Specification of Invocable Semantic Web Resources

Sudhir Agarwal
Institute of Applied Informatics and Formal Description Methods (AIFB),
University of Karlsruhe,Germany
agarwal@aifb.uni-karlsruhe.de

Abstract

In this paper, we present a novel approach for specification of invocable semantic web resources in a unified manner. Our approach enables specification of web services, composite web services, web pages and web sites with one single process description language in a unified manner. The basic idea behind our work is, that different types of invocable web resources differ mainly in the ommunication protocol and message formats. The language abstracts from various communication protocols and message formats and hence enables modeling of business logic without worrying much about technical details. Our approach in contrast to existing annotation techniques does not enforce the providers to modify their web resources.

1. Introduction

The web is no more a mere source of information. It has evolved as the most important medium for offering business. Businesses can be conducted completely electronically, for example, booking/buying a train ticket or semi-electronically, for example buying a book[1] or non-electronically, for example, getting a hair cut in a hair cutting saloon. Further, functionalities are offered as programming interfaces, that can be embedded in a software and executed automatically. Such interfaces are called web services, if they are programming language and architecture independent and are based on standards. Consequently, users (humans or machines) search not only for information but also for services that are offered either directly on the web or in the real world or in the mixed environement.

With web pages we mean typical HTML pages that exist independent of other web pages. Current web services are mostly used as platform independent and programming language independent programming interfaces or remote precedures. Web pages and web services are similar except that the former is for human consumption and latter for machines. Hence, they use different languages (HTML and SOAP message) to carry information. Web sites or web portals are collection of web pages. Such a collection may or may not be implemented to support a business process. In case, it is, for example www.amazon.com, we have the situation, that the web pages in the web site are not completely independent of each other, but occur in some particular order depending on the actions that a user performs or the values that a user enters on the pages of the web site. For example, on a page if the user chooses credit card as the payment method and clicks on continue button, the next page that occurs is different than the one that occurs if the user chooses invoice as the payment method and clicks on continue button.

Since in the current web there is no equivalent for machines that offer similar functionality as web sites, current efforts in the web developement include introduction of composite web services. In figure 1, the upper right corner shows this missing part that will be introduced in the new future to complete the picture.

Like current web services are similar to web pages, composite web services are similar to web sites. Like web sites, composite web services are not one step proceses like a web service or the generation of a web page, but support mutliple user interactions and their behaviour depends on the actions and values received by the user.

Web sites generalize web pages, because a web page can be seen as a web site consisting of only one page and composite web services generalize web services because a web service can be seen as a composite web service consisting of only web service (refer to figure 1).

1.1. Current Semantic Web Efforts

The vision of the semantic web is to give machine understandable meaning to the resources available on

1 semi-electronically, because the book is not sent via internet, but with snail mail

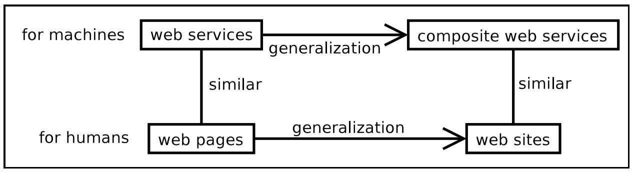

Figure 1. Invocable resources - today and future

the web [1, 2, 3]. Semantic web technologies like ontologies play an important role here.

There are two almost disjunct groups of researchers, one considering web services and composite web services and the other the web pages. Even if web services are very similar to web pages, there is hardly any unifying approach that allows to give semantics to web services as well as web pages.

Further, current efforts in the semantic web research community address only web services, web pages and composite web services. They do not handle web sites (cf. lower right corner of figure 1). Current annotation efforts consider a web page independent of other web pages. The annotation process results in a modified web page containing the domain ontology inside a comment block of the web page. Further, an element of the web page can be connected to the ontology by adding the URI of the relevant ontological entity (concept, relationships, properties, attributes).

Apart from the fact that such an approach enforces the providers of web pages to modify their web pages, it does not consider web pages that do not exist independent of each other but occur in a particular order and contain particular information that depends on the actions performed by and the values received by the user [4].

1.2. Our Approach

Technically, semantic web can be seen as a large collection of processes. These processes performs computations and interact with users. Any inputs (object values or actions) and outputs (information or result) are received and delivered by communicating/interacting with the user. Note, that the information that a process delivers can be about anything, even a description of a process that can not be executed in the web. The basic difference among electronic, semi-electronic or non-electronic processes is, that they communicate with the user via different protocols.

Considering this insight, we conclude that a semantic web process description language that offers an explicit communication model can be used to specify the contents and processes in the semantic web. The communication model must be able to abstract from different communication protocols, for example HTTP, SOAP, phone, fax, snail mail etc. and message formats, for example, HTML, SOAP messages, voice, paper letter etc. Such a language will allow to specify the actual business logic without worrying about the technical aspects. Such a specification can then be mapped to concrete protocols and message formats that the process should support.

```
<owl:Class rdf:ID="Process">
 <rdfs:subClassOf rdf:resource="&owl:#Class"/>
</owl:Class>

<owl:Class rdf:ID="ProcessItem">
 <rdfs:subClassOf rdf:resource="&owl:#Class"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="processItem">
 <rdfs:domain rdf:resource="#Process"/>
 <rdfs:range rdf:resource="#ProcessItem"/>
</owl:ObjectProperty>
```

Figure 2. Process and Process Items

The structure of our paper reflects the above idea and procedure. In section 2, we introduce a semantic web process desctiption language, in short SWPDL. In section 3, we show how web pages, web sites, web services and composite web services can be specified with SWPDL. In section 4, we present some evaluation results. We compare our work with other works in section 5 and finally conclude in section 6 by listing some benefits of our work from the end user's point of view as well as provider's point of view.

2. Semantic Web Process Description Language

A process is a business offered by an actor, called provider of the the process, to a set of actors, called users of the process. We say, a user uses a process, when he avails himself of a process by invoking a process. We start by modeling a class `Process`. Each instance of the class `Process` represents a concrete process (business logic). Further, each instance of a concrete process represents a concrete run of the concrete process (cf. figure 2).

An instance of a concrete process operates on objects, which can be data objects or real world objects. Thus a concrete process has a set of classes, that take part (instantiated) during an execution of the process. We model class `ProcessItem` and a property `processItem` for this purpose (cf. figure 2).

When a concrete process runs it can perform different activities. It may perform some computational tasks locally, communicate with the user or call other

```
<owl:Class rdf:ID="Activity">
 <rdfs:subClassOf rdf:resource="&owl:#Class"/>
</owl:Class>

<owl:Class rdf:ID="LocalOperationActivity">
 <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>

<owl:Class rdf:ID="CommunicationActivity">
 <rdfs:subClassOf rdf:resource="#Activity"/>
/owl:Class>

<owl:Class rdf:ID="ProcessActivity">
 <rdfs:subClassOf rdf:resource="#Activity"/>
</owl:Class>
```

Figure 3. Process Activities

processes. Hence, we identify following three types of activities that a process can perform [5, 6].

1. local operation activity
2. communication activity
3. process activity

Again, all these activities are classes in a concrete process, which are instantiated when the process is instantiated. We model classes `Activity` that represents all types of activities and its sublasses `LocalOperationActivity`, `CommunicationActivity` and `ProcessActivity` (cf. figure 3). Let O, C and W be the sets of local operation activities, communication activities and web process activities respectively of a process.

The order of execution of various activities is controlled by the control contstructs. We currently consider three types of control constructs *sequence*, *parallel* and *choice*. Each control construct has a set of components and certain control construct specific properties which we will discuss below. A control construct is instantiated when a concrete process is executed. We model class `ControlConstruct` and classes `Sequence`, `Parallel` and `Choice` as its subclasses (cf. figure 4). Let us denote the set of control constructs by S.

With a process component we mean either an activity or a control construct. We model a class `ProcessComponent` as union of class `Activity` and class `ControlContstuct` (cf. figures 5, 3 and 4). Let us denote the set of components by P, that is, $P = S \cup O \cup C \cup W$. The order of execution of the various components is controlled by exactly one control construct. Hence we model a property `mainControlConstruct` with domain `Process` and range `ControlConstruct` (cf. figures 4, 5).

Now, we consider various activities in more detail. A *local operation activity* $o \in O$ has a set of input parameters, a set of output parameters and a grounding.

```
<owl:Class rdf:ID="ControlConstruct">
 <rdfs:subClassOf rdf:resource="&owl:#Class"/>
</owl:Class>

<owl:Class rdf:ID="Sequence">
 <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</owl:Class>

<owl:Class rdf:ID="Parallel">
 <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</owl:Class>

<owl:Class rdf:ID="Choice">
 <rdfs:subClassOf rdf:resource="#ControlConstruct"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="component">
  <rdfs:domain rdf:resource="#ControlConstruct"/>
  <rdfs:range rdf:resource="#ProcessComponent"/>
</owl:ObjectProperty>
```

Figure 4. Control Constructs

```
<owl:Class rdf:ID="ProcessComponent">
  <owl:unionOf rdf:parseType="Collection">
    <owl:Class rdf:about="#Activity"/>
    <owl:Class rdf:about="#ControlConstruct"/>
  </owl:unionOf>
</owl:Class>

<owl:ObjectProperty rdf:ID="mainControlConstruct">
  <rdfs:domain rdf:resource="#Process"/>
  <rdfs:range rdf:resource="#ControlConstruct"/>
</owl:ObjectProperty>
```

Figure 5. Components and Main Control

Grounding connects an operation with its concrete implementation for example a Java method or another web service. When a local operation is executed its corresponding implementation is invoked with the set of input parameters as described in the operation activity (cf. figure 6).

To describe a communication activity the notion of a *message* is necessary. A *message* is an object that carries a set of objects [7]. With a *communication activity* an actor (sender) sends a message to another actor (receiver). The set of objects that a message carries is categorized into two categeries, namely send and receive. The objects in categorized as send objects are the objects that sender of the message wants to deliver to the receiver of the message and the those categorized as receive are objects, the sender expects to receivers values for from the receiver. Further, a message has a grounding, which is mapping between a message and its concrete implementation, for example as HTML or SOAP message. We model a class `Message` with properties `send`, `receive` and `grounding`. Further, we model a class `MessageGrounding` (cf. figure 7).

After a message is sent, the communication activity

```
<owl:ObjectProperty rdf:ID="input">
  <rdfs:domain rdf:resource="#LocalOperationActivity"/>
  <rdfs:range rdf:resource="&owl:#Class"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="output">
  <rdfs:domain rdf:resource="#LocalOperationActivity"/>
  <rdfs:range rdf:resource="&owl:#Class"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="LocalOperationGrounding">
 <rdfs:subClassOf rdf:resource="&owl:#Class"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="grounding">
  <rdfs:domain rdf:resource="#LocalOperationActivity"/>
  <rdfs:range rdf:resource="#LocalOperationGrounding"/>
</owl:ObjectProperty>
```

Figure 6. Local Operation Activity

```
<owl:Class rdf:ID="Message">
</owl:Class>

<owl:ObjectProperty rdf:ID="send">
  <rdfs:domain rdf:resource="#Message"/>
  <rdfs:range rdf:resource="#ProcessItem"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="receive">
  <rdfs:domain rdf:resource="#Message"/>
  <rdfs:range rdf:resource="#ProcessItem"/>
</owl:ObjectProperty>

<owl:Class rdf:ID="MessageGrounding">
 <rdfs:subClassOf rdf:resource="&owl:#Class"/>
</owl:Class>

<owl:ObjectProperty rdf:ID="grounding">
  <rdfs:domain rdf:resource="#Message"/>
  <rdfs:range rdf:resource="#MessageGrounding"/>
</owl:ObjectProperty>
```

Figure 7. Message

waits until the corresponding response has arrived only if the message contained a non-empty set of receive objects. Otherwise, the execution of the communication activity ends with the sending of the message. In case the set of receive objects is non-empty, the execution of a communication activity is complete as soon as the corresponding response message from the receiver arrives. A *process activity* is the activity that is used to call a process from inside a process. When a process is executed its main control construct is executed (cf. figure 8).

Now we describe how various control constructs are executed (cf. figure 4). We start with the simplest control-construct, a *sequence*. A sequence is an ordered set of components. We denote a sequence of the activities p_1, p_2, …, p_n with $sequence(p1, p_2, \ldots, p_n)$ where for each $i \in \{1, \ldots, n\}$ $p_i \in P$. For a sequence s we denote the ordered set of its components with $s.P$. The

```
<owl:ObjectProperty rdf:ID="message">
  <rdfs:domain rdf:resource="#CommunicationActivity"/>
  <rdfs:range rdf:resource="#Message"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="process">
  <rdfs:domain rdf:resource="#ProcessActivity"/>
  <rdfs:range rdf:resource="#Process"/>
</owl:ObjectProperty>
```

Figure 8. Communication and Process Activities

```
<owl:Class rdf:ID="ComponentPosition">
</owl:Class>

<owl:ObjectProperty rdf:ID="component">
  <rdfs:domain rdf:resource="#ComponentPosition"/>
  <rdfs:range rdf:resource="#ProcessComponent"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="position">
  <rdfs:domain rdf:resource="#ComponentPosition"/>
  <rdfs:range rdf:resource="&rdfs:#Literal"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:ID="componentPosition">
  <rdfs:domain rdf:resource="#Sequence"/>
  <rdfs:range rdf:resource="#ComponentPosition"/>
</owl:ObjectProperty>
```

Figure 9. Control Construct Sequence

execution semantics of a sequence s is described recursively by the following rules

$$\begin{cases} \text{execute } p_1 \text{ then execute } \mathit{sequence}(p_2, \ldots, p_n) & \text{if } s.P \neq \emptyset \\ \text{do nothing} & \text{if } s.P = \emptyset \end{cases}$$

We handle the order of components by modeling a class `ComponentPosition` (cf. figure 9).

Now we describe the control construct *parallel*. Like a sequence, parallel construct p also has a set of components which we denote by $p.P$. The set $p.P$ need not be an ordered set. The execution semantics of a parallel construct p is described by the following rules

$$\begin{cases} \text{execute } p_1, p_2, \ldots, p_n \text{ parallely} & \text{if } p.P \neq \emptyset \\ \text{do nothing} & \text{if } p.P = \emptyset \end{cases}$$

The execution of a parallel construct p is finished when the execution of all the elements of $p.P$ is finished. In case an actor is a sequential actor the parallel construct p can be implemented through a sequence with any permutation of the elements of $p.P$.

Now, we describe the *choice* construct. We first need a notion of a choice type. A choice type is a special kind of class whose instances can take one of the explicitly predefined values. We model a class `ChoiceType` as a subclass of `ProcessItem`. In the description of a concrete process, the class `ChoiceType` can be instantiated with the help `owl:oneOf` to restrict the values to a predefined set of values. An instance of such a class can then take only one of the predefined values and will participate during the execution of a concrete process.

We denote the choice class of a choice construct c by $c.o$. We denote a choice construct by $choice((v_1, p_1), (v_2, p_2), \ldots, (v_n, p_n))$, if an instance of $c.o$ can take values from $\{v_1, v_2, \ldots, v_n\}$ with $p_1, \ldots, p_n \in P$. We denote with $c.C$ the set of such value-component pairs. The execution semantics of a choice construct c is

$$\begin{cases} \text{execute } p_1 & \text{if } c.C \neq \emptyset \text{ and } c.o \text{ has value } v_1 \\ \text{execute } p_2 & \text{if } c.C \neq \emptyset \text{ and } c.o \text{ has value } v_2 \\ & \vdots \\ \text{execute } p_n & \text{if } c.C \neq \emptyset \text{ and } c.o \text{ has value } v_n \\ \text{do nothing} & \text{if } c.C = \emptyset \end{cases}$$

We assume that a component is executed synchronously, that is, the execution of a component starts after the completion of its predecessor component.

3. Specification of Web Resources

In this section, we show how web pages, web sites, web services and composite web services can be specified with SWPDL. Already existing web processes can be seen as implementations and can be given more meaning by specifying them with SWPDL and connecting them to the such a specification.

The web contains web pages and classical web services[2]. Services that need more interaction than just receiving the user request and sending him the response are currently available only for human consumption in form of web sites. Efforts from the research community to fill in this gap include introduction of composite web services.

The basic idea behind the specification of all the web resources with one uniform language lies in (1) identifying different types of abstract resources, namely, objects that participate in a process, operations that operate on the objects and messages that are sent to or received from the user (2) specifying the business logic with these abstract resource types and (3) viewing web services and web pages or composite web services and web sites as concrete occurrences of the process description.

2 with classical web services we mean RPC (Remote Procedure Call) style web services.

3.1. Specification of a Web Page

When a web page is requested by a user, for example by entering its URL in a browser or clicking on a link on a web page, the server obtains a request (message) from the user. The server then performs some local operations[3] and sends a web page (message) to the client machine, which is then shown to the user by the browser.

Such a process can be specified by a sequence of a communication activity followed by a local operation activity followed by another communication activity. The process description has reference to the objects that are used in the process. The message describing a simple web page contain only objects since there is no user input or action is expected.

The mapping between a message and an web page is simply a table that consists of two columns. The first column contains references to ontological entities that participate in the process and the second column contains the reference to the corresponding display element in the web page. Since an ontological entity as well as a web page element can be referenced uniquely with the help of URIs and XPATH expressions, defining such a mapping is quite straightforward. Since web pages are implemented in HTML and HTML has a tree structure, an XPATH expression to reference an element can be built easily [8, 9]. The process of recognizing and specifying ontological entities of a web page and even definition of above mentioned mappings is often referred to as semantic annotation and there is already some work and tool support to perform these tasks [10, 11].

3.2. Specification of a Web Service

Similar to a web page, a web service can be specified by a sequence of a communication activity followed by a local operation activity followed by another communication activity. In this case, however the message format is SOAP message rather than HTML. Again, since SOAP message has XML syntax and XML has a tree structure, the elements of a SOAP message can be easily referenced by XPATH expressions [9, 12].

3.3. Specification of a Web Site

Specification of a web site is a two step process. In the first step, each web page is annotated with the domain ontology. In the second step, the flow of the web pages is specified by using SWPDL ontology. We assume, that the first step is already done and focus on

3 local operations are performed in case of static web pages too.

the second step. Consider the situation that a user has loaded a web page of the web site and is viewing it in his browser. We differentiate between the following three types of actions that the user can perform on the web page.

1. User enters another URL in his browser. In this case the user loads another web page and we are again in the same situation.
2. User fills in some fields in a form on the web page and clicks on one of the submit buttons. Recall, that an SWPDL message can contain not only objects but also references to the ontological entities whose values are to entered by the user. The filling in the values in the form fields is equivalent to assigning values to the ontological entities. Clicking the submit button actually performs two tasks. First, since the submit button is another field in the form like other fields, e.g. an edit field, the user assigns this field one of the predefined values. Second, the user sends a message to the server through the same action. Note, that the submit buttons have predefined values encoded in the HTML page and the user does not have to enter any value for them but only confirm a value by clicking on the corresponding button.

   Again, each element in an HTML form can be referenced uniquely (e.g. by an XPATH expression). The mapping between a message that also contains references to ontological entities and the elements of an HTML form can be done in a similar fashion as earlier.
3. User clicks on a link on the web page. Depending on how the link is programmed the action has effect same as action (1) or action (2).

3.4. Specification of a Composite Web Service

Again, the process description contains the objects that participate in the process. Further, the description a message type for each different message that is used to carry objects from server to client or from client to server and a choice type, for each different choice made by the server. For each different local operation and communication that participates in the composite web service, a local operation type and a communication type is described respectively. For web services that are used in the composite web services, there is no need to describe a new type.

Messages can be mapped to SOAP messages as described earlier. Grounding a local operation to a concrete implementation, for example, a method in a Java bean is also quite straightforward. The only thing that is new in a composite web service is that it may contain other web services. We differentiates between two cases, (1) the provider of the composite web service does not want the user to exprience that his composite web service uses another web service. In this case, he will define a local operation and ground that to the web service. The user cannot communicate with the embedded web service and the provider of the composite web service becomes client for the embedded web service (2) the provider embeds a web service as an web service activity in his composite web service. In this case, the user communicates directly with the embedded web service[4] and the composite web service gets back the control as soon the embedded web service is finished.

4. Evaluation

We developed a tool that allows a user to specify an invocable web resource like a web page, a web service, a web site and a composite web service interactively. Such specifications are currenlty stored locally for the evaluation purpose but we envision that in the real setting they are provided by the providers of the web resources and are publicly available. The tool also allows the user to chose a specification and execute it.

We developed two SOAP based web services with Java, one that returns best laptop offer based on the input parameters, CPU speed and hard disk size, and the other offers insurance terms for a given laptop offer. We specified each web service in SWPDL with our tool. Further, we specified a composite web service that combines the two web services in a sequence. This web service returns the best laptop offer together with the best insurance terms for the laptop offer.

We also specified some web sites of the employees of our research group. Each such web site consists of a few web pages reachable directly or indirectly from the start page and contain information about teaching, research, publications etc.

Our tool loads and shows a repository of such specifications and shows. On selecting an entry in the list, the process is presented as an interactive control-flow-graph. The properties of a process component can be viewed by clicking on the component in the graphcical representation. Further, a process can also be executed. Currently, our tool supports grounding to Java methods for local operation grounding. Further, it supports

4 note, that the embedded web service can again be a composite web service with multiple interactions with the user.

only HTML and SOAP message for message groundings.

In case of a web service, when a process is executed and the user receives a message that expects him to enter values, the client KB access component retrieves the values from the client KB and automatically fill them in. If there are any objects that are not connected to the KB a generic form like user interface is generated, where the user can enter the missing values manually and respond to the message. In case of HTML messages however, the user interface is sent by the server itself, which is shown as it is. Automatic filling of the values from the client KB is implemented with javascript. We are currently planning the implementation of discovery of web resources on the basis of their explicit specifications.

5. Related Work

There are quite a few languages that are similar to our language. DAML-S differentiates between atomic and composite web services [13, 14]. We have only one concept for a process. DAML-S does not have local operations, which makes it difficult to semantically differentiate between the businees logic implemented in some programming language and running locally from a web process that a provider actually offers to others. DAML-S foresees atomic web services to specify local operations. We believe that this may be undesired by a provider in case he actually does not want to offer a particular method as a web service, but only a composite web service that uses that method. In DAML-S the provider needs to take care that he does not publish the description of such a web service and needs extra effort to protect the web service, in case someone tries to invoke it. DAML-S does not have messages and hence no explicit communication model. In DAML-S the communication between a user and the server can take place only at the time of invocation of a web service. In DAML-S it is also not clear how the inputs and outputs of a composite web service are related to those of its component web service.

BPEL4WS, WSCI and WSCL are some other languages that are similar to our process description language [15, 16, 17]. All these languages can model composite web services but have slightly different foci. One commonality among these languages is, that they are XML based and tightly bound with WSDL [18]. BPEL4WS however focusses more on the business related technical issues such as transactions.

Our language is different from these languages because it is ontology based and more abstract. Modeling the language as OWL ontology allows the concrete process specifications to be related with other semantic web resources. Although there are quite a few languages that are similar to our SWPDL to some extent, we are to the best of our knowledge not aware of any work that has shown how all sorts of semantic web resources can be specified by one single language.

Regarding our tool, we believe, that it is complementary to the existing annotation tools, since its main task is to support the users to specify the flow of web pages and web services rather than the content of the web pages based on a domain ontology. On the other hand, if a web page contains information about a process, that is, its content actually describes a process, our SWPDL ontology can play the role of the domain ontology and such a web page can be annotated with already existing annotation tools.

6. Conclusion

We presented an approach for specification of processes in the semantic web. We showed how already running web based business can be made more explicit without enforcing the providers to change their running web sites or web servics. We showed how web pages, web pages, web services and composite web services can be specified in one unified framework and we presented our tool that supports a provider in specifying his web resources and an end user in searching, browsing, using the web resources more efficiently and comprehensively.

End users as well as the providers have many advantages when the explizit specifications of the resources are made public. We list here a few.

- Users not only search for information but also for services (web-based or real world) in the web. When specifications of the resources are publicly available, users can search for not only for web pages and web services semantically.
- Searching for business processes has different criteria than searching for information. Currently, keyword based search engines like google are used for searching both types of resources. Business processes can not be searched for semantically if they are not described explicitly and in machine understandable form.
- When the specifications of a process is known to a user, the process becomes deterministic from the point of view of the user. Without explicit process specifications, a user can not know which values he should enter or which button he should click to get to a desired result. He can not know which

page will appear next when he clicks on a button. A user can not know how many form he has to fill to get the desired result. If a user has to fill, say ten forms on a web site to purchase a book and after filling and submitting nine forms he experiences that the form requires his birth date, he may want to cancel the purchase procedure because of his privacy requirement. When specifications of a process are made public, a user can check beforehand whether the requirements of a service are compatible with his requirements.

- Neither HTML nor SOAP message format allows to define constraints for values, however ontologies do. If web pages and web services are connected with a formal specification, constaints for the values can be defined in the ontology. Web browsers and web service clients can consider such contraints and perform a plausibility check already on the client side before the values are submitted to the server. Currently such checks are hardcoded in javascript in case of HTML pages.
- With a semantic process description language, providers have a way to specify their web services such that they can be easily found if someone is looking for exactly that service. Current keyword based search engines easily find popular web pages but a less popular useful page is very difficult to find. In addition to that real world businesses can be specified semantically in the web.
- Not only providers of web services can share data and functionality but also the providers of web pages.
- Providers do not need to hard code the logic for the flow of web pages and messages. Rather, they can use more generic tools and can first specify business logic without worrying much about the web pages and then design the web pages.
- Providers can easily (without complex heuristic based information retrieval methods) use the contents of other web pages (ontological instances can be generated from the web page, the ontology and the mapping between the ontology and the web page of the third party).

7. Acknowledgements

I thank Barbara Sprick for proof reading and discussing technical details about SWPDL. This work was funded by the SemIPort and Internetökonomie-SESAM projects of the Federal Ministry of Education and Research (BMBF).

References

[1] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, May 2001.

[2] D. Fensel, J. Hendler, H. Lieberman, and W. Wahlster, Eds., *Spinning the Semantic Web.* MIT Press, 2002.

[3] P. Patel-Schneider and D. Fensel, "Layering the semantic web: Problems and directions," in *ISWC2002: Ist International Semantic Web Conference, Sardinia, Italy*, ser. Lecture Notes in Computer Science. Springer, June 2002, pp. 16–29.

[4] S. Agarwal, S. Handschuh, and S. Staab, "Surfing the service web," in *ISWC2003: 2nd International Semantic Web Conference*, ser. Lecture Notes in Computer Science. Springer, October 2003.

[5] C. A. R. Hoare, *Communcating Sequential Processes.* Prentice Hall, 1985.

[6] R. Milner, *A Calculus for Communicating Processes*, ser. Lecture Notes in Computer Science. Springer, 1980, vol. 92.

[7] ——, "A polyadic pi-calculus: a tutorial," Tech. Rep., 1991.

[8] W3C, "Hypertext markup language," W3C, Tech. Rep., 2003.

[9] ——, "XML path language," W3C, Tech. Rep., 2003.

[10] S. Handschuh and S. Staab, "Cream - creating metadata for the semantic web," *Computer Networks*, vol. 42, pp. 579–598, AUG 2003, elsevier.

[11] S. Handschuh, S. Staab, and F. Ciravegna, "S-cream – semi-automatic creation of metadata," *Expert Update, Special Issue - Intelligent Services for The Knowledge Lifecycle*, pp. 20–31, 2002.

[12] W3C, "Simple object access protocol," W3C, Tech. Rep., 2003.

[13] A. Ankolekar, M. H. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. V. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. R. Payne, and K. Sycara, "DAML-S: Web Service Description for the Semantic Web," in *ISWC2002: Ist International Semantic Web Conference, Sardinia, Italy*, ser. Lecture Notes in Computer Science. Springer, June 2002, pp. 348–363.

[14] K. Sycara, M. Paolucci, A. Ankolekar, and N. Srinivasan, "Automated discovery, interaction and composition of semantic web services," *Journal of Web Semantics*, vol. 1, no. 1, pp. 27–46, 2003.

[15] T. Andrew, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana, "Business process execution language for web services," Bea Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, Tech. Rep., 2003.

[16] B. Systems, Intalio, SAP, and S. Microsystems, "Web service choreography interface," W3C, Tech. Rep., 2002.

[17] H. P. Company, "Web service choroegraphy language," W3C, Tech. Rep., 2002.

[18] Ariba, IBM, and Microsoft, "Web service description language," W3C, Tech. Rep., 2001.