASK {?x wth:temperature ?t .
  FILTER (?t < "10"^^xsd:integer) .}
.....................................................................
PREFIX yp: <http://www.example.org/yellowpages#>

CONSTRUCT {_:yp1 yp:loc "Innsbruck";
                 yp:type "Restaurant" .}
WHERE {}
```

The triples which are asserted by the choice construct match the condition graph pattern of the yellow pages service: "`{?x yp:loc ?l; yp:type ?t.}`". The choice thus ensures that the input data which is required by the last activity of our composition is available in the process space, and can be moved into the activity space. From there the activity's consumes construct is invoked and the semantic data is lowered to the required SOAP message (Table 6), which is sent to the yellow pages service. The response XML is analogously lifted back to RDF and written back to the process space as final output of the process. In our example a graph is constructed that describes a restaurant or tennis club entry with a name, phone number and address.

This concludes our proof-of-concept implementation. Out of a set of lightweight semantic service descriptions with conditions and effects in form of graph patterns, and a formal specification of a process, we construct a composition that is entirely based on well-established (Semantic) Web technologies: RDF, SPARQL and XSLT/XSPARQL for lifting and lowering, if required.

[18] The predicate 'temperature' matches the predicate of the lifted output of the weather service.

Table 6. Lowering schema for the yellow pages service

```
<lowering>
  <sparqlQuery>
    PREFIX yp: <http://www.example.org/yellowpages#>
    SELECT ?loc ?type
    WHERE { ?x yp:loc ?loc; yp:type ?type .}
  </sparqlQuery>
  <xsl:transform version="2.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:yp="http://www.example.org/yellowpages#">
    <xsl:output method="xml" version="1.0" ... />
    <xsl:template match="/sparql">
      <yp:TelephoneNumberRequest>
      <xsl:for-each select="results/result">
        <yp:loc>
        <xsl:value-of select="binding[@name='loc']/literal"/>
        </yp:loc>
        <yp:type>
        <xsl:value-of select="binding[@name='type']/literal"/>
        </yp:type>
      </xsl:for-each>
      </yp:TelephoneNumberRequest>
    </xsl:template>
  </xsl:transform>
</lowering>
```

6 Conclusion

Although we have not yet been able to evaluate our approach in scale to make sure that SPARQL-minded [Semantic Web] Service composition is efficient, we are very confident that it matches up to the needs of more complex services and process too. Experience with state-of-the-art semantic repositories has shown that they are performing enough to realise semantic spaces for compositions that rely on several millions of triples.

Recapulating the paper, we have presented a novel approach to [Semantic Web] Service comprehension and composition that leverages established semantic technologies such as RDF(S) or SPARQL to create processes and to determine the control- and data-flow, respectively. We intend to establish a composition approach that is more intuitive and more appealing to the linked data community. Indeed, instead of logics-heavy capability descriptions, as proposed by various Semantic [Web Service] framework, we propose to provide conditions and effects in form of graph patterns, which are directly applied for the construction of the input and output (RDF) messages of the involved services. In that way, the services emerge from the Semantic Web, remain in the linked data world and, after execution, explicitly feed RDF data back to the Semantic Web.

Future work includes the realization of more complex evaluation scenarios, and the inauguration and evaluation of the approach in the context of knowledge-intensive processes on top of private linked data and (public) Linked Open Data.

Acknowledgement: The work is supported by the EU FP7 IP SOA4All, and the infrastructures project Seals. We thank our colleagues from these projects for their valuable discussions and related work, namely Jacek Kopecky and Carlos Pedrinaci.

References

1. Cardoso, J., Sheth, A.: Semantic Web Services, Processes and Applications. Springer (August 2006)
2. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services: The Web Service Modeling Ontology. Springer (November 2006)
3. Studer, R., Grimm, S., A. Abecker (eds.): Semantic Web Services: Concepts, Technologies, and Applications. Springer (June 2007)
4. Fielding, R.: Architectural Styles and the Design of Network-based Software Architectures. PhD thesis, University of California, Irvine (2000)
5. Richardson, L., Ruby, S.: RESTful Web Services. O'Reilly Media (May 2007)
6. Kopecky, J., Roman, D., Moran, M., Fensel, D.: Semantic Web Services Grounding. In: Int'l Conference on Internet and Web Applications and Services. (February 2006)
7. Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., Fensel, D.: Web Service Modeling Ontology. Applied Ontology **1**(1) (2005) 77–106
8. Börger, E., Stärk, R.: Abstract State Machines. Springer (2003)
9. Fensel, D., Lausen, H., Polleres, A., de Bruijn, J., Stollberg, M., Roman, D., Domingue, J.: Enabling Semantic Web Services. Springer (2006)
10. Nitzsche, J., Norton, B.: Ontology-based data mediation in BPEL (for semantic web services). In: Business Process Management Workshops. LNCS, Springer (2008) 523–534
11. Vitvar, T., Kopecky, J., Viskova, J., Fensel, D.: WSMO-Lite Annotations for Web Services. In: 5th European Semantic Web Conferences. (June 2008) 674–689
12. Nixon, L., Simperl, E., Krummenacher, R., Martin-Recuerda, F.: Tuplespace-based computing for the Semantic Web: A survey of the state of the art. Knowledge Engineering Review **23**(1) (March 2008) 181–212
13. Le-Phuoc, D., Polleres, A., Morbidoni, C., Hauswirth, M., Tummarello, G.: Rapid Prototyping of Semantic Mash-Ups through Semantic Web Pipes. In: 18th World Wide Web Conference. (April 2009) 581–590
14. Gelernter, D.: Generative Communication in Linda. ACM Transactions on Programming Languages and Systems **7**(1) (January 1985) 80–112
15. Engelmore, R.: Blackboard Systems. Addison-Wesley Publishers (November 1988)
16. Krummenacher, R., Norton, B., Simperl, E., Pedrinaci, C.: SOA4All: Enabling Web-scale Service Economies. In: 3rd Int'l Conference on Semantic Computing. (September 2009)
17. Norton, B.: A Theory for Flow-Oriented Software Processes. PhD thesis, Department of Computer Science, University of Sheffield (2009)
18. Rohloff, K., Dean, M., Emmons, I., Ryder, D., Sumner, J.: An Evaluation of Triple-Store Technologies for Large Data Stores. In: OTM Confederated Int'l Workshops and Posters. (2007) 1105–1114
19. Ontotext AD: Robust Engines for Analysis and Management of Text and Data. Ontotext Brochure (2009)