Towards Optimized Data Fetching for Service Discovery

Maciej Zaremba, Tomas Vitvar, Matthew Moran
Digital Enterprise Research Institute
National University of Ireland, Galway
{firstname.lastname}@deri.org

Abstract

The advent of Service Oriented Architecture makes services the most flexible, platform-independent choice for exposing and integrating business functionality across a network. However, the lack of service discovery mechanisms, that go beyond simple keyword search to enable automated late-binding of service requesters to providers, remains a major problem. Realistic late-binding involves matchmaking of client requests with service descriptions, based on frequently changing fine-grained client requests, and dynamically changing service functionality. The intricacies of service functionality cannot be specified by generic static descriptions since the functionality offered by the service may depend on the client at hand, their specific request and the service's current capabilities. In this paper, we propose a semantic framework supporting dynamic data fetching from services during the discovery phase on service instance level entailing a communication overhead which we aim to minimize.

1 Introduction

In business environments, the specifics of functionality offered by services may change very often. It is not feasible to capture all such details in static descriptions, as is assumed by many semantic and non-semantic approaches to service discovery [13, 2, 19]. Static descriptions are suitable for the initial phase of the discovery but often an interaction with a service is required during discovery to check dynamic properties defining the service's functionality and evaluate its suitability at a given point in time for a given client. On the other hand, a drawback of interacting with a service during the discovery phase can be a significant communication overhead.

An example of static-only service discovery is through the UDDI-based matchmaking process of [2] which takes into account only the static categorization of the Web services and their *tModels* using keyword-based matching mechanisms without any support for the possibility that the service parameters may change dynamically. Such an approach supports discovery only at a broad level while more realistic Web service late-binding needs to be based on client requests at a much finer level of granularity. By late-binding in SOA we understand all runtime tasks that lead up to a service requester and provider being brought together, ready to interact. These tasks include discovery, mediation, selection and others, as explained in [17]. The reasons why late-binding has not become a reality to date include: (1) insufficient expressivity of underlying service descriptions and client search requests supporting only keyword-based matching and (2) only static service functionality descriptions. Changes to the fine grained parameters of a service search (e.g. service pricing) can result in differing sets of matching services. These fine grained parameters are usually missing from the static part of service descriptions meaning that service discovery often remains a manual design-time process while in many cases it could be automated.

For example, a request to "buy a Harry Potter book" usually is abstracted to a search for descriptions of services that sell books. But once such a service is located, it must be determined if the service sells Harry Potter books and if those books are in stock. Taking Amazon as an example, it is clearly unfeasible to include data for its entire catalogue of books and their availability directly in the service description. Such information has a dynamic character and therefore should only be fetched from the service at discovery-time when needed.

Discovery based on static and non-semantic descriptions include the following weaknesses:

- A single Web service may provide quite broad functionality and describing all its aspect may produce large descriptions that quickly become outdated. Frequent re-publishing of descriptions is not practical.
- The intricacies of Web service functionality may depend on the customer at hand and the actual request. For example, there may be different product prices for frequent customers compared to first-time buyers. The service provider may not advertise its pricing policy.

However, providing additional operations which can be invoked in order to find out information like pricing before committing to any real-world action (e.g., buying something) is sound and acceptable from a business point of view. For the client, details of the offer are essential for the service discovery process in deciding which service to use.

- The keyword based approach does not cater for expressing any complex requirements over the service functionality.

This paper extends our previous work on service discovery with support for data fetching [18] towards optimizing the communication required with candidate services. Our approach to minimization of the communication is based on the user request analysis, namely only data which is referred by the user query is fetched from the service. In our approach both user request and service descriptions use ontologies to define their underlying data models. The rich expressivity provides the means for powerful machine-based reasoning, required for fine-grained run-time Web service discovery. This paper shows how a combination of the data-fetching mechanism with ontology-based representations of Web services and user requests makes this possible.

The rest of the paper is structured as the following. In Section 2 we introduce different kinds of semantics relevant for services and our discovery framework. In Section 3, a detailed description of the service discovery with data-fetching support and its algorithm is given. Section 4 describes the implementation and evaluation of our work. Related work is presented in Section 5. Finally, in section 6, we conclude the paper and provide an insight for future work.

2 Definitions

We follow the categorization of the discovery process into two distinct phases as defined in [5], namely:

Web Service discovery operates on the abstract capability level where the functionality of the service is generalized to a high-level description. An example could be Amazon, described as an online product-selling service, where product categories are statically defined and referenced in service descriptions which can be effectively updated as categories and their properties change. On this abstract level, a user request of purchasing a *Harry Potter* book or a *4GB iPod* is generalized to purchase of a *book* or an *mp3 player* respectively. Reflecting entire product catalogue in the static description is not viable and unmaintainable since products, their availability and their prices change dynamically. It would require continuous updates to the catalogue in real time which is rarely an option from business perspective.

Service discovery operates on a concrete instance of a service and depends on a number of factors including: current business circumstances (e.g., product terms and availability), service requester status (e.g., a customer with a long-term relationship or first-time user) and given request (e.g., buy a book and ship it to the certain location). Services like Amazon take user preferences and history into account when generating individual offers. This is fine-grained user specific information requiring Amazon's private business and is generally only taken into account during interaction with the service - not before. In other words it depends on a communicative exchange between the Amazon service and the service requester.

Taking the Amazon example, the level of granularity for *Web service* discovery is not sufficient (buy a book) for automatically determining if the service matches the request (buy a Harry Potter book) however as a first phase of the discovery process it allows to narrow down number of candidate Web services using solely their static description. In the second step, suitability on the candidate services from the first phase can be further examined on their instance level with dynamically obtained service description.

For purposes of our work we use definitions from [16] for semantic description of both parties using the following types of semantics: *information*, *functional*, and *behavioral*.

Information Semantics is the formal definition of some domain knowledge used by the service in its *input* and *output* messages. We describe the information semantics as an ontology defining the terminology of the domain together with a knowledge base as the instantiation of the ontology.

Formally, the information semantics is a structure

$$O = (C, R, E, I) \tag{1}$$

with a set of classes (unary predicates) C, a set of relations (binary and higher-arity predicates) R, a set of explicit instances of C and R called E (extensional definition), and a set of axioms called I (intensional definition) that describe how new instances are inferred.

Functional Semantics is a static description of the service capability, i.e. what the service can offer to its users. We define a capability as

$$F = (\Sigma, \phi^{pre}, \phi^{eff}), \tag{2}$$

where $\Sigma \subseteq (\{x\} \cup C \cup R \cup E)$ is the signature of symbols, i.e. variable names $\{x\}$ or identifiers of elements from C, R, E of some information semantics O; ϕ^{pre} is a precondition which must hold in a state before the service can be invoked and ϕ^{eff} is the effect, a condition which must hold in a state after the successful invocation. Preconditions and effects are defined as statements in logic $\mathcal{L}(\Sigma)$.

Behavioral Semantics is a description of the public and the private behavior of a service. For our work we only use the public behavior (called choreography[1]) as a description of a protocol which must be followed by a client in order to invoke the service, fetch data from the service or perform a negotiation with a service. Thus, a service may contain different kinds of choreographies intended for different purposes. We describe a choreography as a protocol from the service point of view, i.e. all the messages are sent in to the service from the network and all the messages are sent from the service out to the network. We define the choreography X (read: chi) of the service using a state machine as

$$X = (\Sigma, L), \tag{3}$$

where $\Sigma \subseteq (\{x\} \cup C \cup R \cup E)$ is the signature of symbols, i.e. variable names $\{x\}$ or identifiers of elements from C, R, E of some information semantics O; and L is a set of rules. Further, we distinguish dynamic symbols denoted as Σ_I (input), and Σ_O (output) and static symbols denoted as Σ_S. While the static symbols cannot be changed by the service invocation, the dynamic symbols correspond to input and output data of the service which can be changed by the invocation. Each rule $r \in L$ defines a state transition $r : r^{cond} \rightarrow r^{\mathit{eff}}$ where $cond$ is defined as an expression in logic $\mathcal{L}(\Sigma_I \cup \Sigma_S)$ which must hold in a state before the transition is executed; *eff* is defined as an expression in logic $\mathcal{L}(\Sigma_I \cup \Sigma_O \cup \Sigma_S)$ describing how the state changes when the transition is executed.

Semantic descriptions enhance various descriptive parts of services. The parts regarding service invocation (such as how and where the service can be accessed) are reused, thus so called grounding from semantic level to the underlying technology for service invocation must be defined. In our approach we use grounding to WSDL which is used for on-the-wire message serialization (WSDL binding), physical Web service access (WSDL service and endpoint) and communication (SOAP). Using this grounding Σ_I and Σ_O are linked with the input and output messages respectively of an underlying WSDL interface operation. When the transition is executed the input data is transformed to the XML representing the input message (lowering) and passed to the underlying operation which responds with an output message in XML. The output message is then transformed to the corresponding output data (lifting).

In addition, we denote the description of the Web service and the goal as $\mathcal{W}$ and $\mathcal{G}$ respectively. For each such description, $\mathcal{D}$, we denote the information semantics as $\mathcal{D}_O$, the capability as $\mathcal{D}_F$, and choreography as $\mathcal{D}_X$.

[1]Please note, that our notion of the choreography is different from the one used by the Web Service Choreography Description Language (WSCDL). In WSMO terminology Choreography represents a stateful interface to the functionality offered by the Web service. `http://www.w3.org/TR/ws-cdl-10/`

3 Discovery with Data Fetching

Our main focus in this paper is the *service discovery* phase where dynamically obtained service instance definitions are matched against the user Goal and required communication with the service should be limited to fetching the data relevant from the Goal perspective omitting unnecessary information provided by the service in order to improve performance of the overall discovery process.

The matching is defined by the following set-theoretic relationships [10]: (1) exact match, (2) subsumption match, (3) plug-in match, (4) intersection match and (5) disjointness. If the goal and the Web service match, based on relationships 1-4, then service discovery is performed where it is checked if the service can satisfy the specifics of the service request, by consulting the data of the goal and the service. If all data is not available, it needs to be obtained from the service by performing so called *data fetching*. In this section we further elaborate on the service discovery phase and define the algorithm. For the service discovery we define the matching function:

$$s \leftarrow matching(\mathcal{G}, \mathcal{W}, \mathcal{B}_{gw}), \tag{4}$$

where $\mathcal{G}$ and $\mathcal{W}$ is a goal and a service description respectively and $\mathcal{B}_{gw}$ is a common knowledge base for the goal and the service. The knowledge base contains data which must be *directly* (through descriptions $\mathcal{G}_O$ and $\mathcal{W}_O$) or *indirectly* (through data fetching) available so that the matching function can be evaluated. The result s of this function can be: (1) $match$ when the match was found (in this case all required data in $\mathcal{B}_{gw}$ is available), (2) $nomatch$ when the match was not found (in this case all required data in $\mathcal{B}_{gw}$ is available), or (3) $nodata$ when some required data in $\mathcal{B}_{gw}$ is not available and thus the matching function cannot be evaluated.

We further assume that all required data for the goal is directly available in the description $\mathcal{G}_O$. The data fetching step is then performed for the service when the matching function cannot be evaluated (the result of this function is $nodata$). We then define the knowledge base as:

$$\mathcal{B}_{gw} = \mathcal{G}_O \cup \mathcal{W}_O \cup \{y_1, y_2, ..., y_m\}, \tag{5}$$

where $\{y_i\}$ is all additional data that needs to be fetched from the service in order to evaluate the matching function.

Further, we denote $\mathcal{W}_X$ as the data-fetch interface of the service $\mathcal{W}$ with output symbols Σ_O and input symbols Σ_I. The matching function can be then evaluated if data $\{y_i\}$ can be fetched from the service through the data fetch interface if input data Σ_I is either initially available in the knowledge base $\mathcal{B}_{gw}$ (data directly available from the goal or web service ontologies) or the input data becomes available during the processing of the interface.

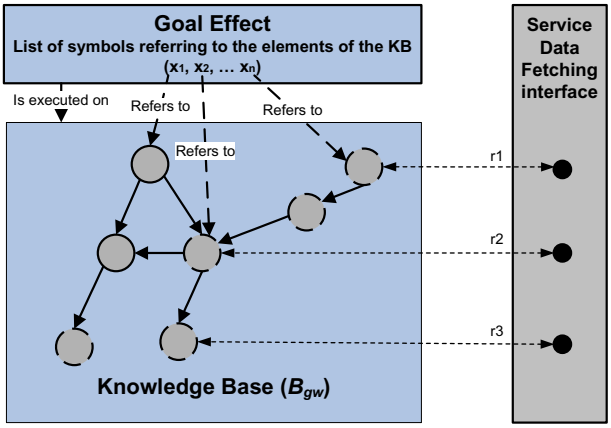

Figure 1. Minimization of the Provider Interactions

In addition, as illustrated in Figure 1, we only fetch the data from the interface if this data can be used for evaluation of the matching function (in general the data-fetch interface can provide data not required for the matching – see the rule `r3` in Figure 1). In $\mathcal{B}_{gw}$ full circles denote available information while dotted circles denote unavailable information which can be obtained through the data-fetch interface. However, since the fetching operation can be costly in terms of the generated communication, only the parts of $\mathcal{B}_{gw}$ which are referenced from the Goal effect ϕ^{eff} should be fetched.

Let ϕ^{eff} be the effect of the goal capability $\mathcal{G}_F$, L be the set of rules of the data-fetch interface $\mathcal{W}_X$, and let Σ_O be the set of output symbols of that interface. Then, we only use the rule $r \in L$ iff exists $x \in r^{eff}, x \in \Sigma_O$ such that $x \in \phi^{eff}$. Please note that this rule can be in addition executed if the input data is available during processing (i.e. r^{cond} holds in the $\mathcal{B}_{gw}$) (see the algorithm in Section 3.1).

3.1 Algorithm

In algorithm 1, the matching function is integrated with the data fetching which provides instance data for the concepts referred from the goal effect ϕ^{eff}. The algorithm operates on inputs, produces outputs and uses internal structures as follows:

Input:

- Web service W for which we denote W_O as the web service ontology with initial instance data and $\mathcal{W}_X$ as data-fetch interface of the Web service with rule base L. In addition, for each rule $r \in L$ we specify the data of the rule effect r^{eff} as $r.data$ and the action $r.action$ with values add, $update$, $delete$ meaning that if the rule is executed the action performs the effect of the rule, i.e. changing the state by adding, updating or deleting data in the memory (knowledge base).
- Goal description G for which we denote G_O as the goal ontology with initial instance data and G^{eff} as the goal capability effect. For W and G it must hold that they match at abstract level (Web service discovery).

Output:

- Boolean variable s indicating the result of the matching function between W and G, i.e. $match$ or $nomatch$.

Uses:

- Processing memory M containing data fetched during execution of rules of the data fetching interface.
- Knowledge base B_{gw} which contains data for processing of the matching function.
- Boolean variable $modified$ indicating whether the knowledge base has been modified or not during the processing.

Algorithm 1 Minimized Data Fetching for Discovery

1: $B_{gw} \leftarrow G_O \cup W_O$
2: $M \leftarrow B_{gw}$
3: **repeat**
4: $modified \leftarrow false$
5: $s \leftarrow matching(G, W, B_{gw})$
6: **if** $s = nodata$ **then**
7: **while** get r from L: $holds(r^{cond}, M)$ **and** $r.data \in G^{eff}$ **and not** $modified$ **do**
8: **if** $r.action = add$ **then**
9: $add(r.data, M)$
10: $add(r.data, B_{gw})$
11: $modified \leftarrow true$
12: **end if**
13: **if** $r.action = remove$ **then**
14: $remove(r.data, M)$
15: **end if**
16: **if** $r.action = update$ **then**
17: $update(r.data, M)$
18: $update(r.data, B_{gw})$
19: $modified \leftarrow true$
20: **end if**
21: **end while**
22: **end if**
23: **until** $s \neq nodata$ **or not** $modified$

The algorithm tries to fetch data from the service by processing the service's data-fetch interface. For each rule present, which can be executed, it checks whether its result will provide any information referenced by G^{eff}. For example G^{eff} may refer to the concept *price* of a given product which is unavailable in the B_{gw}, however a rule exists

which can result in an instance of the *price* concept being obtained. Once the data fetching operations are executed and new facts are added, updated or removed, a $modified$ flag is set to true and B_{gw} can be matched again. This cycle ends when no data can be fetched from the interface or the matching function can be evaluated (the result is $match$ or $nomatch$).

The algorithm assumes that the rules of the data-fetch interface can be executed independently. In particular this means that if there is a symbol referencing a concept in the knowledge base and there is a rule which can fetch the data for that concept, there is no other rule which needs to be executed prior in order to execute the rule fetching the data. Although our assumption that more realistic scenarios of data fetching should have independent rules (see Section 4.2), we acknowledge that this is an open issue of our approach which we plan to investigate in our future work.

The algorithm uses independent memory (memory M) from the knowledge base (B_{gw}) for processing the data-fetch interface. This allows that already-obtained data cannot be removed from the knowledge base while, at the same time, correct processing of the interface is ensured. The memory M is used not only for data but also for control of interface processing (in general, the content of the memory does not need to always reflect the content of the knowledge base). According to the particular interface definition, the data can be fetched step-wise allowing minimization of the interactions with the service during discovery. This maintains decoupling between elements as services are described semantically and independently from user requests. For example, during the service-creation phase a service provider (creator) does not know which particular data will be required for a particular data-fetch (in general, matching with a goal could require some or all defined data which depends on the definition of the request). The interface defined using rules allows to get only the data which is needed for the matching (for example in some cases only price is needed, in other cases a price and location of the selling company could be needed, if offered by the service depending on what is referred in the user request).

4 Implementation and Evaluation

In this section we describe our implementation for service discovery with data-fetching using Web Service Modeling Ontology (WSMO) [14] as a conceptual framework and we use Web Service Execution Environment (WSMX) [7] for implementation and execution of service discovery with data-fetch support. We detail the required modeling steps and explain the overall WSMX-based discovery process. Our service discovery model introduced in this paper has been implemented and evaluated through the Semantic Web Services Challenge [2]) discovery scenarios. There were two scenarios introduced. The first was related to package shipment where five different shippers offer various purchasing and shipment options. They provide different availability and pricing for their services with constraints on package destination, weight, dimension and shipment date where not all information can be statically provided. The second SWS-Challenge discovery use-case tackles product provisioning where different vendors provide PC hardware where their stock and prices change very often. It also involves simple composition since sometimes only a combination of the devices from different vendors can satisfy user requests and constraints. We have comprehensively addressed both scenarios and provided Web services proven to be a suitable testbed for evaluating our model since not all information could be provided in service descriptions meaning they had to be dynamically obtained at discovery-time.

[2]http://www.sws-challenge.org

4.1 WSMO, WSML, WSMX

WSMO provides a conceptual model and a language for semantic markup describing all relevant aspects of general services which are accessible through a Web service interface. The ultimate goal of such markup is to enable the (total or partial) automation of tasks (e.g. discovery, selection, composition, mediation, execution, monitoring, etc.) involved in both intra- and inter-enterprise integration settings. WSMO defines the underlying model for the WSMX Semantic Web services architecture and execution environment and provides the conceptual model formalised by the Web Service Modeling Language (WSML)[1] family of ontology languages, used to formally describe WSMO elements. Thus, WSMO, WSML and WSMX form a coherent framework covering all aspects of the Semantic Web services.

Both the descriptions of WSMO Goals and Web Services include elements for describing *capabilities*. We use the following parts of WSMO Capabilities in our service discovery:

- **Preconditions** describe conditions which must hold in a state required before the service can be executed. WSMO Preconditions map to ϕ^{pre} of the capability descriptions as defined in Section 2.
- **Postconditions** describe conditions in a state which must hold after the service is executed. WSMO Postconditions map to ϕ^{eff} of the capability descriptions as defined in Section 2.

From the perspective of a goal description, the capability describes the functionality that the owner of the goal wishes to achieve from a Web service. Correspondingly, the capability of a Web service describes the functionality offered by that service. To a large extent, the responsibility of a discovery mechanism, in the context of WSMO, is to find services whose capability matches that of the provided goal.

In addition, the Web service interface defines *choreography* and *orchestration* allowing the modeling of external and internal behavior of the service respectively. We define the interface for data-fetch using a specific choreography *namespace*[3] allowing to distinguish a specific meaning for its usage from the meaning of the interface defining execution choreography used for consuming the service functionality within the same WSMO service.

4.2 WSDL to Choreography Mapping

The modeling of Semantic Web service behavioral descriptions is, to a significant extent, based on existing Web service standards. We map existing WSDL service descriptions to the WSMO Semantic Web services where additional descriptions can be provided. Mapping from existing, syntactic service descriptions to the semantic layer is the first step of the modeling process after which resulting descriptions can be aligned by the domain expert.

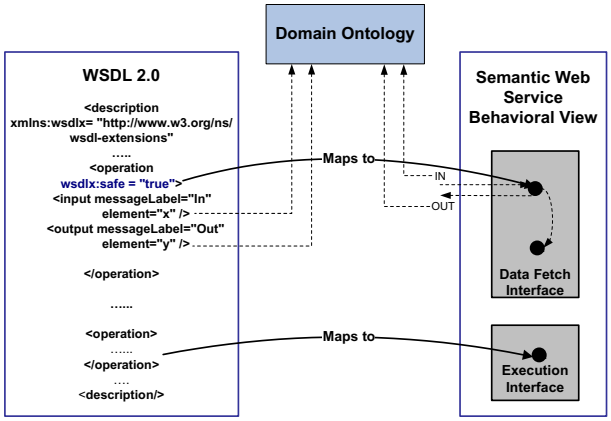

Figure 2. WSDL 2.0 to WSMO Web service Mapping

XML Schema defined in the WSDL can be mapped to the given domain ontology using Semantic Annotations of Web Service Description Language and XML Schema (SAWSDL [4]) which provides a generic and agnostic mechanism for semantically annotating Web services. As described in [16], the SAWSDL allows to annotate WSDL schema elements with elements from the information semantics and WSDL interfaces with behavioral semantics of the service. In addition, the extension of WSDL 2.0 comes with the very relevant notion of so called *safe methods*, relevant from the dynamic discovery point of view. When the *safe* attribute of an operation is set to *true*, the operation indicates that it is informative, independent on other operations and will not cause any real world effect when invoked (e.g. like agreeing to buy something). For the purpose of our work, safe operations may be mapped to the data-fetch interface allowing a user to find out more about the functionality of the service. Figure 2 presents this mapping. WSDL operations without the *safe* attribute are mapped to the service execution interface.

[3] we specify the URI for the namespace as *"http://wsmx.org/datafetch#"*

4.3 Modeling Ontologies, Goals and Services

We base examples on a simple composition service for computer hardware, where PC hardware stock and price information is not available in the service description and needs to be fetched during the service discovery. We also emphasize how this communication is minimized by looking at the concepts referred to in the goal capability. In section 4.4 we further describe the evaluation of our implementation in the broader context of the SWS Challenge requirements. In order to implement the scenario, we first need to create semantic models for ontologies, goals and services. We describe these models in the following subsections. We present examples of ontologies, services and goals definitions in WSML using the following prefixes to denote their respective namespaces: do – domain ontology, df – data fetch interface, gl – goal ontology.

4.3.1 Ontologies

Ontologies provide rich data models used for the definition of goals and services. In our scenario we use a common domain ontology with additional ontologies to define specific axioms or concepts used by the descriptions of services and/or goals.

The common ontology defines shared concepts used in the description of the goal and services, such as $Location$, $Notebook$, $DockingStation$, etc. In addition, we use the common ontology to specify named relations for services and goals. Specific ontologies for goals and services declare axioms that define the relations to represent their conditions. An analogy for this approach are interfaces in programming languages like Java. The interface declares some functionality but does not say how this should be implemented. Using this approach, we define a set of relations in the common ontology which represent the axioms that a service may need to define. Listing 1 shows the simple definition for the $isCompatible$ relation from the common ontology and its implementation in the service ontology.

```
1  /* isCompatible relation in the domain ontology */
2  relation do#isCompatible (ofType do#Notebook, ofType do#
       DockingStation)
3
4  /* implementation of the isCompatible relation in the service ontology */
5  axiom isCompatibleDef definedBy
6     (?notebook[do#GTIN hasValue ?gtinX] memberOf do#Notebook and
7     ?dockingstation[do#supportsGTIN hasValue ?gtinY]
8     memberOf do#DockingStation and
9     ?gtinX = ?gtinY implies
10    do#isCompatible(?notebook, ?dockingstation).
```

Listing 1. $isCompatible$ **relation**

The relation $isCompatible$ is true if the notebook sold by the service provider can be used with one of the available ($DockingStation$). This axiom can be used in the goal query to check compatibility of the two components.

4.3.2 Services

We focus on the description of the data-fetch interface of one of the vendors service showing how and which data can be fetched during discovery.

```
1  stateSignature WSVendorStatesignature
2
3      in do#NotebookListReq withGrounding { _"http://sws−challenge.org
           /vendor.wsdl#(VendorPort/listNotebooks/in0)"}
4      in do#WebCamListReq withGrounding { _"http://sws−challenge.org
           //vendor.wsdl#(VendorPort/listWebCams/in0)"}
5      out do#NotebookList
6      out do#WebCamList
7
8  interface df#WSVendorDataFetchInterface
9    choreography WSVendorDataFetchChoreography
10     ...
11     transitionRules WSVendorDataFetchTransitionRules
12       /* Rule 1: Request for the list of notebooks */
13       forall {?notebookListReq} with (
14         ?notebookListReq[mo#location hasValue ?clientlocation]
15                 memberOf do#NotebookListReq and
16                 ?clientLocation memberOf mo#Location and
17                 mo#isAvailable(?clientLocation)
18       ) do
19         add(_# memberOf do#NotebookList)
20       endForall
21
22       /* Rule 2: Request for the list of Web cameras*/
23       forall {?webcamsListReq} with (
24         ?WebCamsListReq memberOf do#WebCamListReq
25       ) do
26         add(_# memberOf do#WebCamList)
27       endForall
```

Listing 2. Vendor data fetching interface

In listing 2, the first rule (line 14) describes how to get the list of notebook prices depending on the user location. A user-location ($location$ variable) is taken from the notebook quote request. For the listing of notebooks the location matters and first the location of the client has to be checked. If notebooks are not shipped to the client location, no data will be fetched, since the client will not be able to buy from this vendor due to the address constraints. For example, notebooks may be sold only in US for tax and shipment reasons while other, lightweight items (e.g. Web cameras) may be shipped all over the world. In the definition of the second rule there are no constraints over the client's location and available Web cameras may be sold to any location in the world. Concepts $NotebookListReq$, $WebCamsListReq$ and $NotebookList$, $WebCamList$ are defined as input and output vocabularies respectively. The relation $isAvailable$ is described in the common ontology and its axiom is provided in the Web service ontology.

4.3.3 Goals

The example goal for the scenario describes the user's aim to buy a laptop and docking station and to ship them to a specific location. In addition, the goal specifies a preference that price be used for selection of the best service where multiple matching services are discovered.

```
1  Goal GoalPurchaseHardware
2    nfp
3      _"preference" hasValue "?price"
4    ...
5    endnfp
6    ...
7    capability GoalPurchaseHardwareCapability
8  postcondition
9    definedBy
10     ( ?x[do#price hasValue ?priceX, do#hddGB hasValue ?hddGBX
           , do#memoryMB hasValue ?memMBX]
11       memberOf do#MacNotebook and ?memMBX >= 512 and
             ?hddGBX > 40 and
12     ?y[do#price hasValue ?priceY] memberOf do#DockingStation
13       and isCompatible(?x,?y)
14       and ?price = (?priceX + ?priceY)
15       and ?price < 2000).
16 ...
```

Listing 3. User Goal in WSMO

The goal as in listing 3 is defined for our scenario with respect to the implementation of the matching function from section 3 (we discuss this implementation in section 4.4). The goal expression contains references to two concepts, namely mo#Notebook and mo#DockingStation. Instances of these two concepts are not available in the static Web service description and have to be fetched during the discovery phase as specified in algorithm 1. The goal defines the capability postcondition specifying how to get a quote for the product while at the same time the product must be available to be shipped to the location specified by the notebook request. Hard constraints are expressed over the parameters of the laptop. It must have at least 512 MB RAM and over 40 GB hard drive capacity. Additionally, a compatible docking station should be ordered. The previously-defined axiom $isCompatible$ is used for this purpose. The overall price of both components should not exceed 2000 Euro.

4.4 Implementation

The scenario is implemented as follows: when the goal is generated out of the request specified by the user, it is sent to the WSMX system. WSMX starts a new operational thread (execution semantics) which first invokes the discovery component which in turn returns a list of services matching the goal. This list is passed to the selection component to select the service that best fits the user request. Control passes to the choreography engine which uses the choreography descriptions of the goal and service respectively, to drive the message exchange with the discovered service. This section describes the implementation of the algorithm from section 3 within the discovery component of WSMX. The details about other parts of the execution process can be found in our previous work in [7].

After the discovery phase, the execution semantics starts the conversation by processing the execution choreographies of the goal and selected service resulting in invoking and consuming of the service capability by the user.

Section 3 describes two steps for discovery. A prototype for the *Web service discovery* is under development in the WSMO working group. The implementation, described here, focuses on the steps of *Service discovery* matching and *data-fetching*. A match between the goal and Web services is determined on the knowledge base created out of their descriptions, including instance data (both available from the descriptions and fetched). The goal capability defines a query (listing 3) which is used to query the knowledge base.

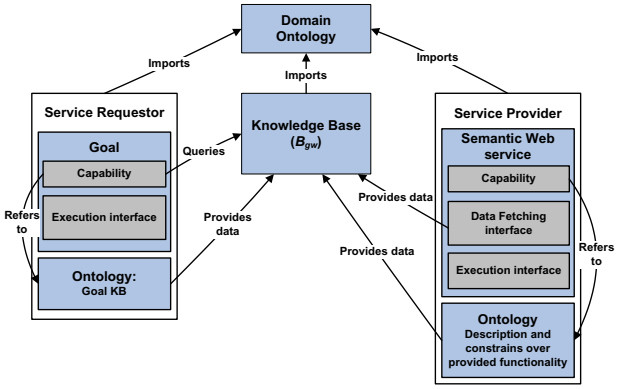

Figure 3. Knowledge Base B_{gw}

According to the algorithm 1 in section 3, the knowledge base B_{gw} is created for every goal and Web service from the repository as shown in figure 3. Initially, the knowledge base imports all concepts from the domain ontology and data from both goal and Web service descriptions. If the data-fetch interface is available then parts of it may be able to obtain the instance data of some of the concepts referred to in the goal query. In our case, data fetching will be executed for $Notebook$ and $DockingStation$ provisioning. Once the knowledge base is populated with up-to-date information on available notebooks and docking stations, a query such as the one defined in listing 3, can be performed on the KB. It is worth noting that, thanks to this approach, only the necessary parts of the data-fetch interface will be utilized and no unnecessary communication will be generated. If the result of the evaluation is $true$, we add the Web service to the list E of Web services to the position determined by the preference. If the result of the evaluation is $false$, match of the next service from the list is attempted. Otherwise, the cycle ends and the next service from the repository is processed. We briefly discuss this implementation in the next section 4.5.

4.5 Evaluation

Our implementation has been evaluated according to the methodology defined by the SWS Challenge. The SWS Challenge is an initiative led by a Semantic Web Services community providing a standard set of increasingly difficult problems, based on industrial specifications and requirements. Entrants to the SWS Challenge are peer-evaluated to determine if semantically-enabled integration approaches reduce costs of establishing and maintaining the integration between independent systems. In each SWS challenge workshop, the entrants first address an initial scenario of a particular problem (e.g. mediation, discovery) in a testing environment prepared by the SWS Challenge organizers. The organizers then introduce some changes to back-end systems of the testing environment when the adaptivity of solutions is evaluated – solutions should handle introduced changes by modification of declarative descriptions rather than code changes. This evaluation is done by a methodology, developed by the SWS Challenge organizers and participants, which identifies following so called *success levels*. *Success level 0* indicates a minimal satisfiability level, where messages between the participant solutions and the backend systems are properly exchanged in the initial scenario. *Success level 1* is assigned when changes introduced in the scenario require code modifications and recompilation. *Success level 2* indicates that introduced changes did not entail any code modifications but only declarative parts had to be modified. *Success level 3* is assigned when changes did not require either modifications to code or the declarative parts, and the system was able to automatically adapt to the new conditions.

Our implementation was evaluated to successfully address the scenario with dynamic PC hardware product data fetching, where various constraints over the hardware parameters (like HDD capacity, type and speed of the processor, etc.), customer location and overall price had to be considered, scoring success level 2. The implementation proved to be generic as only modifications of the WSMO goals were necessary in order to correctly handle introduced changes. No changes in WSMX code or in the descriptions of the services were required – only the goal requests had to be changed.

In the initial version of our work it was not possible to distinguish between dynamically fetched data required for the user request evaluation and data which is irrelevant resulting from fetching from all *safe* endpoints exposed via data-fetch interface. The introduction of our optimization allowed to fetch only the relevant data which significantly decreased communication with the service especially since multiple service endpoints were provided for fetching information on different kinds of products. For example for the goal of buying a laptop with compatible docking station only information on these two products were fetched leaving out all other unreferenced, though available, information.

5 Related Work

Our work can be compared to other solutions of SWS-Challenge discovery scenario as described in [12]. The University of Jena and University of Milano solutions provide an ad-hoc, hard-coded mechanisms for data fetching whereas our approach aims at generic solution considering performance factors. Kifer et al. [11] point out the need for a contracting phase in service discovery and include it in their proof obligation for service matching. However, they do not consider an additional interface for fetching missing information dynamically. There is no directly comparable work in the SOA area which would allow for the fetching of additional data to aid discovery at run-time. WSDL 2.0 safe methods could be seen as a step in the direction of enabling Web services for a dynamic discovery process leading to fully-fledged service late-binding. In his W3C position paper [3], Dziembowski highlights the need for instance-based Web servive discovery but with a very limited proposal on how this should be achieved.

The problem of insufficient static descriptions for fine-grained discovery requests within dynamic domains has been addressed using CORBA's *Trading Service* in the eMarketplace domain [15] where dynamic properties of a offered functionality exposed as a CORBA object can be calculated within a business's private space and where results can be integrated into the *Trading Service* discovery process. *Trading Service* consists of name-value property pairs which can be either static or dynamic. In the case of a dynamic property, the external *Dynamic Property Evaluator* entity residing within the business provider private space is called. *Dynamic Property Evaluator* returns different results depending on the current state of the business provider (e.g., its current stock, prices or date and time of the day) and client's request. It is worth noting that due to the code-based representation of CORBA actual evaluation of the *Trading Service* is carried out in the client's code while our semantic-based approach allows to shift the overall discovery process to the middleware requiring the client only to provide a fine-grained request. A similar approach to dynamic service functionality aspects is currently missing in SOA and our work attempts to fill this gap.

Research into goal-based discovery for WSMO and WSMX takes a step-wise approach with both theoretical and implementation concerns addressed at each stage. Three strategies have been investigated in this manner. The first is keyword-based discovery [10], which uses an algorithm that matches keywords, identified from the goal description, with words occurring in various parts of the Web service description, namely in, non-functional properties, concept names and predicate names. The second strategy is for a lightweight Semantic Web services discovery described in [6]. This approach models a service in terms of the objects it can deliver. The term object, in this sense, means something of value the service delivers in its domain of interest. A third strategy is based around the use of quality-of-service attributes as described in [8]. Upper level ontologies describing various domains for quality-of-service attributes are provided and non-functional properties are introduced to the service descriptions whose meanings are defined in these QoS ontologies. The approach in this paper is compatible with each of the matching strategies as it extends the matching power by requesting data from the service that is not directly available in its description.

In some extend our work on can be compared to Web service Service Level Agreement (SLA) [9] and service contracting. Web Service SLA are also often a subject of the communication before execution phase between requester and provider since SLA depend on the request, and states of both requester and provider. SLA operate on the Web service technical level where various parameters like Web service availability, reliability, response time dynamically change. Our data fetching mechanism during the discovery phase pertains to the business service view where the actual offer can be dynamically obtained and used by the discovery engine to evaluate suitability of the service against specified request. With respect to service contracting our approach can be considered as pre-contracting as we concentrate on the retrieval of additional data from the service provider to make a more exact match during discovery.

6 Conclusion and Future Work

In this paper we have presented an approach for semantic discovery supporting realistic late-binding performed at run-time with an emphasis on minimizing required communication. The fine granularity of both client request and service functionality descriptions, expressed semantically, allows the shifting of the final decision on which service to interact with, from the client side to semantic middleware where required service functionality details are matched against client request. We are aware that our approach, due to the logical reasoning, computational complexity and generated communication overhead, is scalable only for a limited number of services, therefore data fetching and detailed evaluation takes place in the final phase of the discovery process, which is preceded by category-based matchmaking and static description semantic discovery. Applied optimization to dynamic data-fetching decreases the generated communication, especially for more complex services offering a broad range of functionality (e.g. a warehouse offering multitude of products with different constraints on shipment and different pricing options). The combination of semantics with the dynamic data-fetch mechanism brings significant benefits to the runtime service late-binding, facilitating the volatile and frequently-changing nature of services in SOA.

For future work, we plan to extend our discovery frame-

work in two directions: (1) support for service contracting, (2) evaluation of our work with large number of services. Service functionality details disclosed during the discovery phase should be propagated downwards to the service execution phase once the client (or middleware acting on the behalf of client) decide to consume the dynamically proposed functionality. It is worth noticing that data fetched during the discovery can be used for constructing a client-service contract with a certain time period of validity. However, an additional protocol would be required for finalizing such a contract, which may also require negotiation with the service. Our discovery framework has been successfully applied to the SWS-Challenge discovery problems where several Web services have been semantically described and their safe operations been exposed via data-fetch interfaces. We plan to evaluate the applicability of our approach to a greater number of services from different domains, to examine the scalability of the advantages obtained through using semantic descriptions with the data-fetch mechanism, over purely informal and static service descriptions.

Acknowledgments

This work is supported by the Science Foundation Ireland Grant No. SFI/02/CE1/I131, and the EU projects Knowledge Web (FP6-507482), SemanticGov (FP-027517) and SUPER (FP6-026850).

References

[1] J. de Bruijn, H. Lausen, A. Polleres, and D. Fensel. The web service modeling language: An overview. In *Proc. of the European Semantic Web Conference*, 2006.

[2] A. Dogac, Y. Tambag, P. Pembecioglu, S. Pektas, G. Laleci, G. Kurt, S. Toprak, and Y. Kabak. An ebXML infrastructure implementation through UDDI registries and RosettaNet PIPs. In *Proc. of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 512–523, 2002.

[3] K. Dziembowski. Dynamic Service Discovery. In *A position paper for the W3C Workshop on Web Services for Enterprise Computing, available at http://www.w3.org/2007/01/wos-papers/gestalt*, 2007.

[4] J. Farrell and H. Lausen. Semantic Annotations for WSDL and XML Schema available at http://www.w3.org/TR/sawsdl/. Technical report, 2007.

[5] D. Fensel, U. Keller, H. Lausen, A. Polleres, and I. Toma. What is wrong with Web services Discovery. In *W3C Workshop on Frameworks for Semantics in Web Services*, Innsbruck, Austria, June 2005.

[6] A. Friesen and S. Grimm. DIP WP4 Service Usage, D4.8 Discovery Specification, available at http://dip.semanticweb.org/documents/D4.8Final.pdf. Technical report, 2005.

[7] T. Haselwanter, P. Kotinurmi, M. Moran, T. Vitvar, and M. Zaremba. WSMX: A Semantic Service Oriented Middleware for B2B Integration. In *ICSOC*, pages 477–483, 2006.

[8] M. Hauswirth, F. Porto, and L.-H. Vu. P2P and QoS-enabled Service Discovery Specification, available at http:/dip.semanticweb.org/documents/D4.17-Revised.pdf. Technical report, 2006.

[9] L. jie Jin, V. Machiraju, and A. Sahai. Analysis on Service Level Agreement of Web Services. Technical report, HP Laboratories Palo Alto, 2002.

[10] U. Keller, R. Lara, H. Lausen, A. Polleres, L. Predoiu, and I. Toma. WSMO D10.2 Sematic Web Service Discovery available at http://www.wsmo.org/TR/d10/v0.2/d10.pdf. Technical report, 2005.

[11] M. Kifer, R. Lara, A. Polleres, C. Zhao, U. Keller, H. Lausen, and D. Fensel. A Logical Framework for Web Service Discovery. In *ISWC 2004 Workshop on Semantic Web Services: Preparing to Meet the World of Business Applications, CEUR Workshop Proceedings*, volume 119, Hiroshima, Japan, 11 2004.

[12] U. Kster, A. Turati, M. Zaremba, B. Knig-Ries, D. Cerizza, E. D. Valle, M. Brambilla, S. Ceri, F. Facca, and C. Tziviskou. Service Discovery with SWE-ET and DIANE - A Comparative Evaluation By Means of Solutions to a Common Scenario. In *9th International Conference on Enterprise Information Systems (ICEIS2007)*, Funchal, Madeira-Portugal, June 2007.

[13] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *1st International Semantic Web Conference (ISWC)*, pages 333–347, 2002.

[14] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web Service Modeling Ontology. *Applied Ontologies*, 1(1):77 – 106, 2005.

[15] A. Schade, C. Facciorusso, S. Field, and Y. Hoffner. Advanced Dynamic Property Evaluation for CORBA-Based Electronic Markets. In *Second International Workshop on Advanced issues of E-Commerce and Web-Based Information Systems*, pages 109–116, 2000.

[16] T. Vitvar, J. Kopecky, and D. Fensel. WSMO-Lite: Lightweight Semantic Descriptions for Services on the Web. WSMO Working Draft v0.2, DERI, 2007. Available at: `http://www.wsmo.org/TR/d11/v0.2/`.

[17] T. Vitvar, A. Mocan, M. Kerrigan, M. Zaremba, M. Zaremba, M. Moran, E. Cimpian, T. Haselwanter, and D. Fensel. Semantically-enabled service oriented architecture : concepts, technology and application. *Service Oriented Computing and Applications*, 2(2):129–154, 2007.

[18] T. Vitvar, M. Zaremba, and M. Moran. Dynamic Service Discovery through Meta-Interactions with Service Providers. In *Proceedings of the 4th European Semantic Web Conference (ESWC 2006)*, June 2007.

[19] M. Voskob. UDDI Spec TC V4 Requirement - Taxonomy support for semantics. OASIS, 2004. http://www.oasis-open.org. Technical report.