Adaptive Service Binding with Lightweight Semantic Web Services

Carlos Pedrinaci, Dave Lambert, Maria Maleshkova, Dong Liu, John Domingue, and Reto Krummenacher

Abstract Adaptive service selection is acknowledged to provide a certain number of advantages to optimize the service provisioning process or to cater for advanced service brokering. Semantic Web Services, that is services that have been enriched with semantic annotations have often been used for providing adaptive service selection by deferring the binding of services until runtime. Thus far, however, research on Semantic Web Services has mainly been dominated by rich conceptual frameworks such as WSMO and OWL-S which require a significant effort towards the annotation of services and rely on complex reasoning for which there are no efficient solutions that can scale to the Web yet. In this chapter, inline with current trends on the Semantic Web that sacrifice expressivity in favour of performance, we present a novel approach to providing adaptive service selection that relies on simple conceptual models for services and less expressive formalisms for which there currently exist mature and performant implementations. In particular, we present a set of concep-

Carlos Pedrinaci
Knowledge Media Institute, The Open University, Milton Keynes, UK, e-mail: c.pedrinaci@open.ac.uk

Dave Lambert
Knowledge Media Institute, The Open University, Milton Keynes, UK, e-mail: d.j.lambert@open.ac.uk

Maria Maleshkova
Knowledge Media Institute, The Open University, Milton Keynes, UK, e-mail: m.maleshkova@open.ac.uk

Dong Liu
Knowledge Media Institute, The Open University, Milton Keynes, UK, e-mail: d.liu.open.ac.uk

John Domingue
Knowledge Media Institute, The Open University, Milton Keynes, UK, e-mail: j.b.domingue@open.ac.uk

Reto Krummenacher
Semantic Technology Institute, University of Innsbruck, Austria, e-mail: reto.krummenacher@sti2.at

tual models defined in RDF(S) that support both Web services and Web APIs and we show how simple templates abstracting user requirements can be automatically transformed into SPARQL to enable service selection in a scalable manner.

1 Introduction

Web services provide means for encapsulating software functionality as remotely accessible components, independent of programming language and platform. Considerable effort has been devoted to defining architectures, developing communication middleware, and creating languages and process execution engines that can support the creation of complex distributed systems by seamlessly combining Web services. Service-oriented architectures (SOAs) advocate the development of solutions whereby service providers advertise the services they offer in a shared and publicly accessible repository. Software developers or intelligent applications can then access this repository in order to find suitable services for a given purpose and subsequently invoke them.

Web services have increasingly been used within and in some cases between enterprises. However, despite the essential advantages brought by service-oriented technologies, their use in enterprise settings is not without problems. For instance, the execution of business processes defined in this manner typically relies on rigid process models which interact with a fixed and predefined set of partner services. This rigidity impedes or at least complicates to a large extent very desirable features like the dynamic replacement of services based on their current state, the selection of those that better fit a certain context, etc. Conventional solutions to such problems are brute-force: for example, modifying the process models with somewhat artificial branches. This approach results in models that are more complex, and adapting as well as maintaining them in the light of changing conditions turns out to be a hard task [36]. These limitations are even more important in open environments like the Web, where additional difficulties appear such as the heterogeneity of data formats or the unreliability of servers. Recent trends indicate that other technologies, such as HTTP-based Web APIs, are preferred in these cases [10].

In this chapter we address the rigidity of business processes by providing adaptive service selection. This kind of technique, also known as late-binding, relies on deferring the selection and binding to the service to be executed until runtime so that up to date detailed information concerning the state of the business process and other contextual factors such as the previously monitored performance of services can help to adapt the selection to the (presumably) most optimal or appropriate solution. Adaptive service selection has often been based on exploiting semantic annotations of services. Research in this area has thus far b een mainly driven by rich Semantic Web Services (SWS) conceptual models such as the Web Service Modeling Ontology (WSMO) [17] and OWL-S [29], which rely on expressive knowledge representation formalisms such as Web Service Modeling Language (WSML) [11] and OWL [35] complemented by some rule language. On the basis of these technologies, rich SWS

can be defined on top of which refined service discovery algorithms and techniques can be implemented. However, creating these descriptions represents a significant knowledge acquisition bottleneck, and reasoning over them carries a considerable computational overhead that has limited the scalability of these approaches.

The issue of scalability is one that has always been central in the Semantic Web community, where recent practices, best exemplified by the linked data initiative [5], are disregarding expressivity in favour of performance and scalability. Considerable effort has gone into developing systems that can efficiently support storing, updating, and reasoning over RDFS [30] – a simple ontology representation language for the Web. Additionally, a standardised language and protocol for querying these repositories called SPARQL [38] has been devised which nowadays supports the systematic development of applications on top of large RDF(S) knowledge bases.

The work in this chapter has been conducted in the SOA4All project, an EU funded research programme which aims to harness the scalability of the Web, and to use lightweight Web technologies to begin an incremental approach to reaching its objective of 'enabling a Web of billions of services'. Based on the service portability scenario [REF Use Case Chapter] we present a novel approach to providing adaptive service selection based on lightweight semantic technologies, notably RDFS and SPARQL, so as to provide an efficient and scalable solution that can be applied on a Web scale. In particular, we provide adaptive service selection within workflows by defining processes based on service templates that specify abstract objectives rather than by directly naming concrete services. At runtime, service templates can be directly transformed into SPARQL queries that can be used to retrieve suitable services from existing repositories of service annotations informed by the current conditions and contextual knowledge which includes external information such as the location of a customer, monitoring data, and so forth.

The chapter is organised as follows. We first introduce the case study that is used to explain the technologies and methods described herein (Section 2). We then present background knowledge about Semantic Web Services and introduce related work in the area of service adaptability (Section 3). We then present our overall approach (Section4) and cover each of the main technical aspects implementing it, including the formal model for describing services (Section 4.1), the kinds of service descriptions produced (Section 4.2), the service repository (Section 4.3), and service matchmaking techniques based on SPARQL (Section 4.4). Throughout, we illustrate our approach with examples based on the case study. Finally, in Section 5 we present our main conclusions and introduce lines for future research.

2 A Case Study in Added-Value Services and Service Portability

Chapter [REF Use case Chapter] defines a common use-case to be used throughout the book for illustration, clarity, and coherence purposes. The chapter identifies a number of scenarios and business goals that need to be tackled by mobile operators.

In this chapter we focus on two of these business goals: the provisioning of added-value services (TELCO-BG-05), and supporting service-portability (TELCO-BG-04).

The first business goal—providing added-value services—is currently attracting more interest from telecom operators since they otherwise run the risk of becoming "dumb pipe providers" [10]. This business goal is largely aligned with current Web trends where Web APIs and RESTful services are increasingly being offered, and where an appropriate combination and integration of the data these services provide enables the provisioning of a wealth of useful low-cost added-value services. The second business goal is also related to existing trends, in this case on customers mobility, and on the fact that currently client terminals are increasingly powerful and can give access to all sorts of services offered by operators as well as openly on the Web. Both business goals come hand-in-hand as strategically important for telecom operators that aim to provide added-value services to give themselves a competitive advantage and additional revenues. Both business goals are at the core of a use-case lead by BT in SOA4All, and in general are strongly aligned with the overall goal pursued by the project which seeks to support the creation of a Web where billions of services are offered and consumed by billions of providers and customers [10].

Given that both business goals are highly generic, we use a more concrete scenario for illustration purposes. The reader should note, however, that the models, techniques and systems introduced in this chapter could be applied in a wide range of scenarios. The scenario that we have chosen is one where the Cell Phone Operator offers a simple added-value service for frequent travelers allowing them to receive timely notification of traffic reports, or delays in flights and trains so that they can rearrange their trip if necessary. This information is made available via subscription through SMS messages, or on demand using a Web-based interface.

Let us imagine a frequent traveller that has to spend a few days in Vienna to attend a meeting, and then returns to London. While in Vienna, she wants to be aware of the local traffic, so as to reach her various meetings across the city on time. The traffic reports are naturally location specific, so it is necessary to know the desired location for obtaining the reports which in turn determines which of the publicly available services is to be contacted. In this case, the selection of the service to invoke to obtain traffic information is based on the physical location of the phone and the Cell Phone Operator of Austria can directly deal with this request.

After a few days of meeting in Vienna she has to return to London. She therefore wants to know if the flight and tube line have any delays or planned disruptions. In this case the necessary information concerning her journey is stored within an online travel management system like TripIt.com that only authorised systems can access. In this specific case, the Cell Phone Operator she is using in Austria needs to redirect the invocation at the expense of a small roaming fee. Indeed the Cell Phone Operator wants to offer a service that is able to deal with a wide-range of locations (virtually anywhere) and it is of utmost importance that this service is available for the customers at anytime in a completely transparent manner.

Achieving these goals presents a number of technical requirements that are worth highlighting and are schematically depicted in Figure 1. First and foremost, offering

access to such a wide-range of services offered by third parties requires a means for appropriately brokering services by dealing with data heterogeneity and supporting the selection of the most appropriate services to invoke given the location and kind of transportation used. This also highlights the need for supporting the use of diverse kinds of service technologies including WSDL but also Web APIS and RESTful ones offered directly via HTTP. Additionally, because of the need to have access to a very large amount of services it is not appropriate to embed the service selection within the process definition extensionally through a direct hard-wired inclusion of the service. Instead, an intensional definition using declarative statements regarding the suitability of services is a more appropriate solution. It is only through this manner that the process model for carrying out these activities can remain simple and that new information providers can easily be added or removed as the need arises or based on their current state, performance, etc. Finally, in order to support the portability of services it is necessary that Cell Phone Operators of remote countries can transparently redirect the request of customers of other operators or can directly deal with the request if no privately owned information is involved.

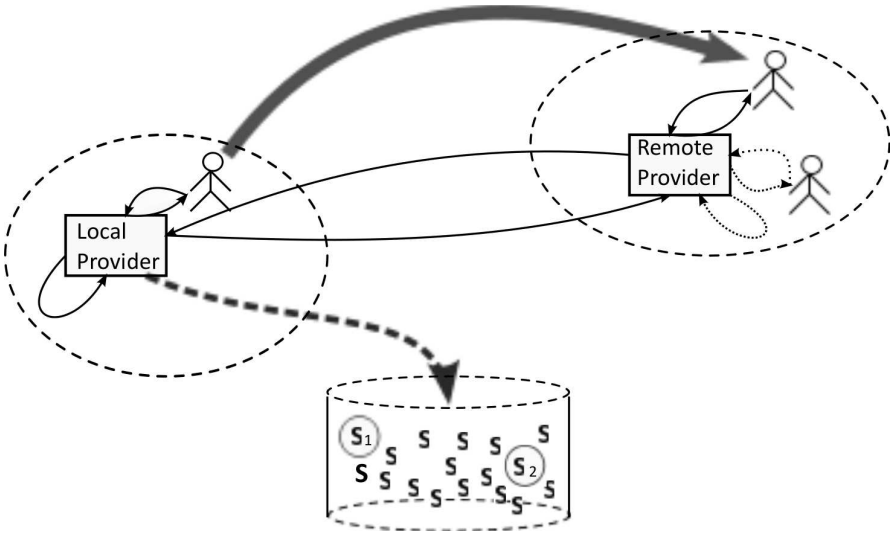

Fig. 1 Service portability through adaptive search.
On the left is the Vienna locale, service provider, and our user. On the right, the users and providers in London. At the bottom is a service repository.

In the remainder of this chapter we describe how, by means of simple semantic annotations, it is possible to achieve these two business goals. For the sake of clarity and simplicity, we shall refer throughout the chapter to traffic report services that take as input a geographical location *location* and a human language identifier *language*, and supply a textual summary in *language* of the traffic situation in *location*. Similarly, we shall use several ontologies to describe the services. These

include the ontologies for standard parts of the semantic web, our framework, and domain specific vocabularies for traffic reporting. In particular, we use an ontology of human languages[1], and use a W3C ontology for discussing geographical location.

3 Background and Related Work

Web Services are software systems that offer their functionality over the Internet via platform and programming-language independent interfaces defined on the basis of a set of open standards such as WSDL, SOAP and further WS-* specifications [13]. Constructing distributed systems out of Web services is a matter of identifying suitable Web services and orchestrating them (through control and dataflow) in such a way that they achieve the desired goal. However, there exist situations, as faced in the case study that we address in this chapter, where the service to be used for execution within a concrete workflow depends on conditions that are knowable only at runtime (e.g., location of the requester), or where optimisations can be achieved by tracking certain aspects such as the overall performance exhibited by equivalent services. In the remainder of this section we shall review some of the main proposals for improving flexibility in SOAs, focusing on research in Semantic Web Services, since that area is the basis for our work.

3.1 Adaptive Service Binding

Since the advent of Service-Oriented Computing, much research has been devoted to discovering means for applications to take into account changing environments in order to adapt to the situation at hand [34]. Research in this area spans a wide range of topics including compensation handling, self-configuration, self-management, self-adaptation, and self-healing. One often pursued line of work on providing a certain level of adaptability within workflows is based on Quality of Service QoS aware binding mechanisms [2,22,50]. The approaches suggested differ significantly in the means for obtaining relevant data, how the QoS is measured and computed, as well as the techniques for bringing a certain level of adaptability within processes. [2] highlights the importance of including QoS information while selecting services. [50] proposes middleware that is able to compute local and global optimisations in order to obtain the best composition at runtime. [22] focusses on outlining how processes defined in BPEL can be enhanced with adaptive capabilities by combining it with Aspect-Oriented Programming.

The aforementioned approaches rely to a certain extent on well-known and controlled environments whereby the services that can be used are known in advance. A different point in the services space is characterised by the use of Semantic Web Ser-

[1] One such vocabulary is at http://www.lingvoj.org/lang/, but it does not quite fit our purpose. Since its use here is pedagogical, we take some liberties with respect to its actual content.

vices and related execution engines and machinery which aim to cater also for open environments like the Web where assumptions about data homogeneity or service availability for example cannot be made a priori. In this respect it is worth mentioning the work carried out in the METEOR-S project through the use of WSDL-S and the notion of service templates as a means to provide late-binding facilities [47]. This work is indeed closely related to ours in that we share the notion of a service template as a placeholder for describing intensionally a family of services that are suitable, and which must be retrieved and ranked at runtime. Similarly, work around WSMO [17] has proposed the use of goals within orchestrations thus allowing the process activities to be resolved and the best one invoked at runtime based on the functionality provide and other arbitrary concerns such as QoS, cost, or trust [19, 32, 33].

Finally, we consider context-aware systems. Although research in this field often concentrates on the distribution of sensors for gathering contextual information, its representation and processing mechanisms, their aim is to provide adaptive systems that can better tackle the situations at hand by being aware of the surrounding contextual conditions. Research in this area has therefore produced a number of architectures, conceptual models and approaches that are of particular relevance to our endeavour. The reader is referred to [4] for a survey on these matters and to [41] for a selection of methods, architectures and technologies bringing context-awareness to Web service technologies.

3.2 Semantic Web Services

Although service-oriented systems are highly appealing from an engineering perspective, developing them requires substantial manual effort to locate, interpret and integrate services. Consequently, Web services are mostly used within controlled environments such as large enterprises rather than on the (public) Web [10]. This is illustrated by the fact that currently there are only 28,000 Web services on the Web[2], whereas organizations like Verizon are estimated to have around 1,500 Web services deployed internally [10]. It has been argued that one possible reason for this lack of take up is that WS-* Web Services do not fully embrace the principles of the Web [10, 48].

Recently, the world of Web services has changed significantly with the proliferation of Web APIS, also called RESTful services [39] when they conform to the architectural style [18]. This kind of service is characterized by simplicity (at least compared to WS-*) and is typically used in conjunction with Web 2.0 technologies and social networking applications. These services are usually described in natural language, on unstructured HTML pages. Attempts to introduce WSDL-style machine readable descriptions [20] have not been popular with developers. As a consequence, and despite their popularity, the development of Web applications that integrate dis-

[2] http://webservices.seekda.com/

parate services in this manner suffers from a number of limitations similar to those we previously outlined for (standard) Web services with an increased complexity due to the fact that most often no machine-processable description is available. Discovering services, handling heterogeneous data, and creating service compositions are largely manual, tedious tasks which result in the development of custom tailored solutions that use these services.

Semantic Web Services were proposed as an extension of Web services with semantic descriptions in order to provide formal declarative definitions of their interfaces, and what the services do [31]. The essential characteristic of SWS is therefore the use of knowledge representation languages with well-defined semantics, e.g., RDFS [30], OWL [35] and WSML [11] to name a few, that are amenable to automated reasoning. On the basis of these semantic descriptions, SWS technologies seek to increase the level of automation that can be achieved throughout the life-cycle of service-oriented applications which include the discovery and selection of services, their composition, their execution and their monitoring among others. Part of the research on SWS has been devoted precisely to identifying the requirements for SWS systems, and defining conceptual frameworks and architectures that cover the entire life-cycle of SWS [7, 12, 15, 17, 31, 33, 44].

The main approaches devised so far can roughly be divided into top-down and bottom-up. Top-down approaches to the development of semantic Web services like WSMO [17] and OWL-S [29] are based on the definition of high-level ontologies providing expressive frameworks for describing Web services. On the other hand, bottom-up models such as WSDL-S [1] and the Semantic Annotations for WSDL and XML Schema (SAWSDL) [14] adopt an incremental approach to adding semantics to existing Web services standards, adding to WSDL specific extensions that connect the syntactic definitions to their semantic annotations.

The landscape of Semantic Web Services is thus characterized by a number of conceptual models that, despite a few common characteristics, remain essentially incompatible due to the different representation languages and expressivity utilized as well as because of conceptual differences. For example, WSMO contains the notions of goal (to represent the client/user perspective) and mediator (to resolve heterogeneities), which have no equivalent in OWL-S. SAWSDL differs significantly from both OWL-S and WSMO, leaving aside the definition of processes and the provisioning of a high-level conceptual model, focusing instead of providing a minimal yet extensible syntactic extension on top of existing standards for describing Web services (WSDL) and their data model (XML Schema). Regardless of the concrete approach followed, the vast majority of the Semantic Web Services initiatives were based upon adding semantics to WSDL Web services. It is only recently that researchers have started focusing on Web APIs and RESTful services, the main examples being SA-REST [42] and MicroWSMO [28].

Service selection, also referred to as matching or matchmaking in several papers, has been a core research topic of the SWS community. The goal of service matchmaking is to, given a request for retrieving some *kind of* Web service, i.e., that is a family of services that meet certain criteria, identify all those Web service advertisements that match to a certain degree the request. Perhaps the best known

matchmaker is Sycara et al.'s SEMANTIC MATCHMAKER [44]. It was one of several matchmakers proposed for OWL-S which matched requests according to input and output types [27, 45]. Each request's input and output types are compared pairwise with the corresponding inputs and outputs of a service description, with the comparisons resulting in per-variable matches named *exact* (the requested and offered types are the same), *plugin* (the offer is a subclass of the request), *subsumes* (the offer is a superclass of the request), and *fail* (there is no relationship). These matches are assigned numerical scores, and a service's degree of match to the original query is determined by the product of the scores for each variable.

In WSMO, the concept of a goal is used to specify problems from a client's perspective. The goals are defined using preconditions and effects, in the manner of planning operators, and through processing by heavyweight reasoners, they can be resolved with a service or orchestration of services. Examples of WSMO-based discovery engines are for instance IRS-III [12], Glue [46] or the template-based system described in [43]. The latter is quite relevant to the work presented here since it also uses the notion of templates although in this case the templates have to be structured in a hierarchy that can be exploited to speed up the search process, and therefore achieves performance improvements in controlled environments restricted to a particular domain where deep hierarchies can be defined.

Computing subsumption relations in a description logic is not a reliably fast operation, and working with the more abstract relation between goals and services is harder still. There have been some recent attempts to improve performance. The MX matchmaker treats OWL classes as keywords, trading subsumption for a vector-space similarity measure akin to information retrieval. Quality of matching is claimed to be as good as subsumption, while performing ten times faster [23].

Independently from the concrete conceptual framework used the aforementioned approaches rely on rich models that have significantly limited their uptake for two main reasons. First, they require considerable human labour for annotating the services. Secondly, these models being based on expressive knowledge representation formalisms, their complexity is such that reasoning over services descriptions is computationally demanding which in turn limits their scalability. The reader is referred to [11], [35], and [21] for details on the computational complexity of WSML, OWL, SWRL – the rule language often used in OWL-S descriptions – respectively).

In this chapter, as opposed to the research described above, we present a novel approach to providing adaptive service selection based on semantic annotations of services. The novelty lies on the use of lightweight semantic technologies which limit the expressivity of service annotations and hence the reduce the potential for carrying out highly complex service matching, in order to simplify the creation of these annotations for developers as well as to support their efficient and scalable manipulation.

4 Scalable Late-binding of Services based on Lightweight Semantic Annotations

In traditional workflows and business processes the execution relies on syntactically specified and rigid process models which interact with a fixed and predefined set of partner services. This rigidity impedes the provision of desirable features like the replacement of services based on their current state, the selection of those that better fit a certain context, etc. A typical approach is to modify the process models with artificial branches which exist only to work around implementation-level difficulties. Unfortunately with this approach, the resulting models are more complex, and maintaining or extending them to adapt to changing conditions becomes a harder task [36].

Our approach to this problem is based on the use of Semantic Web Services, that is of semantic annotations of services that support the application of automated machinery in order to reason about the functional and nonfunctional characteristics of services. In particular, we advocate that workflow definitions use service templates as internal activities instead of concrete and prefixed services whenever flexibility in service selection is desired. At runtime, these service templates can be bound to specific services selected on the basis of the existing conditions and informed by contextual knowledge which may include monitoring data, user location or other aspects that may affect which service is the most appropriate. Since service templates are described semantically, both the required functional and nonfunctional properties have clear semantics. This enhances the interpretation of services by humans, and more importantly, it allows service selection and data mismatches to be resolved at runtime, hence the term late-binding, as supported by Semantic Web Services middleware such as the so-called Semantic Execution Environments [16, 33].

Replacing services by service templates that define intensionally 'families of services' brings a number of benefits from a process execution perspective:

- Process models are relatively independent of the services used. If a particular service is not available the middleware can choose another functionally equivalent service without needing to change the process model. Similarly, and of particular relevance for the case study used within this chapter, should a service not be suitable due to the location of service client (e.g., the user is currently abroad), the middleware can automatically redirect the client request to the right service in a way that is completely transparent for the customer and that requires no adaptation of the client terminal (i.e., the mobile phone in our case).
- Process models are independent of the services' internal data model. The process model has its own semantically annotated data model. If the partner's data model differs from that, semantic models help to bridge the gap.
- Partner services can be selected based on business aspects. Nonfunctional information about cost, quality of service, trust and legal constraints to name a few, can be taken into account so that the selected service is the most suitable from a business perspective.

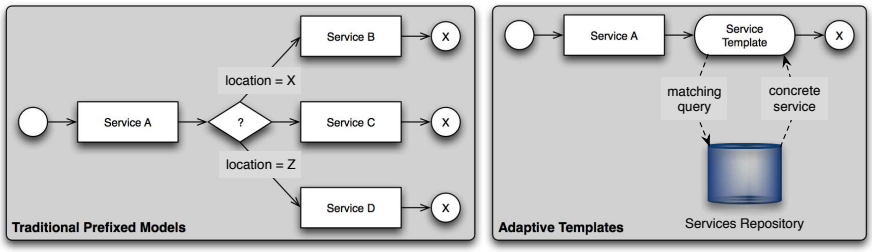

Fig. 2 Illustration of changes brought by our approach.

The overall approach, depicted in Figure 2, therefore relies on the provisioning of semantic annotations for services and the corresponding storage and querying system, on the replacement of workflow activities by service templates, and on the adaptation of execution environments in order to take service templates into account and trigger the selection of appropriate services automatically. The research presented herein builds upon our previous experience [36] of using WSMO goals [17] for defining service templates within BPEL processes based on the BPEL4SWS extensions [32, 33]. The focus of this research, however, lies in an attempt to bring our technologies closer to the Web by:

- embracing current trends on services on the Web such as RESTful services and Web APIs;
- building upon existing Web standards and technologies to better support the uptake of these technologies in a Web-scale; and
- reducing the complexity of the semantic descriptions in order to support faster and more scalable service matchmaking while reducing the knowledge acquisition bottleneck faced when using rich Semantic Web Services descriptions such as WSMO [10].

In this chapter we will therefore present the technologies and methods used to achieve the goals above, leaving aside aspects such as the internals of Semantic Execution Environments, or negotiation steps [9] for the sake of clarity and space. The interested reader is referred to [16, 17, 32, 33, 36] for further details. In the remainder of this chapter we shall cover how we support the adaptive binding of services at runtime in a scalable manner by using service templates which can be automatically transformed into SPARQL queries that can interpreted by state of the art RDF stores to select suitable services efficiently. We first cover in Section 4.1 the formalisms used for describing services semantically using RDFS. We then show in Section 4.2 and Section 4.3 how one can use state of the art technologies from the Semantic Web to support the seamless publishing of services in a way that enables and simplifies their discovery by interested parties in a scalable manner. Finally, in Section 4.4 we introduce a model for describing service templates and we present a simple algorithm that can translate these into SPARQL queries to support the selection

of services at runtime, based on functional and nonfunctional parameters such as contextual information.

4.1 Lightweight Semantic Descriptions for Services on the Web

Currently, there are two main communities, which define competing frameworks for describing semantics for services. These efforts are WSMO [17] and OWL-S [**?**], and they follow a top-down approach for enhancing Web service technology with semantics. In particular, they assume that the service semantic model and the actual service invocation and communication mechanisms are defined in parallel or jointly at design time. As a result, the description of a newly engineered service already comprises semantic information. Even though WSMO and OWL-S have been used in some application areas, this approach is not well suited to incrementally enhancing existing systems based on the service-oriented architecture, where thousands of WSDL-described services or Web APIs are already available on the Web or in intranets. Therefore, it is problematical to use these semantic frameworks to extend existing services.

To better address the problem of incremental or ad-hoc semantic annotation of services, we base our approach on WSMO-Lite [49], a minimal extension to SAWSDL. WSMO-Lite provides a means to create lightweight semantic service descriptions in RDFS [6] by annotating various WSDL elements in accordance with SAWSDL [14] annotation mechanism. In parallel, we use MicroWSMO [28] to annotate services that are not described in WSDL. MicroWSMO is a microformat-based language based on the same kinds of annotations as WSMO-Lite, adapted to support the annotation of HTML-based descriptions of Web APIs. Finally, we provide the minimal service model, a simple RDFS model that provides 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 when at selection time.

4.1.1 WSMO-Lite

To date, SAWSDL is the only Semantic Web Services specification that is a W3C Recommendation. It defines a set of extensions to WSDL, as well as rules for linking WSDL elements to semantic information. In particular, SAWSDL supports three kinds of annotations over WSDL and XML Schema, namely *modelReference*, *liftingSchemaMapping* and *loweringSchemaMapping*. These three annotation types enable links to be made from parts of a WSDL document to associated semantic elements, or to the specifications of data transformations from a syntactic representation to its semantic counterpart and vice versa. In this way, it enables the incremental addition of semantics on top of existing WSDL descriptions, providing a basis for extending results from a well-established approach. However, SAWSDL only provides simple means for connecting service elements to semantic entities and does

not define any concrete service semantics such as types, formats or model for the semantic descriptions.

WSMO-Lite builds upon SAWSDL, overcoming some of SAWSDL's limitations while remaining lightweight. WSMO-Lite makes explicit the intended meaning for *modelReference* annotations without modifying SAWSDL but rather informing users on how they should structure the models their annotations point to. For instance, if the annotation is a functional categorisation, the URI the *modelReference* points to should be that of a sub-class of an instance of *wsl:FunctionalClassificationRoot* (see Listing 4)[3]. Similarly, should the *modelReference* point to the effect of a service, the URI should be that of an instance of *wsl:Effect*. The WSMO-Lite ontology can be used to capture four aspects of service semantics:

- The *Information Model* defines the data model. In particular, it describes the model for input and output messages and is represented by using a domain ontology, along with associated data lifting and lowering transformations.
- The *Functional Semantics* define what the service does by using functionality classification or preconditions and effects. It describes what the service can offer to its clients when it is invoked by assigning it to a particular class of service functionality, defined in a classification ontology.
- The *Behavioral Semantics* define the sequencing of operation invocations when invoking the service. The behavior of a service can be described through a choreography or a workflow definition. However, behavioral semantics are not represented explicitly in WSMO-Lite.
- The *Nonfunctional Semantics* define any service-specific implementation or operational details such as service policies, implementation information or running environment requirements. Nonfunctional properties can include the price of a service or the Quality of Service (QoS) aspects such as performance and reliability. These are defined by using ontologies for nonfunctional properties, which should in this case be grounded on *wsl:NonFunctionalParameter*

4.1.2 MicroWSMO

In addition to the lightweight description of WSDL-based services, our approach supports also the adaptive use of Web APIs in business processes. MicroWSMO forms the basis for our work on semantically describing Web APIs. MicroWSMO uses microformats for adding semantic information on top of existing HTML Web API documentation, by relying on hRESTS (HTML for RESTful services) [26] for marking service properties and making the descriptions machine-processable. hRESTS provides a number of HTML classes that enable the marking of service operations, inputs and outputs, HTTP methods and labels, by inserting HTML tags within the HTML. It therefore enables, through simple injections of HTML code into Web pages, the

[3] Throughout the chapter we assume a set of namespaces are declared for clarity and space reasons. The namespaces used can be found in the Appendix.

transformation of unstructured HTML-based descriptions of Web APIs into structured services descriptions similar to those provided by WSDL.

With the hRESTS structure in place, HTML service descriptions can be annotated further by including pointers to the semantics of the service, operations and data manipulated. Similarly to WSMO-Lite, MicroWSMO adopts the SAWSDL annotation mechanisms and uses three main types of link relations: 1) model, which can be used on any service property to point to appropriate semantic concepts identified by URIs; 2) lifting and 3) lowering, which associate messages with appropriate transformations (also identified by URIs) between the underlying technical format such as XML and a semantic knowledge representation format such as RDF. Therefore, MicroWSMO, based on hRESTS, enables the semantic annotation of Web APIs in the same way in which SAWSDL, based on WSDL, supports the annotation of Web services. MicroWSMO also adopts the WSMO-Lite ontology as the reference ontology for annotating Web APIs semantically. By doing so, both WSDL services and RESTful services, annotated with WSMO-Lite and MicroWSMO respectively, can be treated homogeneously.

4.1.3 Minimal Service Model

The Minimal Service Model, depicted in Figure3, provides a minimal and common conceptual model in RDFS for capturing the semantics of services may they be WSDL-based or Web APIs. It therefore provides the ground for treating them homogeneously when carrying out tasks such as the discovery of services. The Minimal Service Model given in Listing 4 builds upon a number of modules, including SAWSDL's syntactic properties, WSMO-Lite as a minimal extension to SAWSDL, and hRESTS's support for Web APIs. The Minimal Service Model defines services as having a number of *operations*, each of which have *input* and *output messages* and *faults*. Web APIs are supported through the addition of two hRESTS properties, including the *address* as a URI template, and the HTTP *method*. Finally, the Minimal Service Model is completed by the SAWSDL elements for linking semantic information through the *modelReference* and for providing lifting and lowering mechanisms.

The Minimal Service Model captures the essence of services in a way that can support service discovery, matchmaking and invocation by directly operating on the model properties. However, despite its simplicity, it is still broadly compatible with WSMO and OWL-S service models, as well as with services annotated according to WSMO-Lite and MicroWSMO principles. Although providing a formal mapping for each of these languages is out of the scope of this work, we note that the elements captured in the minimal service model are common to existing models, with the exception of the hRESTS extensions specific to RESTful services. Therefore, we base our approach for adaptive device discovery, described in the following sections, on this minimal service model.

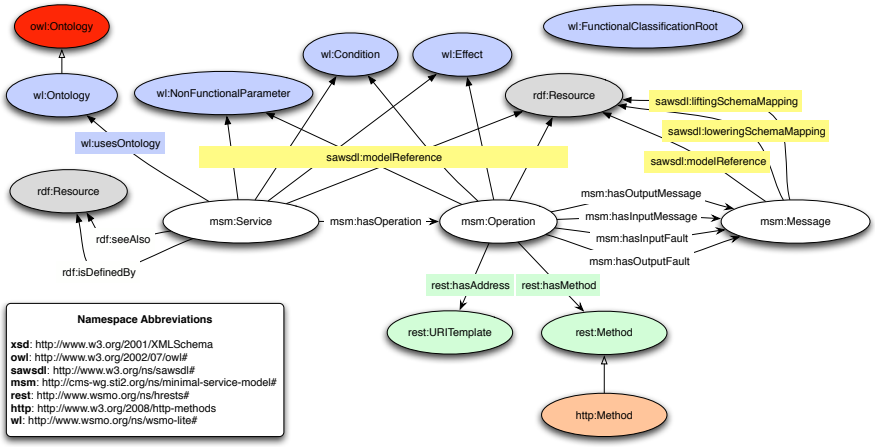

Fig. 3 Service models used. See Appendix for the corresponding N3 serialisation.

4.2 Services and Annotations

In this section, we show the kinds of WSMO-Lite descriptions that might be available for the kinds of traffic report services we imagine in our use-case. In listings 1 and 2, we see N3 represenations of the RDF resulting from processing MicroWSMO descriptions. The reader is referred to [28] for more details on how annotations can be created and how RDF can be extracted from these annotations. Common input parameters are the latitude, longitude, and language of the user, while common output parameters include the number of traffic incidents, incident names, times, resulting delays, and textual descriptions.

Listing 1 Traffic information service in the UK

```
1
2  service1 rdf:type msm:Service ;
3       rdfs:isDefinedBy <http://myTrafficInformation.co.uk/descriptionxmlapi.asp# trafficinfo > ;
4       sawsdl:modelReference <http://www.service-finder.eu/ontologies/ServiceCategories#Travel> ,
5            <http://www.service-finder.eu/ontologies/ServiceOntology#AuthenticationModel> ,
6            <http://example.com/classification/onto# trafficInformation > ,
7            <http://example.com/mimetype/onto#xmlSupport> .
8  operation1 rdf:type msm:Operation ;
9       rdfs:label "UK Traffic Information" ;
10      hr:hasMethod "GET" ;
11      hr:hasAddress "http://myTrafficInformation.co.uk/ xmlgettrafficinfo .aspx" .
12 inputmsg rdf:type msm:Message ;
13      sawsdl:loweringSchemaMapping <http://example.com/UKTraffic-Info-lowering.xsparql> ;
14      sawsdl:modelReference <http://example.com/onto#User> ,
15           geo:long , geo:lat ,
16           <http://www.geonames.org/ontology#Map> ,
17           <http://example.com/geoontology/onto#Range> ,
18           <http://example.com/imagesontology/onto#relatedImageWidth> ,
19           <http://example.com/textontology/onto#Language> .
20 operation1 msm:hasInputMessage :inputmsg .
21 outputmsg rdf:type msm:Message ;
22      sawsdl:liftingSchemaMapping <http://example.com/UKtraffic-Info-lifting.xsparql> ;
```

```
        sawsdl:modelReference <http://example.com/trafficontology/onto#Number> ,
            <http://example.com/trafficontology/onto#IncidentName> ,
            <http://example.com/trafficontology/onto#Created> ,
            <http://example.com/trafficontology/onto#Delay> ,
            <http://example.com/trafficontology/onto#Duration> ,
            <http://example.com/trafficontology/onto#Description> ,
            <http://example.com/trafficontology/onto#Updated> ,
            <http://example.com/trafficontology/onto#TrafficArea> .
    operation1 msm:hasOutputMessage :outputmsg .
    service1 msm:hasOperation :operation1 .
```

Listing 2 Service for Traffic Information in Austria

```

    service2 rdf:type msm:Service ;
            rdfs:isDefinedBy <http:// traffic .de/web-services.html#trafficinfo> ;
            sawsdl:modelReference <http://www.service-finder.eu/ontologies/ServiceCategories#Travel> ,
                    <http://example.com/payment/onto#Free> ,
                    <http://example.com/classification/onto# trafficInformation > ,
                    <http://example.com/mimetype/onto#json> .
    operation1 rdf:type msm:Operation ;
            rdfs:label "Staumeldungen fuer Oestereich" ;
            hr:hasMethod "GET" ;
            hr:hasAddress "http:// traffic .de/trafficinfoJSON" .
    inputmsg rdf:type msm:Message ;
            sawsdl:loweringSchemaMapping <http://example.com/DETraffic-Info-lowering.xsparql> ;
            sawsdl:modelReference geo:long , geo:lat ,
                    <http://example.com/onto#CurrentTime> ,
                    <http://example.com/textontology/onto#Language> .
    operation1 msm:hasInputMessage :inputmsg .
    outputmsg rdf:type msm:Message ;
            sawsdl:liftingSchemaMapping <http://example.com/DETraffic-Info-lowering.xsparql> ;
            sawsdl:modelReference <http://example.com/trafficontology/onto#Number> ,
                    <http://example.com/trafficontology/onto#IncidentName> ,
                    <http://example.com/trafficontology/onto#Priority> ,
                    <http://example.com/trafficontology/onto#Created> ,
                    <http://example.com/trafficontology/onto#Delay> ,
                    <http://example.com/trafficontology/onto#Description> ,
                    <http://example.com/trafficontology/onto#Diversion> .
    operation1 msm:hasOutputMessage :outputmsg .
    service2 msm:hasOperation :operation1 .
```

4.3 Services Publication

The object of syntactic and semantic descriptions of Web services is to provide information about services in a way that can automatically be processed by machines. However, at present, these descriptions can only be retrieved through the Web of documents, which is essentially designed for human beings, or through specific interfaces to silos of services such as UDDI [8] that have failed to see significant uptake. This is particularly true for syntactic descriptions of services (although there is an RDF mapping for WSDL), but also for semantic descriptions, which have remained somewhat disconnected from current practices in the Web of Data [5].

A fundamental step for bringing services closer to the Web, thus better enabling their discovery and supporting their use, is publishing them based on current best practices on the Web. A key component within our approach is therefore a service

publishing platform that plays a role similar to UDDI registries but which is based on a set of fundamentally different principles and technologies. In particular, we advocate the publishing of service annotations as Linked Data. The term Linked Data refers to a set of best practices for publishing structured data on the Web which is based on four main principles [5]:

1. Use URIs for naming things;
2. Use HTTP URIs so that also people can look up those names using Web browsers;
3. Provide information using the standards (RDF, SPARQL); and
4. Include links to other URIs, so that people and machines can discover more things.

Linked Data principles have already been adopted by a growing number of data providers, leading to an exponential growth of a Web of Data containing billions of assertions across diverse domain such as governmental data, music, and encyclopaedia knowledge. Adopting these very simple principles leads to the creation of a global data space that can be queried, browsed and combined on the fly both by machines and humans thanks to the use of standards like HTTP, RDFS, and SPARQL.

Our notion of a service repository is built around the notion of service registry always present in Service-Oriented Architectures, as well as on the Linked Data principles highlighted above. In particular, we view the service repository as a platform that facilitates the publication of semantic annotations of services on the Web as linked data, allowing humans and machines to publish, browse and discover publicly available services, using the models described earlier as the *lingua franca*. It is worth noting in this respect that we here distinguish between the actual syntactic service descriptions in WSDL, WADL or HTML, from the semantic annotations expressed according to the formalisms described previously. It is therefore possible to provide a common view over services of differents kinds in a simple an convenient manner that can serve as a basis for discoverying, querying and using services.

iServe[4] is our implementation of a service repository [37]. iServe provides both an interactive user interface as well as a RESTful API and a SPARQL endpoint that expose services as linked data. It uses as its core conceptual model the Minimal Service Model described previously and it currently includes a number of import mechanisms able to deal with WSDL files including SAWSDL annotations, with descriptions adopting the WSMO-Lite specific extensions, and also with MicroWSMO annotations of Web APIs. This import mechanism, illustrated in Figure 4, transforms the service descriptions into the appropriate terms within the Minimal Service Model and automatically generates *rdfs:definedBy*, *rdfs:seeAlso*, and *owl:sameAs* relations allowing humans and machines to discover additional information. The first relationship is established between the service annotation and the actual document describing the service (e.g., a WSDL file). It therefore allows systems to find the actual interface description definition needed for invocation after it has been determined that the service is the appropriate one to use. The *rdfs:seeAlso* relationship points to documentation and additional information about the service in case

[4] See http://iserve.kmi.open.ac.uk

developers need it. Finally, *owl:sameAs* allows us to assert that one particular service annotation is actually the same as another one published by a third party in some other repository. In this respect it is worth noting that although currently there are no other repositories publishing services in a similar way, owl:sameAs relations are automatically generated linking to the RDF mapping of WSDL [25] so that any application already using this approach internally can directly interact with iServe.

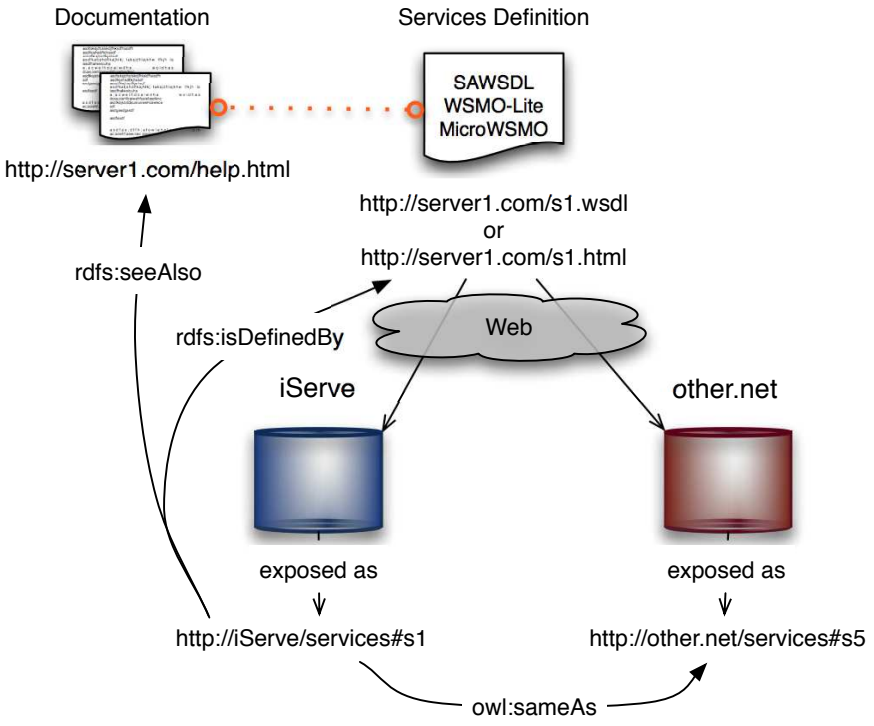

Fig. 4 iServe publishing process.

Currently the data in iServe comes mostly from the SAWSDL Test Collection[5], OWL-S Test Collection[6] and the use cases of the EU project SOA4All. The current implementation already highlights how Web services and Web APIs can be described by means of an homogeneous (but extensible) conceptual model—the Minimal Service Model—and how they can be published as linked data, therefore better promoting their discovery based on the use of well established and adopted principles from the Web of Data. Additionally and by virtue of using existing lightweight semantic technologies such as RDFS and SPARQL, the service repository benefits from a grow-

[5] See http://www.semwebcentral.org/projects/sawsdl-tc/

[6] See http://projects.semwebcentral.org/projects/owls-tc/

ing body of performant and mature tools able to cater for the required scalability for dealing with large numbers of services on the Web.

4.4 Template-based Service Selection

The objective of creating service descriptions is so that we can later find the services, and reason about them. In this section, we are concerned with the problem of connecting a client requiring a service with a service provider. This task is known as matchmaking (in multi-agent systems) or service selection (a more recent term, common in Web services). The basic approach is for the client to create a formal description of their requirement, and for the matchmaker to compare this against the stored service descriptions to find the most appropriate matches [24].

Since our approach is centred on achieving scalability and ease of use by building on Linked Data standards, we introduce simple notions of service requests, using RDF and SPARQL. In our approach, a service request comprises a set of inputs and outputs, as well as the functional classification of a service[7]. We define RDF *service templates* which capture service requests in a similar way to the service descriptions, and transform those templates to SPARQL queries, trading expressivity for scalability. Such templates can be used operationally in workflows, acting as place-holders for late binding services; in workflow composers, enabling automated suggestion of next steps in a pipeline based on type, and enabling partial verification of correctness; in discovery engines, to find suitable services; in ranking and selection engines, to order them by suitability; and in execution engines, to supply values to service invocations.

Despite our caveat that a template is strictly less appropriate than a goal with pre- and post-conditions, the use of input and output types is actually well suited to searching for stateless services (alternatively, information processing services which are defined reasonably well in terms of a mapping from input to output), since in those cases any preconditions and effects would be only be a refinement of the typing of those inputs and outputs. Our chosen scenario of finding a traffic reporting service is one such case.

In designing these templates, we intended that they straightforwardly map into SPARQL queries over a repository of service descriptions using WSMO-Lite and the Minimal Service Model. Our templates take the following form:

```
ServiceTemplate a rdfs:Class.
  hasFunctionalClassification a rdf:Property;
  hasInput a rdf:Property;
  hasOutput a rdf:Property;
  hasPreference a rdf:Property;
  hasRequirement a rdf:Property;
```

[7] A similar functionality is present in OWL-S [29] through the *serviceCategory* property, but we are not aware of any matchmaker that uses it.

The `hasFunctionalClassification` property shadows the use of a service's model reference to an object that is a subclass of `FunctionalClassificationRoot`. These functional classifications might be written to the UNSPSC code. For traffic information reporting, no such UNSPSC code currently exists, so services might instead use the appropriate DBPedia (structured information extracted out of wikipedia) [3] page to create a functional classification:

```
<http://dbpedia.org/resource/Category:Transportation_by_mode>
    rdf:type wsl:FunctionalClassificationRoot .
```

The *Requirement* and *Preference* properties are specified such that:

$$Requirement: Service \rightarrow Bool$$
$$Preference: Service \rightarrow Value$$

The intent with these is to allow the user to specify further constraints in a language of their choice, including SPARQL [38] and WSML [11]. We do not use these constraints in this chapter, but they are analagous to the pre-conditions and effects of services. In section 4.4.3 we will illustrate how they can be used by service providers to specify the geographical limits to the usefulness of their services.

Following our use-case, the service offered is based on a unique service template that captures a family of services that can provide traffic reports. At runtime, the service template needs only be instantiated with concrete data corresponding to the users location and the language required in order to provide all the necessary information for querying the service repository:

```
viennaTrafficRequest rdf:type st:ServiceTemplate ;
  st:hasFunctionalClassfication
    <http://dbpedia.org/page/Category:Road_traffic_management> ;
  st:hasInput [ rdf:type geo:lat; rdf:value "48.033"^^xsd:long ] ;
  st:hasInput [ rdf:type geo:long ; rdf:value "16.366"^^xsd:long ] ;
  st:hasInput [ rdf:type lang:Language ; rdf:value lang:English ] .
```

The reader should note that *Road Traffic Management* is a sub category of *Transportation by Mode* in the categorisation chosen and that Vienna is located at $48°12'31.5''\text{N}, 16°22'21.3''\text{E}$.

4.4.1 Automatic Transformation of Service Templates to SPARQL

Services can be found by creating a SPARQL query, and in particular, queries can be derived from a service template. Since there is no explicit mediation in the model, finding exact matches for input and output types is important. We can connect these inputs to the model references from the SAWSDL and MicroWSMO.

Given our template, we want to transform it to a SPARQL query. The simplest such query (because it is the most specific) is this:

```
SELECT ?service ?operation
WHERE {
    ?service rdf:type msm:Service ;
```

```
        msm:hasOperation ?operation ;
        sawsdl:modelReference
                        <http://dbpedia.org/page/Category:Road_traffic_management> .
    ?operation msm:hasInputMessage ?input ;
        msm:hasOutputMessage ?output .
    ?input sawsdl:modelReference geo:lat ;
        sawsdl:modelReference geo:long ;
        sawsdl:modelReference lang:Language .
```

This provides for services which match exactly the functional classification, and the input types. Had our original service template contained `hasOutput` constraints, those would have appeared in a similar way to the input types. The algorithm for constructing such queries is shown in figure 4.4.1.

EXACT-MATCH-SELECT(*template,repository*)

```
 1  query ← "SELECT ?service ?operation WHERE {"
 2  query ← query + "?service rdf:type msm:Service ;"
 3  query ← query + "?service msm:hasOperation ?operation ;"
 4  query ← query + "sawsdl:modelReference" + template.hasFunctionalClassfication
 5  query ← query + "?operation msm:hasInputMessage ?input"
 6  query ← query + "?operation msm:hasOutputMessage ?output"
 7  for input ∈ template.hasInput
 8      do query ← query + "?input sawsdl:modelReference" + input
 9  for output ∈ template.hasOutput
10      do query ← query + "?output sawsdl:modelReference" + output
11  return EXECUTE-SPARQL(query,repository)
```

Fig. 5 Algorithm for converting service templates to SPARQL `SELECT` queries

We can also write various forms of this to account for the *exact*, *plugin*, *subsumes*, and *fail* match degrees common in the literature. This is done by the addition of subclass relations in the SPARQL queries. For instance, the following would find services which offered a *plugin* alternative to our request:

```
SELECT ?service ?operation
WHERE {
    ?service rdf:type msm:Service ;
        msm:hasOperation ?operation ;
        sawsdl:modelReference ?fcr .
    ?fcr rdfs:subClassOf
        <http://dbpedia.org/page/Category:Road_traffic_management> .
    ?operation msm:hasInputMessage ?input ;
        msm:hasOutputMessage ?output .
    ?input sawsdl:modelReference ?i1 ;
        sawsdl:modelReference ?i2 ;
        sawsdl:modelReference ?i3 .
     ?i1 rdfs:subClassOf geo:long ;
     ?i2 rdfs:subClassOf geo:lat ;
     ?i3 rdfs:subClassOf lang:Language ;
```

4.4.2 Match Reports

We now have a means for creating queries to discover services that match. To return the results, the natural solution is to use another RDF vocabulary which enables the labelling of each matching service with the match degree of the inputs, outputs, and functional classification.

```
st:MatchmakingResults a rdfs:Class .

st:hasServiceTemplate a rdf:Property ;
    rdfs:domain st:MatchmakingResults ;
    rdfs:range st:ServiceTemplate .

st:hasMatch a rdf:Property ;
    rdfs:domain st:MatchmakingResults ;
    rdfs:range st:Match .

st:Match a rdfs:Class .

st:MatchDegree a rdfs:Class .
        st:ExactMatch a st:MatchDegree .
        st:PluginMatch a st:MatchDegree .
        st:SubsumesMatch a st:MatchDegree .

st:hasMatchDegree a rdf:Property ;
    rdfs:domain st:Match ;
    rdfs:range st:MatchDegree .

st:hasMatchingElement a rdf:Property ;
    rdfs:domain st:Match .
```

The `matchDegree` property can be attached to each appropriate `modelReference` in the service description. Computing the match degree requires access to the class hierarchy for all the referenced types, and so can only feasibly be done at the service repository.

4.4.3 Checking Preconditions

The service requestor now has a set of services and operations that have matched to some degree the input and output types, and the functional classification. The client is now in a position to check that the preconditions of the services hold. Service preconditions are written as SPARQL `ASK` queries against the requestor's knowledge base. Essentially the same approach was applied in to OWL-S services [40].

For the purpose of evaluating the preconditions, the requestor's knowledge base will have in it only the active service template. This can be identified with the `?template rdf:type st:ServiceTemplate` pattern. The rest of the precondition for the Austrian travel service is thus:

```
ASK WHERE {
    ?template rdf:type st:ServiceTemplate ;
        st:hasInputMessage ?lattitute ;
        st:hasInputMessage ?longitude ;
        st:hasInputMessage ?language ;
    ?latitude rdf:type geo:lat .
    ?longitude rdf:type geo:long .
```

```
FILTER (?latitude < 48.5 && ?latitude > 46.5 ) .
FILTER (?longitude < 16.6 && ?longitude > 9.6 ) .
```

The two FILTER expressions place a bounding rectangle around Austria's geographical extremities. Only a request for a traffic within that box should be matched.

5 Conclusions and Future Work

Service-Oriented Computing prescribes the development of systems on the basis of reusable distributed components offered as services in a language and platform independent manner. Key to the development of these kinds of systems is the discovery and selection of services, and there has therefore been a wealth of research focusing on these matters. Semantic Web Services research advocates using semantic annotations of services in order to support advanced discovery and selection techniques based on formal descriptions of both functional and nonfunctional aspects of services. On the basis of these techniques, prototypes have been developed that showcase their potential. However, Semantic Web Services technologies have been up to date based essentially on complex, high-level service ontologies and on highly expressive logics. As a consequence Semantic Web Services technologies have faced an important knowledge acquisition bottleneck and Web-scale solutions have yet to be provided.

SOA4All aims to pave the way for a Web of billions of services, overcoming the drawbacks of current Semantic Web Services technologies by using lightweight semantic annotations, existing Web standards and harnessing Web 2.0 principles. In this chapter we have focussed on the annotation of services using RDFS and a simple conceptual model. We have proposed the publication of services following a number of principles from the Semantic Web which enable easier discovery, selection and use of service annotations, and do so in a scalable manner. We have also presented the notion of service templates as declarative specifications of families of services based on restrictions over their functional or nonfunctional aspects and we have proposed their use within workflows to support the late binding of services. Finally, we have illustrated how these service templates can automatically be transformed into SPARQL queries that can be sent directly to service repositories for automatically selecting suitable services.

The work presented in this chapter are a snapshot of ongoing research that are being evaluated within three use-cases, but the results obtained thus far show already that a considerable amount of limitations currently exhibited by Web services and Semantic Web Services technologies can be mitigated to a certain extent. Further progress with respect to the generation of service annotations is expected to be achieved through the development of a fully-fledged Web-based interface for the annotation, composition and invocation of services known as the SOA4All Studio. Additionally, in the future we aim to use the benchmarking platform and test cases under development within the EU project SEALS, to carry out a comparative analysis

with current service selection solutions to better account for the tradeoff between expressivity and scalability in the context of concrete domains and test-cases.

Acknowledgements This research was funded by the SOA4All project, under European Union grant FP7-215219.

6 Appendix

Listing 3 Namespaces used throughout the chapter.

```
# Standard semantic web namespaces
@prefix xsd: <http://www.w3.org/2001/XMLSchema> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix sawsdl: <http://www.w3.org/ns/sawsdl#> .

# Minimal Service Model, WSMO-Lite, hRESTS namespaces
@prefix wsl: <http://www.wsmo.org/ns/wsmo-lite#> .
@prefix hr: <http://www.wsmo.org/ns/hrests#> .
@prefix msm: <http://cms-wg.sti2.org/ns/minimal-service-model#> .

# Service Templates
@prefix st:   <http://cms-wg.sti2.org/ns/service-template#> .

# Scenario specific
@prefix lang: <http://http://www.lingvoj.org/lingvoj#> .
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
```

Listing 4 Service models used in RDF(S) including Minimal Service Model, WSMO-Lite, hRESTS, and SAWSDL

```
msm:Service a rdfs:Class .
msm:hasOperation a rdf:Property ;
 rdfs:domain msm:Service ;
 rdfs:range msm:Operation .
msm:Operation a rdfs:Class .
msm:hasInputMessage a rdf:Property ;
 rdfs:domain msm:Operation ;
 rdfs:range msm:Message .
msm:hasOutputMessage a rdf:Property ;
 rdfs:domain msm:Operation ;
 rdfs:range msm:Message .
msm:hasInputFault a rdf:Property ;
 rdfs:domain msm:Operation ;
 rdfs:range msm:Message .
msm:hasOutputFault a rdf:Property ;
 rdfs:domain msm:Operation ;
 rdfs:range msm:Message .
msm:Message a rdfs:Class .
msm:usesOntology a rdfs:Property ;
 rdfs:domain msm:Service ;
 rdfs:subPropertyOf rdfs:seeAlso .
msm:hasFunctionalClassification a rdfs:Property ;
 rdfs:subPropertyOf sawsdl:modelReference .
msm:hasNonfunctionalProperty a rdfs:Property ;
 rdfs:subPropertyOf sawsdl:modelReference .
msm:hasCondition a rdfs:Property ;
```

```
 rdfs:subPropertyOf sawsdl:modelReference .
msm:hasEffect a rdfs:Property ;
 rdfs:subPropertyOf sawsdl:modelReference .

# WSMO-Lite
wsl:Ontology rdfs:subClassOf owl:Ontology.
wsl:FunctionalClassificationRoot rdfs:subClassOf rdfs:Class.
wsl:NonFunctionalParameter a rdfs:Class.
wsl:Condition a rdfs:Class.
wsl:Effect a rdfs:Class.

# hRESTS properties added to the above model
hr:Method a rdfs:Class .
hr:hasAddress a rdf:Property ;
 rdfs:domain msm:Operation ;
 rdfs:range hr:URITemplate .
hr:hasMethod a rdf:Property ;
 rdfs:domain msm:Operation ;
 rdfs:range hr:Method .
# a datatype for URI templates
hr:URITemplate a rdfs:Datatype .
# HTTP Methods possible methods for RESTful services
http:Method rdfs:subClassOf hr:Method .

# SAWSDL properties
sawsdl:modelReference a rdf:Property .
sawsdl:liftingSchemaMapping a rdf:Property .
sawsdl:loweringSchemaMapping a rdf:Property .
```

References

1. Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit Sheth, and Kunal Verma. Web service semantics - wsdl-s, Nov 2005. W3C Member Submission.
2. Eyhab Al-Masri and Qusay Mahmoud. Web service discovery and client goals. *Computer*, 42(1):104 – 107, Jan 2009.
3. Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. In *In proceedings of 6th International Semantic Web Conference, 2nd Asian Semantic Web Conference*, pages 722–735, November 2008.
4. Matthias Baldauf, Schahram Dustdar, and Florian Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, pages 263–277, Jun 2007.
5. Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data - the story so far. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 2009.
6. Dan Brickley and Ramanathan V. Guha. RDF Vocabulary Description Language 1.0: RDF Schema, 2002. http://www.w3.org/TR/rdf-schema.
7. Mark Burstein, Christoph Bussler, Michal Zaremba, Tim Finin, Michael N Huhns, Massimo Paolucci, Amit P Sheth, and Stuart Williams. A semantic web services architecture. *IEEE Internet Computing*, 9(5):72–81, 2005.
8. Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers. UDDI Specification Version 3.0.2. Technical report, OASIS, 2004.
9. Marco Comuzzi, Kyriakos Kritikos, and Pierluigi Plebani. Semantic-aware service quality negotiation. In *ServiceWave '08: Proceedings of the 1st European Conference on Towards a Service-Based Internet*, pages 312–323, Berlin, Heidelberg, 2008. Springer-Verlag.
10. John Davies, John Domingue, Carlos Pedrinaci, Dieter Fensel, Rafael Gonzalez-Cabero, Morgan Potter, Marc Richardson, and Sandra Stincic. Towards the open service web. *BT Technology Journal*, 26(2), 2009.
11. Jos de Bruijn. D16.1v0.21 the web service modeling language wsml, Oct 2005.

12. John Domingue, Liliana Cabral, Stefania Galizia, Vlad Tanasescu, Alessio Gugliotta, Barry Norton, and Carlos Pedrinaci. Irs-iii: A broker-based approach to semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(2):109–132, 2008.
13. Thomas Erl. *SOA Principles of Service Design*. Jul 2007.
14. Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema (SAWSDL). Recommendation, W3C, August 2007.
15. Dieter Fensel and Chris Bussler. The Web Service Modeling Framework WSMF. *Electronic Commerce Research and Applications*, 1(2):113–137, 2002.
16. Dieter Fensel, Mick Kerrigan, and Michal Zaremba. *Implementing Semantic Web Services: The SESA Framework*. 2008.
17. Dieter Fensel, Holger Lausen, Axel Polleres, Jos de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. 2007.
18. Roy T Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, 2000.
19. Stefania Galizia, Alessio Gugliotta, and Carlos Pedrinaci. A formal model for classifying trusted semantic web services. In *3rd Asian Semantic Web Conference (ASWC 2008)*, Bangkok, Thailand, 2008.
20. Marc Hadley. Web application description language. Technical report, Aug 2009.
21. Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML, May 2004. Last Visited: April 2005.
22. Dimka Karastoyanova and Frank Leymann. BPEL'n'Aspects: Adapting Service Orchestration Logic. *IEEE International Conference on Web Services, 2009. ICWS 2009.*, pages 222 – 229, Jul 2009.
23. Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated semantic web service discovery with OWLS-MX. pages 915–922, 2006.
24. Matthias Klusch and Katia Sycara. Brokering and matchmaking for coordination of agent societies: a survey. In *Coordination of Internet agents: models, technologies, and applications*, pages 197–224. Springer-Verlag, 2001.
25. Jacek Kopecký. Web services description language (wsdl) version 2.0: Rdf mapping. Technical report, Jun 2007.
26. Jacek Kopecky, Karthik Gomadam, and Tomas Vitvar. hrests: an html microformat for describing restful web services. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*, Nov 2008.
27. Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, 8(4):39, 2004.
28. Maria Maleshkova, Jacek Kopecký, and Carlos Pedrinaci. Adapting sawsdl for semantic annotations of restful services. In *Workshop: Beyond SAWSDL at OnTheMove Federated Conferences & Workshops*, 2009.
29. David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. OWL-S: Semantic Markup for Web Services. Member submission, W3C, 2004. W3C Member Submission 22 November 2004.
30. Brian McBride. *The Resource Description Framework (RDF) and its Vocabulary Description Language RDFS*, chapter 3, pages 51–66. 2004.
31. Sheila McIlraith, Tran Son, and Honglei Zeng. Semantic web services. *Intelligent Systems, IEEE*, 16(2):46 – 53, Jan 2001.
32. Jorg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann. Bpel for semantic web services (bpel4sws). *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, pages 179–188, 2007.
33. Barry Norton, Carlos Pedrinaci, John Domingue, and Michal Zaremba. Semantic execution environments for semantics-enabled soa. *IT-Methods and Applications of Informatics and Information Technology*, Special Issue in Service-Oriented Architectures:118–121, 2008.

34. Michael P Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. *Computer*, 40(11):38–45, 2007.
35. Peter Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax, Feb 2004. Last Visited: March 2005.
36. Carlos Pedrinaci, Christian Brelage, Tammo van Lessen, John Domingue, Dimka Karastoyanova, and Frank Leymann. Semantic business process management: Scaling up the management of business processes. In *Proceedings of the 2nd IEEE International Conference on Semantic Computing (ICSC) 2008*, Santa Clara, CA, USA, August 2008. IEEE Computer Society.
37. Carlos Pedrinaci, Dong Liu, Maria Maleshkova, Dave Lambert, Jacek Kopecký, and John Domingue. iServe: a Linked Services Publishing Platform. In *Ontology Repositories and Editors for the Semantic Web at 7th Extended Semantic Web Conference*, 2010.
38. Eric Prud'hommeaux and Andy Seaborne. Sparql query language for rdf. Recommendation, W3C, January 2008.
39. Leonard Richardson and Sam Ruby. *RESTful Web Services*. May 2007.
40. Marco Luca Sbodio and Claude Moulin. SPARQL as an expression language for OWL-S. In *OWL-S: Experiences and Directions, a workshop at the 4th European Semantic Web Conference (ESWC 2007)*, June 2007.
41. Quan Sheng, Jian Yu, and Schahram Dustdar. *Enabling Context-Aware Web Services: Methods, Architectures, and Technologies*. Chapman and Hall/CRC, 2010.
42. Amit Sheth, Karthik Gomadam, and Jon Lathem. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. *Internet Computing, IEEE*, 11(6):91 – 94, Nov 2007.
43. Michael Stollberg, Martin Hepp, and Jörg Hoffmann. A Caching Mechanism for Semantic Web Service Discovery. In *Proceedings of the International Semantic Web Conference 2007*. Springer, 2007.
44. Katia Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of semantic web services. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(1):27 – 46, 2003.
45. David Trastour, Claudio Bartolini, and Javier Gonzalez-Castillo. A Semantic Web Approach to Service Description for Matchmaking of Services. In Isabel F. Cruz, Stefan Decker, Jérôme Euzenat, and Deborah L. McGuinness, editors, *Proceedings of SWWS'01, The first Semantic Web Working Symposium*, pages 447–461, 2001.
46. Andrea Turati, Emanuele Della Valle, Dario Cerizza, and Federico M Facca. *Using Glue to Solve the Discovery Scenarios of the SWS-Challenge*, pages 185–197. 2009.
47. Kunal Verma, Kaarthik Sivashanmugam, Amit Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. Meteor-s wsdi: A scalable p2p infrastructure of registries for semantic publication and discovery of web services. *International Journal of Information Technologies and Management*, 6(1):17–39, 2005.
48. Steve Vinoski. Putting the "web" into web services: Interaction models, part 2. *IEEE Internet Computing*, 6(4):90–92, 2002.
49. Tomas Vitvar, Jacek Kopecky, Jana Viskova, and Dieter Fensel. Wsmo-lite annotations for web services. In Manfred Hauswirth, Manolis Koubarakis, and Sean Bechhofer, editors, *Proceedings of the 5th European Semantic Web Conference*, LNCS, Berlin, Heidelberg, June 2008. Springer Verlag.
50. Liangzhao Zeng, Boualem Benatallah, Anne Ngu, Marlon Dumas, Jayant Kalagnanam, and Henry Chang. Qos-aware middleware for web services composition. *Software Engineering, IEEE Transactions on*, 30(5):311 – 327, May 2004.