

# SOA4All: Towards a Global Service Delivery Platform

Reto KRUMMENACHER<sup>a,1</sup>, John DOMINGUE<sup>b</sup>, Carlos PEDRINACI<sup>b</sup> and Elena SIMPERL<sup>a</sup>

<sup>a</sup>*Semantic Technology Institute, University of Innsbruck, Austria*

<sup>b</sup>*Knowledge Media Institute, The Open University, Milton Keynes, UK*

**Abstract.** Establishing Web services as resources on the Web opens up productive but challenging new possibilities for open, highly dynamic and loosely-coupled service economies. In addition, lifting services to the semantic level provides a sophisticated means for automating the main service-related management processes and the composition of arbitrary functionalities into new services and businesses. In this article we present the SOA4All approach to a global service delivery platform. By means of semantic technologies, SOA4All facilitates the creation of service infrastructures and increases the interoperability between large numbers of distributed and heterogeneous functionalities on the Web.

**Keywords.** Service-Oriented Architecture, Semantic Web services, Global Service Delivery Platform

## Introduction

The Web service technology stack offers various means for making software functionality accessible as remote components, independent of particular programming languages and platform implementations. Significant work was done in specifying architectures, middleware, languages, communication protocols and process execution engines that can support the creation of complex distributed systems by seamlessly coordinating Web services. Service-Oriented Architectures (SOAs) foster the development of such distributed and loosely-coupled solutions whereby service providers advertise the services they offer, and solution providers and software developers access the service repositories to search for suitable services to invoke for the given purpose or to build and execute processes.

Within the SOA4All project, the core ideas of SOA are re-thought with the aim of making services ubiquitous on the Web. The chosen approach is to combine the principles which underpin the Web, Web 2.0, semantics, context and SOA and to derive an architecture based on these principles. In particular, from the Web we take openness, decentralization, and the fact that communication is driven by a ‘persistent publish and read’ paradigm rather than by messaging. In SOA4All, Semantic Web languages are leveraged to increase the automation of various common tasks during the life-cycle of services, such as their discovery and composition. From Web2.0 we take the value of easy-to-use interfaces and of social networks. Finally, automated context

---

<sup>1</sup> Corresponding Author.

adaptation capabilities are embedded within the architecture in order to support the use of services in unforeseen contexts. In provisioning a Web where services exist in billions, we argue that the SOA4All architecture provides a Web-based example of a global service delivery platform. In particular, by empowering Web services as resources on the Web, SOA4All yields the fundamental building blocks for the creation of new business opportunities in form of open and loosely-coupled service economies.

SOA4All addresses the Web of Services in which millions of users work with billions of services. In order to manage the resulting wealth of resources – e.g., service descriptions, user profiles or process models – there is a significant need for automation. Without automation it would neither be feasible to scale to the dimensions of the Web, nor to offer the service delivery platform functionality. Automation relies on metadata to create more abstracted views onto real objects. Semantic technologies are a key component for such work and were successfully applied in many recent projects on software and services to lift services and their descriptions to a level of abstraction that deals with machine-understandable conceptualizations, and that decreases the dependency upon human users. Thanks to semantics, it is possible to automate the services' life-cycle management.

In this chapter we first present the conceptual architecture of the SOA4All service delivery platform. Section 2 presents the integration middleware in form of federations of distributed services buses. In Section 3, the platform services are introduced; platform services provide the basic components of a service delivery platform, such as the discovery and composition services or the reasoning engines. Section 4 depicts a consolidated example of how the service delivery platform integrates in order to establish an executable process. To conclude the chapter, we provide a wrap-up and depict some future directions.

## **1. SOA4All Conceptual Architecture**

A global service delivery platform (GSDP) is an open platform through which domain independent services can be used to build problem-specific service solutions. SOA4All establishes a service delivery platform that is targeting Web services (traditional WS-\* stack-based and RESTful services, as well as Web APIs). Future implementations of a GSDP will have to consider other exposable functionalities too, such as mobile services or sensors networks in order to fully enable the 'Everything as a Service' paradigm.

The SOA4All platform focuses on automating the management of services that are traditionally driven by technologies such as WSDL and SOAP, and on empowering RESTful services. Automation is advocated through the application of semantics and hence by means of Semantic Web services. In other words, services are annotated by means of Semantic Web languages, and the platform services operate on the semantic descriptions of services and processes, rather than the actual software implementation or the physical endpoints. In fact, the services turn into utilities, which disappear in the Web that becomes the platform and a public, open and distributed alternative to private legacy systems. The service capabilities and the offered quality of service become the decisive characteristics, rather than the endpoint location or provider.

In the following, we present the SOA4All conceptual architecture that is grounded in four main building blocks: the Distributed Service Bus in the very centre of Figure 1; the SOA4All Studio, as the user front-end, the platform services at the very bottom, and so-called business services, as well as processes and their descriptions.

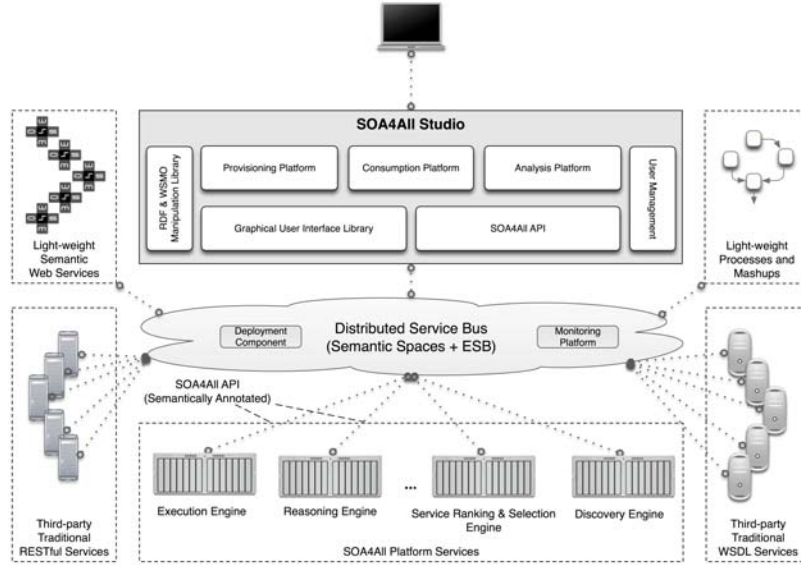


Figure 1. SOA4All architecture.

### 1.1. Distributed Service Bus

The Distributed Service Bus enables Web-style communication and collaboration via semantic spaces and service bus technology, and yields the core runtime infrastructure. The DSB augments enterprise service bus technology with distributed service registries, the layering of the service bus on top of established Internet-enabled middleware, and the enhancement of the communication and coordination protocols by means of semantic spaces. Spaces are seen to significantly increase the scalability of the bus in terms of interaction between distributed and autonomous services [1]. A more detailed description of the DSB and semantic spaces is given in Section 2.

### 1.2. SOA4All Studio

The SOA4All Studio is a Web-based user front-end that consists of three components which offer service provisioning at design time, consumption and analysis of services at runtime, respectively:

The **Provisioning Platform** has two main purposes: i) tools to semantically annotate services; and ii) a process editor that allows users to create, share, and annotate executable process models based on a light-weight process modeling language. Service annotations are based on WSMO-Lite [2], a minimal extension to SA-WSDL [3] that empowers the creation of lightweight semantic service descriptions in RDFS. In parallel, MicroWSMO [4] is used to annotate services that are not described in WSDL, such as RESTful services or Web APIs. MicroWSMO is a microformat-based language around the constructs known from WSMO-Lite, however, adapted to support the annotation of HTML-based descriptions, as they are usually available for this type of software exposures. Finally, SOA4All provides a minimal service model in RDFS that yields an overarching conceptual model able to capture the semantics for both Web

services and Web APIs, thus allowing both kinds of services to be treated homogeneously within SOA4All.

The **Consumption Platform** is the gateway for service consumers. It allows users to formalize goals. A goal is a formal specification of an objective and as such yields an implicit specification of the services that need to be executed. User objectives are transformed into processes that are compositions of service descriptions and so-called service templates together with control and data flow information and potentially further constraints on the services and their execution. Service templates define process-internal activities instead of concrete services whenever flexibility in service selection is desired. At runtime, service templates are resolved to specific services that are selected on the basis of conditions and informed by contextual knowledge which may include monitoring data, user location or other aspects that affect the appropriateness of a service endpoint.

The **Analysis Platform** collects and processes monitoring events from the service bus, extracts and produces meaningful information out of it and displays the results to users. Monitoring events come from data collectors that perform basic aggregation from distributed sources in the service delivery platform. Data collectors are installed at the level of the bus and the execution engine, and are installed to cover all aspects of monitoring necessary in the context of SOA4All: monitoring of process executions, of service end-points that are invoked through the service bus, and finally of the infrastructure itself. While users can select particular services to be monitored in terms of quality of service attributes, simpler and less comprehensive data can also be collected for all other services that are empowered through the bus, but that are not explicitly monitored upon user request; e.g., the moving average for response time.

### *1.3. Platform Services*

Platform services provide the minimally necessary functionality of a service delivery platform, such as service discovery, ranking and selection, composition and invocation. These components are offered as Web services via the service bus and are consumable in the same manner as any other published business service. Although distinct in their purpose, they have in common that they operate with semantic descriptions of services, service templates and processes rather than with the syntactical representation of those, as it is traditionally the case in service-oriented infrastructures. The SOA4All platform services, shown at the bottom of Figure 1, are detailed in Section 3.

### *1.4. Business (Web) Services and Processes*

Figure 1 was discussed so far with respect to the central components of the SOA4All service delivery platform – the ensemble of DSB, SOA4All Studio and platform services. Jointly, they deliver a fully Web-based service experience: global service delivery at the level of the bus, Web-style service access via studio, and automated service processing and management via platform services. Moving to the corners of Figure 1, we enter the domain of semantic service descriptions and processes: (1) represents the semantic services descriptions, either in form of annotated RESTful services (3) or WSDL endpoints (4); (1) thus represents the so-called Semantic Web services. The semantic descriptions are used for reasoning with service capabilities (functionality), interfaces and non-functional properties, as well as contextual data.

In the right top corner of Figure 1, (2) represents processes and mash-ups. Processes are orderings of Semantic Web services, service templates with associated constraints, data and control flow. A mash-up is a data-centric model of a composition that is almost entirely executable by coordinated access to data in a semantic space. Although being comparably simple, mash-ups provide a promising approach to Web-style service computing, a pre-requisite for light-weight service economies.

## 2. A Federation of Distributed Service Buses

Earlier in this chapter we stated that the decisive factor in choosing a service is no longer the endpoint but the functionality and quality of service, and that alternative implementations might complement or replace services in a process. Consequently, distributed functionalities are leveraged by communities and no longer by dedicated individuals. In such scenarios, average users become prosumers of services, leading to thousands of new services and business profiles created almost on-the-fly.

Services shared by a community are being composed in the associated marketplace that the actors delimit. Partnerships arise as coalitions, translating directly in the need to interconnect marketplaces into peered and more hierarchically structured federations, thus yielding a unique, however, not flat global service economy. The SOA4All service bus embodies the marketplace for the offering of services at Internet-scale, and a middleware that scales to the dimensions of millions of users and billions of services.

Although ESBs deliver the core functionality for service computing, they are generally restricted to corporate settings, and do not suffice in terms of dynamics, openness and scalability. In order to dynamically scale, SOA4All enhances ESB solutions with state-of-the-art technologies in distribution and semantics and incorporates storage, monitoring, and communication for a virtually global Internet-scale ecosystem. An important step in establishing federations over distributed buses is the provisioning of message routing that does not require point-to-point connections between bus nodes. SOA4All leverages a multi-level, hierarchical organization of bus nodes [5]. The bus relies on a one level hierarchy, as federations are created between distributed service buses of different corporations with the first level being the enterprise level, and the second one the inter-enterprise level given by the Internet.

As stated previously, the DSB is further enhanced with semantic spaces that are motivated by the ‘persistent publish and read’ paradigm of the Web. This allows SOA4All to provide additional communication patterns – such as publish-subscribe, event-driven, and semantic matching – that further decouple the communicating entities in terms of time, processing flow and data schema. Semantic spaces have gained momentum in the middleware community, as a response to the challenges of data sharing and service coordination in Web environments. Semantic spaces fuse tuple space computing from parallel processing; publish-subscribe-style notification services and semantics to create a distributed data management platform [1]. The spaces provide operations for publishing RDF data, querying SPARQL end-points, and coordination via graph pattern-based notification services. Semantic spaces enable the creation of virtual containers in form of subspaces or federations. The latter are temporary views over multiple virtual containers, comparable to read-only views in relational databases. Subspaces and federations are used to create dedicated interaction channels which increase scalability by grouping collaborating services and related data.

The semantic spaces exploit established P2P overlays and the same Internet-scale communication platform as the DSB, which fosters co-existence of space and bus nodes (Figure 2), to ensure scalability. Spaces are indexed in a Chord ring [6] and each node of the ring maintains references to peers of a CAN overlay [7] in which the triples of the spaces are indexed and distributed. A 3-dimensional CAN overlay yields a natural solution to the storing of RDF triples; the axes represent the subjects, predicates and objects of RDF statements and values are ordered lexicographically.

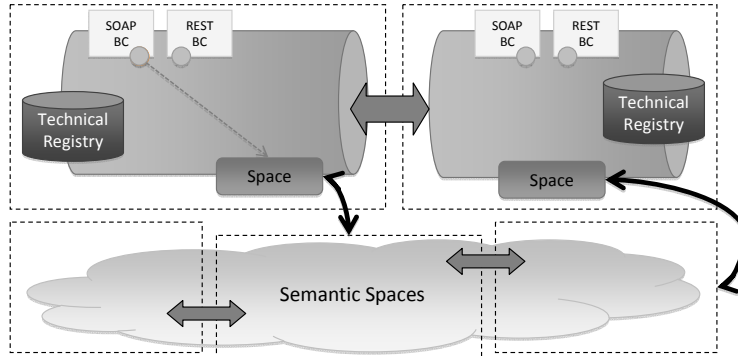


Figure 2. Federation of DSB and semantic spaces.

### 3. Delivering Services: The Platform Services

In the previous section we have presented the SOA4All runtime infrastructure, and turn now to the platform services. A first category – service location – provides users with the possibility to find and rank services according to their needs and context.

**Crawler:** The crawler service is fundamental for service discovery [8]. It collects technical descriptions associated with services from the Web and manages this data for enabling efficient and intelligent retrieval of service-related artifacts. The collected data includes files such as Web site, documentations, pricing and licensing information, and can be delivered either as RDF metadata, or as consolidated non-RDF archive file.

**Service Repository:** The service repository – in SOA4All termed iServe – is used to store and maintain semantic service descriptions, and provides processing capabilities for SA-WSDL and MicroWSMO annotations. The repository offers a RESTful API for users to browse, query and upload semantic annotations and have them automatically exposed as RDF [9]. Through iServe service annotations become part of the Linked Data cloud ([www.linkeddata.org](http://www.linkeddata.org)). The alignment with the Linked Data initiative increases public awareness and brings the annotations into context.

**Discovery:** Service discovery matches user needs to service descriptions. SOA4All uses service templates that abstractly describe a user's objective as a set of RDF triples. Service templates are easily mappable into SPARQL queries to be resolved by iServe. The service template schema is shown in Table 1: the `hasFunctionalCategory` property shadows the functional classification reference that types services according to WSMO-Lite; the input and output properties are used to specify information about the input and the expected output of a service; requirements and preferences are properties that link further constraints. These values match roughly the pre-conditions and effects in WSMO-Lite and are generally more complex than RDF only; e.g., WSML [10].

The discovery service currently offers two types of search: i) full text-based discovery that mostly exploits keyword matching over service descriptions and related documents from the crawler; ii) semantic discovery that leverages service templates. In the simplest case, discovery is reduced to the resolution of the derived SPARQL query. For more sophisticated searches, for example when using requirements and preferences, the discovery service makes use of reasoning.

**Table 1.** RDF Schema for service templates

---

ServiceTemplate	rdf:type	rdfs:Class	.
hasFunctionalCategory	rdf:type	rdf:Property	.
hasInput	rdf:type	rdf:Property	.
hasOutput	rdf:type	rdf:Property	.
hasPreference	rdf:type	rdf:Property	.
hasRequirement	rdf:type	rdf:Property	.

---

**Ranking and Selection:** This service provides means to rank services according to user preferences on non-functional properties of services [11]. It integrates different ranking approaches that exploit the data gathered through crawling and monitoring. A first method uses non-functional properties of services that are given by means of logical rules. Reasoning is applied to compute ranking scores for services based on aggregated non-functional values and their matching degree to user requirements. A second approach is a fuzzy logic-based mechanism that considers a model of preferences that includes vagueness information.

The second category provides the functionality to compose services and to execute the compositions. SOA4All focuses on empowering non-technical users in constructing services, and bases its work on languages that foster re-usability and flexibility in design. The language used in SOA4All is called Lightweight Process Modeling Language (LPML, [12]) – the term lightweight explicitly refers to the usability of the language. LPML is a combination of process modeling concepts found in BPEL, and SOA4All-specific extensions. LPML simplifies BPEL by only considering relevant subsets and extends the existing languages with means to interleave service templates. Furthermore, LPML defines how semantic annotations can be attached to process elements such as activities, goals, input and output parameters, and introduces the concept of data connectors that specify the data flow in service compositions.

**Design-Time Composer:** The design-time composer provides semi-automatic assistance in resolving unbound activities within a process specification. As such, the service is mainly supporting the activities of the studio's process editor. In other words, the design-time composer supports the entire life-cycle of service composition, from supporting the process specification through elaboration of process and template expansions to the discovery and binding of service endpoints as activities within the processes. This includes data mediation and resolution of compatibility problems at the level of service inputs and outputs via service connectors and semantic link operators.

**Composition Optimizer:** The input to the composition optimizer is a complete service composition (as provided through the design-time composer) for which an optimized and executable process specification is sought [13]. Although the composer helps in binding service endpoints, there remain aspects in a process specification that cannot be treated at design-time. A task of the composition optimizer is thus to bind remaining service templates to relevant services. The composition optimizer uses the reasoner when necessary, and considers an extensible quality criteria model by coupling non-functional properties and the semantic descriptions of the process.

**Execution Engine:** The execution engine offers operations to deploy executable processes in an execution environment and to expose processes as invocable Web services. Furthermore, the execution service provides tools to transform light-weight process descriptions into standardized process modeling notations, such as for example BPEL. In SOA4All, the execution engine includes a set of basic mechanisms for adaptive run-time reconfiguration of execution plans in order to react to changes in the execution environment.

To conclude this section we present an additional platform services that offers reasoning support for most of the previously named platform services.

**Reasoner:** The reasoning service exposes a framework of robust and scalable reasoning components that are tailored for each of the WSMML language variants [10]. A variety of interfaces allow for schema/instance reasoning, satisfiability/entailment checking and query answering. The reasoning service has configurable links to service repositories to load service descriptions, and to semantic spaces to access domain ontologies that are required to conclude the desired reasoning tasks.

#### 4. Service Delivery Example

Service delivery in the context of SOA4All focuses on the two areas service location and service construction; although the latter at least partly subsumes the former in order to bind service endpoints to service templates. In this section we exemplify how the different platform services coordinate in order to jointly realize the global service delivery platform.

As a starting point, for our example, we assume that there are semantic service descriptions provided and stored in iServe. The service descriptions were likely created via provisioning tools such as SWEET [14] or SOWER of the SOA4All Studio, which are used for annotating Web APIs or WSDL files, respectively. In order to find the service endpoints and the corresponding documentation, a SOA4All user can profit from the crawler that provides background knowledge to formalize service annotations. In principle, it would also be possible that the crawler detects not only service endpoints, but semantic descriptions, which then, without manual intervention, can be stored in the repository. Our assumed starting point thus covers most of the actions for which the provisioning platform is developed: service endpoint selection, semantic description creation, annotation management and storage in the service repository.

In terms of functional processes that are empowered by SOA4All, the actions triggered through the consumption platform are much more informative. A core component of the consumption platform is the aforementioned process editor that provides a graphical user interface for the modeling of processes. Service compositions are expressed in terms of LPML, which is understood and further manipulated by all service construction-related platform services. An example process with two activities – and without data flow for clarity – is shown in Table 2; note that the example shows the LPML model after the invocation of the design-time composer, and hence a potential Web service is already bound to each of the two activities. The first goal of the process is to determine the latitude and longitude of a city in order to retrieve information on the wind speed from the nearest weather station in a second step. Both services are offered by ws.geonames.org. Data mediation is not necessary in this case since the output of the first activity maps directly onto the input of the second one.



**Table 2.** LPML process description in XML

---

```

<org.soa4all.lpml.impl.ProcessImpl>
...
  <activity class="org.soa4all.lpml.impl.ActivityImpl">
    <operation>getGeoLocationByName</operation>
    <conversation class="org.soa4all.lpml.impl.ConversationImpl">
      <goal class="org.soa4all.lpml.impl.GoalImpl">
        <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
          <referenceURI>http://www.example.org/geolocation#location</referenceURI>
          <type>FUNCTIONAL_CLASSIFICATION</type>
        </org.soa4all.lpml.impl.SemanticAnnotationImpl> </semanticAnnotations>
      </goal>
      <services>
        <serviceReference>http://ws.geonames.org/search</serviceReference>
      </services>
    </conversation>
    <inputParameters>
      <org.soa4all.lpml.impl.ParameterImpl>
        <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
          <referenceURI>http://www.example.org/geo#locationString</referenceURI>
          <type>META_DATA</type>
        </org.soa4all.lpml.impl.SemanticAnnotationImpl> </semanticAnnotations>
      </org.soa4all.lpml.impl.ParameterImpl>
    </inputParameters>
    <outputParameters>
      <org.soa4all.lpml.impl.ParameterImpl>
        <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
          <referenceURI>http://www.w3.org/2003/01/geo/wgs84_pos#long</referenceURI>
          <type>META_DATA</type>
        </org.soa4all.lpml.impl.SemanticAnnotationImpl> </semanticAnnotations>
      </org.soa4all.lpml.impl.ParameterImpl>
      <org.soa4all.lpml.impl.ParameterImpl>
        <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
          <referenceURI>http://www.w3.org/2003/01/geo/wgs84_pos#lat</referenceURI>
          ...
        </org.soa4all.lpml.impl.SemanticAnnotationImpl>
      </org.soa4all.lpml.impl.ParameterImpl>
    </outputParameters>
  </activity>
  <activity class="org.soa4all.lpml.impl.ActivityImpl">
    <operation>getWindSpeed</operation>
    <conversation class="org.soa4all.lpml.impl.ConversationImpl">
      <goal class="org.soa4all.lpml.impl.GoalImpl">
        <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
          <referenceURI>http://www.example.org/weather#WindSpeed</referenceURI>
          ...
        </org.soa4all.lpml.impl.SemanticAnnotationImpl>
      </goal>
      <services>
        <serviceReference>http://ws.geonames.org/findNearByWeatherXML</serviceReference>
      </services>
    </conversation>
    <inputParameters> ... </inputParameters>
    <outputParameters>
      <org.soa4all.lpml.impl.ParameterImpl>
        <semanticAnnotations> <org.soa4all.lpml.impl.SemanticAnnotationImpl>
          <referenceURI>http://www.geonames.org/ontology#windSpeed</referenceURI>
          ...
        </org.soa4all.lpml.impl.SemanticAnnotationImpl>
      </org.soa4all.lpml.impl.ParameterImpl>
    </outputParameters>
    </destination>
  ...

```

---

The first platform service involved in our example is the design-time composer. It is iteratively invoked during the process specification to assist in binding services, expanding templates, checking I/O compatibilities and creating data flow via connectors. Although matching user requirements and process-specific constraints, the outcome of this interactive editing task does not necessarily yield an optimized model. In fact, the design-time composer works mainly on local solutions only and does not deal with the global optimization of a process specification. Therefore it is necessary to work with the completed compositions for performance optimization, context adaptation or for honoring specific user preferences. The composition optimizer accepts complete process models for which it seeks a better global cost function in terms of functional (including semantic similarity of inputs and outputs) and non-functional properties such as the Quality of Service (QoS). The optimizer uses Genetic Algorithms to transform compositions into their optimal versions by replacing service bindings and modifying the data flow without changing the control flow. The transformations are influenced by the context in which a service composition is to be used, and require reasoning support and service discovery for finding concrete and optimized service bindings.

Once a complete model satisfies a user's preferences and requirements in terms of functionality and performance, the process is deployed in the execution engine and exposed as a Web service. As described in Section 3, the execution engine transforms the complete LPML model into a BPEL model enhanced with some SOA4All-specific extensions. Additionally, a corresponding WSDL endpoint is specified, and the execution engine offers one more public service for any deployed process, in addition to the default 'deployProcess' service that the engine hosts.

As the first part of our example has shown, discovery is an essential sub-task of the service construction process. The definition of activity, and in particular the goal element within the LPML process specification can be mapped onto the properties of the service templates (Table 1) which are at the basis of semantic discovery. The annotations of type functional classification that are used to annotate a goal element are transferred to values of the hasFunctionalCategory property. The operations, given through inputOperation and outputOperation in LPML are converted to hasInput and hasOutput respectively. Last, there are other semantic annotations that can be attached to activities, which are of type requirement and non-functional property. These map to hasRequirement and hasPreference accordingly. Table 3 shows a concrete service template that was directly derived from mapping the LPML goal element in Table 2 onto the RDF schema for service templates given in Table 1. The 'st' prefix stands for the service template namespace pointing to <http://cms-wg.sti2.org/ns/service-template#>.

**Table 3.** Concrete service template for the wind speed service

---

```

windSpeedService rdf:type st:ServiceTemplate ;
st:hasFunctionalCategory <http://www.example.org/weather#WindSpeed> ;
st:hasInput rdf:type <http://www.w3.org/2003/01/geo/wgs84_pos#lat> ;
st:hasInput rdf:type <http://www.w3.org/2003/01/geo/wgs84_pos#long> .

```

---

SPARQL queries can be derived from the service templates that can be executed against the semantic service descriptions in iServe. A possible SPARQL query for the template in Table 3 is depicted in Table 4. Note that the classes and predicates are no longer taken from the service template schema but from the minimal service model (prefixed with msm) which borrows some elements from SA-WSDL, identified with the prefix sawsdl. The query selects services which match exactly the functional

category that is searched, and that offer operations with the given input types. Had our service template contained hasOutput specifications, those would have appeared in the query in a similar way to the input types. More sophisticated query specifications that are investigated in the context of service location consider, in addition to the exact match achieved with the example in Table 4, plug-in, subsumes, and fail match degrees common in the literature. One approach to do this is to use subclass relations in the SPARQL query; i.e., the service classification does not reference wind speed directly but rather any subclass of the concept. Executing the SPARQL query against the repository results in a collection of service endpoints that match the functional classification, and the input and output parameters respectively.

**Table 4.** SPARQL query to be executed against the service repository

---

```

SELECT ?service ?operation ?endpoint
WHERE {
  ?service rdf:type msm:Service ;
           rdfs:isDefinedBy ?endpoint ;
           msm:hasOperation ?operation ;
           sawsdl:modelReference <http://www.example.org/weather#WindSpeed> .
  ?operation msm:hasInputMessage ?input ;
             msm:hasOutputMessage ?output .
  ?input sawsdl:modelReference <http://www.w3.org/2003/01/geo/wgs84_pos#long> ;
  sawsdl:modelReference <http://www.w3.org/2003/01/geo/wgs84_pos#lat> .

```

---

This passage from a goal specification to a set of possible service endpoints is followed for all goal elements in the LPML model. At the level of the design-time composer any of the discovered endpoints might be selectable, as they fulfill the basic requirements: typing, as well as input and output parameters. In order to optimize a process, it is however necessary to choose the service that best fits a user's objective and expectations, and ranking becomes important. The ranking and selection service is invoked with the set of possible service endpoints as input, and the best match will be selected for invocation.

## 5. Conclusion

Embracing service-oriented architectures in the context of the Web raises new challenges: increased size and load in terms of users and services, distribution, and dynamicity. The project SOA4All addresses these issues in order to bring so-far mainly business-minded service technologies to the masses. SOA4All paves the way for a Web of billions of services, overcoming many drawbacks of current Semantic Web service technology by leveraging existing Web and Web 2.0 principles and lightweight semantic annotations for services and process in a global service delivery platform.

In this chapter we have presented the concepts and architecture of SOA4All's service delivery platform that is grounded in a federated distributed service bus infrastructure and a set of platform services that deliver the minimal functionality needed to locate and construct services. The platform is central to the achievement of the project's main objective: offering billions of services to millions of users in novel Web-scale service economies. In this context it becomes evident that automation is required, as otherwise, the wealth of service and process descriptions, the users and their context, and the monitoring data that is gathered during the life-time of services cannot be handled. Considering that there is no automation on the Web without

semantics, not at last due to the heterogeneity and dynamics of resources, SOA4All invests in novel techniques, languages and components for annotating services, and for creating, finding and manipulating services and processes at the semantic level.

Further work is dedicated to the realization of comprehensive functional processes for the global service delivery platform. Other work is concerned with the concretization of more tight bounds between SOA4All and established Web technology. Finally, significant work is required in bringing the SOA4All result to the market and in leveraging the technologies in large scale and open service economies.

## Acknowledgment

The work presented is funded by the SOA4All project ([www.soa4all.eu](http://www.soa4all.eu)) under European Union grant FP7-215219. The authors thank the team working on service location from Karlsruhe Institute of Technology, Seekda and the University of Innsbruck; the team on service construction from Atos Origin, TXT e-Solutions, CEFRIEL, University of Manchester, and SAP; and the colleagues from INRIA, EBM WebSourcing and the University of Innsbruck that are developing the service bus.

## References

- [1] L. Nixon, E. Simperl, R. Krummenacher and F. Martin-Recuerda, Tuplespace-based computing for the Semantic Web: A survey of the state of the art, *Knowledge Engineering Review* **23** (2008), 181-212.
- [2] T. Vitvar, J. Kopecky, J. Viskova, and D. Fensel, WSMO-Lite Annotations for Web services, 5th European Semantic Web Conference, June 2008: 674-689.
- [3] J. Farrell and H. Lausen, Semantic Annotations for WSDL and XML Schema (SAWSDL), W3C Recommendation, August 2007.
- [4] M. Maleshkova, J. Kopecky and C. Pedrinaci, Adapting SAWSDL for Semantic Annotations of RESTful Services, Beyond SAWSDL Workshop at OnTheMove Federated Conferences & Workshops, November 2009: 917-926.
- [5] F. Baude, I. Filali, F. Huet, V. Legrand, E. Mathias, Ph. Merle, C. Ruz, R. Krummenacher, E. Simperl, Ch. Hammerling and J.-P. Lorre, ESB Federation for Large-Scale SOA, 25th Symposium On Applied Computing, June 2010.
- [6] I. Stoica, R. Morris D. Karger M.F. Kaashoek and Hari Balakrishnan, Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications, ACM SIGCOMM, August 2001: 149-160.
- [7] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Schenker, A Scalable Content-Addressable Network, ACM SIGCOMM, August 2001: 161-172.
- [8] N. Steinmetz, H. Lausen and M. Brunner, 7th Web Service Search on Large Scale, International Conference on Service Oriented Computing, November 2009: 437-444.
- [9] C. Pedrinaci, J. Domingue and R. Krummenacher, Services and the Web of Data: An Unexploited Symbiosis, AAAI Spring Symposia, March 2010.
- [10] J. de Bruijn and H. Lausen and A. Polleres and D. Fensel, The Web Service Modeling Language WSM: An Overview, 3rd European Semantic Web Conference, June 2006: 590-604.
- [11] I. Toma, D. Roman, D. Fensel, B. Sapkota and J.M. Gomez, A Multicriteria Service Ranking Approach Based on Non-Functional Properties Rules Evaluation, 5th International Conference on Service-Oriented Computing, November 2007: 435-441.
- [12] F. Schnabel, L. Xu, Y. Goronogioitia, M. Radzinski, F. Lecue, G. Ripa, S. Abels, S. Blood, M. Junghans, A. Mos and N. Mehandjiev, Advanced Specification Of Lightweight, Context-aware Process Modelling Language, SOA4All Project Deliverable, September 2009.
- [13] F. Lécué, Optimizing QoS-Aware Semantic Web Service Composition, International Semantic Web Conference, October 2009: 375-391.
- [14] M. Maleshkova, C. Pedrinaci and J. Domingue, Supporting the Creation of Semantic RESTful Service Descriptions, Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web at 8th International Semantic Web Conference, October 2009.