Semi-Automatic Acquisition of Semantic Descriptions of Web Sites

Sudhir Agarwal
Institute of Applied Informatics and Formal Description Methods (AIFB) and
Karlsruhe Service Research Institute (KSRI)
University of Karlsruhe (TH),
Karlsruhe, Germany.
agarwal@kit.edu

Abstract

In order to obtain the desired information or functionality in the Web, a user often needs to perform multiple interactions with the Web site, e.g. submitting Web forms filled up with appropriate information, and the further execution of such a Web process depends on the information provided by the user in the previous steps of the process. The formal models underlying existing systems for supporting users in coping with the Web do not capture the dynamics and data flow of Web processes. As a result, searching for information in the so called "Deep Web" or desired business processes offered via the Web still requires significant manual effort.

In this paper, we present a semantic process description language and present a mapping of the dynamics and data flow of Web sites to our semantic process description language. In order to allow development of more sophisticated methods and tools that consider the dynamics and data flow inside or among Web sites, significant number of descriptions of Web sites are needed. We approach this bootstrapping problem by presenting a technique for semi-automatic acquisition of semantic descriptions of Web Sites.

1. Introduction

There is a tremendous amount of information hidden in the so called "Deep Web" [1]. This information is offered to the users via dynamically generated Web pages [1]. In order to access the information in the deep Web, a user needs to perform certain steps, e.g. submitting the Web forms filled up with appropriate information. For example, in order to find the weather of some city on some data, a user needs to enter the name of the city and the date in a form. Furthermore, a lot of business is offered via Web sites, e.g. selling books, that can have effects in the real world, e.g. ordering a book at an online shop triggers the shipment of the book to user. Again, in order to avail the functionality offered by such business processes, a user needs to perform certain actions, e.g. selecting a book, entering the shipment address, entering the credit card details etc. In addition to Web sites, information and business are also provided via Web services.

1. We will use the terms "dynamic Web pages" and "static Web pages" in the sense of their intuitive meanings. Theoretically, the boundary between the two is not clearly defined.

In order to support a user in coping with the huge amount of information and business provided via the Web, automatic tools have been developed. The most prominent tools are perhaps the search engines like Google, Yahoo! and MSN. The formal models underlying such tools mainly consider the content of the Web pages for building their respective search indexes but hardly the dynamic aspects of the Web. More precisely, the control and data flow among various Web pages and the dependency of the content of a Web page which appears as part of a process (e.g. book ordering process) on the user input received on the previous pages of the process. As a result, currently, it is hard to find Web sites that do offer the desired information but on a page generated dynamically at some later stage of the process and not on the very first (mostly static) page.

In this paper, we view Web pages, Web sites and Web services in a unifying way as Web processes and focus mainly on the dynamic aspects of such processes. With such a process oriented view on the Web, we aim to build the basis for the development of more sophisticated methods and tools that can address the increasing demands of the users. Since acquisition of expressive descriptions of Web processes is one of the biggest hurdles on the way to a repository of descriptions of Web processes, we also present a semi-automatic technique for the acquisition of such descriptions.

The paper is structured as follows. In Section 2, we give a short overview of our formalism for describing distributed processes [2]. In Section 3, we present a mapping of Web artefacts and some HTML elements to our process description language. In Section 4, we present our semi-automatic approach for acquiring the semantic descriptions of Web pages. We summarize our results and discuss some future work in Section 6 after discussing the related work in more detail in Section 5.

2. Semantic Description of Web Processes

As motivated in Section 1, in order to enable more sophisticated automatic support, the necessary information about Web processes must be available in machine interpretable form. In this section, we give a short overview of the language for describing Web processes semantically [2] and show how Web sites can described as processes in this languages. The formalism is a novel combination of the process algebra π-calculus [3] and the description logic $\mathcal{SHIQ}(\mathbf{D})$. In Section 2.1, we propose to use $\mathcal{SHIQ}(\mathbf{D})$ ontologies to describe resources (domain ontologies) semantically. In Section 2.2, we present a formalism for describing the functionality of Web services semantically.

2.1. Modeling Resources with Ontologies

We specify concrete resources as description logic individuals, among which relationships "$=$" and "$\neq$" can be specified. These relationship types are necessary to achieve interoperability in the descriptions of individuals and are directly provided by expressive description logics, e.g. $\mathcal{SHIQ}(\mathbf{D})$, which the decidable variant OWL-DL of the Web ontology language OWL[2] is also based on. The resources can be further classified into sets that can be hierarchically ordered according to the subset relationship. Again, expressive description logics provide the "$\sqsubseteq$" relationship type to relate the sets. In addition to the mentioned relation types, $\mathcal{SHIQ}(\mathbf{D})$ also allows the modeling of arbitrary relation types among concepts and use of relations among individuals.

2.2. Modeling Behaviour

A Web service whether stateless or stateful is a process. Mostly, RPC (Remote Procedure Call) like Web services are stateless, whereas Web services with a flow of Web pages are typically stateful. In order to model the semantics of Web services, we model them as processes. The syntax of the formalism is defined recursively as follows:

$$\begin{array}{lll} P & ::= & \mathbf{0} \mid y[v_1 \ldots, v_n].P \mid y\langle x_1 \ldots, x_n\rangle.P \mid \\ & & l(x_1, \ldots, x_n)(y_1, \ldots, y_m).P \mid \omega?P_1\colon P_2 \mid \\ & & P_1 \parallel P_2 \mid P_1 + P_2 \mid \ @A\{y_1, \ldots, y_n\} \end{array}$$

The *Null process* $\mathbf{0}$ is a process that does nothing. This process is used to denote the termination of a process.

The *Input process* $y[v_1, \ldots, v_n].P$ is a process that inputs arbitrary names $z_1, \ldots, z_n$ at port y, binds them to names $v_1 \ldots, v_n$ and then behaves like the process $P\{z_i/v_i\}$, where z_i/v_i denotes substituting z_i for v_i.

2. http://www.w3.org/2004/OWL/

The *Output process* $y\langle x_1, \ldots, x_n\rangle.P$ is a process that outputs the names $x_1 \ldots, x_n$ at port y and then behaves like the process P.

In an input process expression $y[v_1, \ldots, v_n].P$, $v_1, \ldots, v_n$ are variables. We model variables as DL A-Box individuals within the name space of the corresponding actor. The reason for doing this is that variable v can be bound to a value α (which is again an A-Box individual) by adding an A-Box individual equality axiom $v = x$ in the knowledge base. Once, we have such an axiom in the knowledge base, the variable v can be used just as a value as in case of programming languages. In an output process $y\langle x_1, \ldots, x_n\rangle.P$, $x_1, \ldots, x_n$ are resources, which are also modeled as DL A-Box individuals and are available since all the individual names of the associated ontology can occur freely in the process expression.

In practice, information about the type of communication protocol and the type of messages that can be transmitted over a channel is very useful. E.g. one may wish to know whether the book selling business process will send the book via HTTP as PDF or via surface mail as hard copy.

For communication activities (input as well as output), we use a pair (p, a) in place y, where p is the communication protocol, e.g. "http", "phone", "fax", "surface-mail" etc., and a is the address. Finally, by modeling channel types as description logic individuals we make sure that channel descriptions can be sent and received just like any other resources and thus the mobility is preserved.

The *Local* process $l(x_1, \ldots, x_n)(y_1, \ldots, y_m).P$ performs the operation l locally wrt. to the server with the arguments $x_1, \ldots, x_n$ and produces output $y_1, \ldots, y_m$. It then behaves like the process P.

A local operation is a decidable procedure that updates the A-Box of the agent that executes the local operation. A local operation may perform a query on the local knowledge base or some calculation to add new individuals and add corresponding axioms in the knowledge base to relate the individuals with each other. It can also remove existing DL axioms from the knowledge base. So, we define a local operation type $L(x_1, \ldots, x_n)(y_1, \ldots, y_m)$ as a list of change types Δ and a list of rules R. Each change type $\delta \in \Delta$ is a parameterized proposition, where the parameters belong to the set $\{x_1, \ldots, x_n\}$. Furthermore, a change type δ is adorned with "+" or "-" which indicates whether the proposition corresponding to δ is added to or removed from the knowledge base. For example, if the change type $\{+classMember(x_1, x_2)\}$ belongs to the change types of a local operation type $L(x_1, x_2)()$, executing L with arguments $Peter$ and $Person$ will add the axiom $classMember(Peter, Person)$ in the knowledge base. The set of rules R contain rules that relate the output variables $y_1, \ldots, y_m$ to the input variables $x_a, \ldots, x_n$. A concrete invocation of a local operation type $L(x_1, \ldots, x_n)(y_1, \ldots, y_m)$ contains values for the input parameters $x_1, \ldots, x_n$ and yields values for the output variables $y_1, \ldots, y_m$.

The *Deterministic Choice* $\omega?P_1\colon P_2$ is a process that checks whether the condition ω is true or not. It behaves like the process P_1, if the condition ω it true otherwise like the process P_2. In practice one needs to check rich conditions, e.g. whether the income of person x is higher than the income of person y. The process expression $\omega?P\colon Q$ that behaves like P if the condition ω is true, and otherwise like Q. The condition ω can be any predicate in the ontology of actor that checks the condition. These include concept names, relation names defined in the T-Box, rule heads of DL-safe rules in the R-Box of the ontology and any predicate symbols, the implementations of which lie outside the description logic reasoner. This allows to model very expressive conditions the check for whose truth value is still decidable. For more details about the formalism, in particular its formal semantics, we refer to [2].

The *Composition* $P_1 \parallel P_2$ consists of P_1 and P_2 acting in parallel. The components may act independently; also, an output action of P_1 (resp. P_2) at any output port x may synchronize with an input action of P_2 (resp. P_1) at x, to create a silent (τ) action of the composite agent $P_1 \parallel P_2$.

The *Summation* $P_1 + P_2$ denotes the non-deterministic choice and behaves either like P_1 or like P_2. In contrast to deterministic choice, in which a process evolves depending on the truth value of some condition, e.g. equality, in case of non-deterministic choice, the user of the process selects one of the alternatives and the selected alternative is executed.

The named process expression is called *Agent Identifier*. For any agent identifier A (with arity n), there must be a unique defining equation $A(x_1, \ldots, x_n) \overset{def}{=} P$, where the names $x_1, \ldots, x_n$ are distinct and are the only names which may occur unbound in P. Now, the process *Agent* $@A\{y_1 \ldots, y_n\}$ behaves like $P\{y_1/x_1, \ldots, y_n/x_n\}$. Note that defining equations provide recursion, since P may contain any agent identifier, even A itself.

3. Semantics Description of Web Sites

Having an expressive process description formalism at hand, we now present how Web sites can be seen as processes and thus described by our process description formalism semantically. To do so, we present in this section a mapping between common elements of HTML and our process description language.

Seen abstractly, a Web page consists of mainly two types of elements, namely elements that present information to the user, e.g. text paragraphs, tables etc. and elements that are meant for interacting with the user, e.g. forms and links. In general, a Web page is a part of a Web site and may not be directly reachable but appears only on a certain navigation path of the site. Table 1 presents our view on Web artifacts as a elements of our process description language.

Web Artefact	**Process Language Element**
URL/Web site	Agent identifier
Web page	Message
Link	Invocation of an agent identifier
Form	Input process
Form field name	concept
Form field values	Instances of the concept corresponding to the field name
CGI script	Local operation
Web	Agent identifier with all Web sites as concurrently running components

Table 1. Mapping between Web Artifacts and Elements of our Process Language

When a URL is accessed, the Web server produces a Web page and sends it to the client. In some cases, a URL has arguments (appended to the base URL after the symbol "?"). So, a URL u with arguments $a_1, \ldots, a_n$ can be seen as an agent identifier $u(a_1, \ldots, a_n) \overset{def}{=} u(b_1, \ldots, b_m)(o_1, \ldots, o_l).c\langle o_1, \ldots, o_l\rangle.P$, where $\{b_1, \ldots, b_m\} \subseteq \{a_1, \ldots, a_n\}$. The left hand side of the expression describes that the agent u has arguments $a_1, \ldots, a_n$. The right hand side of the expression describes the defining process of the agent identifier u and says that an agent of type u first performs a local operation u with $b_1, \ldots, b_m$ as input arguments and $o_1, \ldots, o_l$ as outputs. Note that $\{b_1, \ldots, b_m\} \subseteq \{a_1, \ldots, a_n\}$ must hold since $\{a_1, \ldots, a_n\}$ are the only names than may occur freely in the defining process expression. The names $o_1, \ldots, o_l$ denote the content of the HTML page, which is then sent to the client in the subsequent output activity. Some of the o_i may be links to other Web sites or submit buttons of the forms. Each such o_i denotes a possibility to navigate to the next page. That is, process P that denotes the further evolution of the process can be seen as the $P = \sum o_i$ with $o_i \in \{o_1, \ldots, o_l\}$.

If an o_i is a link, it denotes the usage of a URL. The names of the arguments, if any, can be seen as concepts of one or more ontologies, whereas the values of the arguments as instances of the concepts corresponding to the respective argument.

In case of a form, the "action" can be either a link or the invocation of a method executed locally by the server (CGI script). The usage of a URL is equivalent to the invocation of an agent identifier, whereas a CGI script is equivalent to the invocation of a local operation. Submission of a form binds the values entered in the form fields to the names of the corresponding fields. Note, that such a binding happens before the server executes the CGI script. Hence, a form f with field names $v_1, \ldots, v_n$ can be seen as an input process $f[v_1, \ldots, v_n].P$ where P denotes the process representing the non-deterministic choice of all the submit buttons of the form f. That is, if the form f has submit buttons $b_1, \ldots, b_m$, then is $P = \sum b_1, \ldots, b_m$.

The field names $v_1, \ldots, v_n$ can be seen as concepts on one or more ontologies, as a field name denotes the set of all possible values. Sometimes, a form field already contains the list of possible values, e.g. a combo box, a list box or radio buttons. These values represent possible values for the field name and can be seen as instances of the concept corresponding to the field name.

This way, a Web site can be seen as an agent identifier and even the whole Web as an agent identifier with all the Web sites as concurrently running agents.

4. Acquisition of Semantic Descriptions

In this section, we describe how we aim to acquire a large set of high quality semantic descriptions of Web sites. The proposed semi-automatic solution consists of a crawler and an automatic extraction component to achieve initial semantic descriptions formalized with the formalism presented in Section 2, which can be refined at a later stage manually with appropriate tool support.

We denote with $\mathcal{A}$ the set of agent identifiers and with $\mathcal{O}$ the set of domain ontologies. Algorithm 1 is the main crawler loop. Algorithms 2 and 3 are used within Algorithm 1 to create semantic descriptions of a Web page reachable via clicking link or by submitting a form respectively. Algorithm 4 is a utility algorithm that is used within Algorithm 3 to process various types of form fields and add the information as concepts or instances in the domain ontology of a Web page.

Algorithm 1 initializes the set of agent identifiers $\mathcal{A}$ as well the set of domain ontologies $\mathcal{O}$ to $\emptyset$. The algorithm needs a set of URLs that serve as seed URLs for the crawler. When the crawler runs and fetches information about Web pages that are part of a Web site, the process expressing defining the agent identifier corresponding to the Web site needs to be expanded accordingly. For this purpose, the agent identifiers for each of the URLs in the starting queue are created with Algorithm 2, that is responsible for processing links. Once the agent identifiers have been created, the main crawler loop begins and runs until the queue is empty. In each iteration, it dequeues the first link in the queue, while adding possibly new links to the queue that are reachable from the link under consideration via "clicking" links or "submitting" forms.

As it becomes clear from the algorithms presented in this section, the crawling component creates a separate ontology for each agent identifier. We foresee that these ontologies can be aligned semi-automatically. Approaches like [4] can be employed to find alignments among ontologies automatically, which can be further refined manually. Another important information that is foreseen for manual acquisition is defining the relationships between the input and output variables of a local operation type (cf. Section 2.2).

Algorithm 1 Semi-Automatic Acquisition

```
1: Initialize $\mathcal{A}$ to $\emptyset$. Initialize $\mathcal{O}$ to $\emptyset$.
2: Let $Q$ denote the queue containing the seed URLs of the crawler.
3: for all $q \in Q$ do
4:     Create agent identifier for $q$ with Algorithm 2 and denote it with $q()$.
5: while $Q \neq \emptyset$ do
6:     Dequeue the first URL from $Q$, denote it with $q$.
7:     Fetch the Web page $p$ located at $q$.
8:     Replace $h$ in the definition of $q()$ with the string representing the HTML code of the Web page $p$.
9:     Extract all links in the Web page $p$, denote the set of extracted links with $L_p$.
10:    Initialize a set of agent identifiers $N$ to $\emptyset$.
11:    for all $l \in L_p$ do
12:        Create agent identifier for link $l$ with Algorithm 2 and add it to $N$.
13:        enqueue link $l$ to the queue $Q$.
14:    Extract all forms in the Web page $p$, denote the set of extracted forms with $F_p$.
15:    for all $f \in F_p$ do
16:        Create agent identifier for form $f$ with Algorithm 3 and add it to $N$.
17:    Links in $L_p$ and the URLs specified in the "action" elements of the forms in page $p$ are the only possibilities to to the next page from page $p$.
18:    Replace the process expression $P$ at the end of the agent identifier $q()$ with the process expression $\sum_{n \in N} n$. The later describes a non-deterministic choice process with all the action URLs of the forms and links in the Web page $p$ as options, a user chooses from.
```

One of the advantages of our approach is that we do not aim for indexing different instances of a Web page generated dynamically for different values of the arguments. Rather, our descriptions capture the dependency of a Web page on its arguments. More precisely, we do not submit forms for all possible combinations of the values of the form fields and store the resulting Web page. This not only saves a lot of storage space, but also does not cause extra load on the Web servers.

5. Related Work

Current search engines like Google, Yahoo! and MSN offer only limited support for searching for Web pages that offer certain information or effects after some user interactions. The reason for this is that the search is based on a search index that is built automatically by a crawler which cannot reach many dynamically generated Web pages [5]. Another shortcoming of the current search engines is that

Algorithm 2 Create Agent Identifier for a Link

```
Require: URL $l$
1: Let $b$ denote the base URL of $l$. That is, the part of the URL $l$ before "?".
2: Extract the (name, value) pairs of the arguments of link $l$. Let $names(l)$ denote the set of names and $a(l)$ the set of these (name, value) pairs. Further, let $name(a)$, with $a$ a (name,value) pair, denote the name in the pair $a$ and $value(a)$ the value in the pair $a$.
3: if $o_b \notin \mathcal{O}$ then
4:     Add an ontology $o_b$ in the set of ontologies $\mathcal{O}$.
5: for all $a \in a(l)$ do
6:     Add a concept $name(a)$ in the ontology $o_b$
7:     Add an individual $value(a)$ as instance of the concept $name(a)$ in the ontology $o_b$
8: Add a local operation type $L_b(names(l))(h)$ to the set of local operation types
9: Add an agent identifier $b(names(l)) \overset{def}{=} @L_b(names(l))(h).c\langle h\rangle.P$ to the set of agent identifiers $\mathcal{A}$
Ensure: Agent identifier $b$
```

Algorithm 3 Process a Form

```
Require: Form $f$
1: Process the fields of the form $f$ with Algorithm 4.
2: Extract the URL specified under "action" in the form $f$. Let $a_f$ denote the action URL.
3: if Action method is "get" then
4:     Add an agent identifier $f() \overset{def}{=} p[names(f)].@a_f\{names(f)/names(a_f)\}$ to the set of agent identifiers $\mathcal{A}$. {This means that in case action method is "get", the form behaves like the agent corresponding to the action URL after inputting the field values. The field values are passed to the invocation of the agent corresponding to the action URL.}
5: else
6:     Add a local operation type $a_f(names(f))(h)$ to the set of local operation types
7:     Add an agent identifier $f() \overset{def}{=} p[names(f)].@a_f(names(f))(h).c\langle h\rangle.P$ to the set of agent identifiers $\mathcal{A}$. {This means that in case action method is "post", after receiving the input values from the user, the server performs an operation locally with the field values and then produces the next Web page }
Ensure: Agent identifier $f$.
```

Algorithm 4 Process Form Fields

```
Require: Form $f$
1: Extract the names of the fields in form $f$. Let $names(f)$ denote the set of names of form $f$.
2: for all $n \in names(f)$ do
3:     Add the concept $n$ in the ontology $o_p$.
4:     Extract the values of the field $n$ and denote them by $v(n)$
5:     for all $v \in v(n)$ do
6:         add an individual $v$ as instance of the concept $n$ in the ontology $o_p$.
Ensure: $names(f)$
```

they do not support search based on the user constraints on the process of reaching certain information or achieving an effect. E.g. a user that wishes to order a book may want to search for books by their ISBN and may not want to enter his date of birth at any stage of the book ordering process.

The contributions of [5] are manifold. Firstly, it presents an algorithm for informativeness test that is used to evaluate query templates, i.e. combinations of form inputs. Secondly, it presents an algorithms for efficiently traversing the space of query templates to identify those suitable for surfacing. Thirdly, it presents an algorithm for predicting appropriate inputs values for text boxes. Our approach is different from [5] since it is based on semantic annotation of the services and not on indexing probably the whole database of the service providers. Note that in some cases it may be legally forbidden by the service providers for other parties to index their database partially or completely. Furthermore, in our approach we can deal with a sequence of Web pages whereas the approach presented in [5] can deal with single pages.

BPEL4WS [6] is a popular formalism for modeling business processes in the Web. It combines XLANG and WSFL. However, it still lacks formal semantics and reasoning procedures. Therefore, our work is complementary to BPEL4WS as our formal model may be used in its extended form to specify formal semantics for BPEL4WS, which is needed in order to prove certain properties of reasoning algorithms based on BPEL4WS. Note, that all the attempts to define formal semantics of BPEL4WS cover only the dynamic behavior of BPEL4WS. Resource schemas that are an essential part of multi party business processes have not been considered. Furthermore, BPEL4WS views Web services as black boxes with one input and one output activity. In our model, we do not have this restriction. In our view, services can have multiple interactions with the user. BPML[3] is similar to BPEL4WS in the sense that it focuses more on the execution of a business process than on reasoning about properties of a process.

3. http://www.bpmi.org

Traditional research on semantically annotating Web pages mainly focussed on static Web pages [7]. Later some approaches addressed the issue of deep annotation of Web pages. For example [8] relies on the cooperation of the providers of the dynamic Web pages and expects them to generate the semantic annotation together with the Web page and integrate it in the Web page before sending it to the client. Apart from the rather limited acceptance of these approaches in the practice, the annotation formalism underlying these approaches does not allow to describe the user input dependent flow and contents of Web pages.

OWL-S Process Model [9] is an OWL ontology to describe the choreography of composite Web services. However, OWL-S Process Model does not allow to model process variables in a clean way, which makes it difficult to model the relationships between inputs and outputs of various activities. Furthermore, there is no formal semantics of the current OWL-S Process Model, which makes it difficult to automatically reason about the data and control flow of composite Web services modeled with OWL-S. Note, that the formal semantics presented in [10] covers one of earlier versions of DAML-S.

WSMO (Web Service Modeling Ontology) provides the conceptual underpinning and a formal language for semantically describing Web services in order to facilitate the automatization of discovering, combining and invoking electronic services over the Web [11]. WSMO is more a formalized bird-eye view than a concrete Web service description language. Existing reference implementations of WSMO are based on the Web Service Modeling Language(WSML) [4]. In contrast to OWL-S WSMO proposes to model pre-conditions and effects of a service as logical expressions in WSML. However, the semantics of pre-conditions and effects can not be captured by a WSML reasoner since it can not reason about changing knowledge base. Furthermore, currently, WSMO only provides techniques for modeling atomic Web services but there is no WSMO approach for modeling processes.

6. Conclusion and Outlook

In this paper, we addressed the shortcoming of the current search engines while dealing with dynamics and data flow inside or among Web sites. We presented a semantic process description language and showed how Web sites can be viewed as processes with multiple user interactions by presenting a mapping of Web sites to our process description language. In order to enable development of sophisticated methods and tools that can live up to the ever growing expectations of the users, it is necessary to have a significant number of semantic descriptions of Web sites. We address this bootstrapping problem by showing how such semantic descriptions of Web sites can be acquired semi-automatically by presenting algorithms for automatically creating semantic descriptions that can be manually refined and modified. These algorithms can be easily incorporated in a Web crawler.

4. http://www.wsmo.org/wsml/wsml-syntax

Acknowledgment

Research reported in this paper was supported by the EU in the project SOA4All (http://www.soa4all.eu).

References

[1] M. K. Bergman, "The deep web: Surfacing hidden value," Journal of Electronic Publishing, 2001.

[2] S. Agarwal, S. Rudolph, and A. Abecker, "Semantic description of distributed business processes," in Proceedings of AAAI Spring Symposium - AI Meets Business Rules and Process Management, March 2008.

[3] R. Milner, J. Parrow, and D. Walker, "A Calculus of Mobile Processes, Part I+II," Journal of Information and Computation, pp. 1–87, September 1992.

[4] M. Ehrig, S. Staab, and Y. Sure, "Bootstrapping Ontology Alignment Methods with APFEL," in Proceedings of the 4th International Semantic Web Conference, ISWC 2005, Galway, Ireland, November 6-10, 2005., ser. LNCS, Y. Gil, E. Motta, V. R. Benjamins, and M. A. Musen, Eds., vol. 3729. Springer, November 2005, pp. 186–200.

[5] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy, "Google's deep web crawl," Proc. VLDB Endow., vol. 1, no. 2, pp. 1241–1252, 2008.

[6] 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.

[7] S. Handschuh and S. Staab, Annotation for the Semantic Web. IOS Press, 2003.

[8] S. Handschuh, S. Staab, and R. Volz, "Annotation for the deep web," IEEE Intelligent Systems, vol. 18, no. 5, pp. 42–48, SEP, special issue on information integration.

[9] 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, December 2003.

[10] A. Ankolekar, F. Huch, and K. Sycara, "Concurrent Execution Semantics for DAML-S with Subtypes," in Proceedings of the First International Semantic Web Conference: The Semantic Web (ISWC 2002), ser. Lecture Notes in Computer Science (LNCS), I. Horrocks and J. A. Hendler, Eds., vol. 2342. Sardinia, Italy: Springer, 2002, pp. 14–21.

[11] D. Roman, U. Keller, H. Lausen, J. de Bruijn, R. Lara, M. Stollberg, A. Pollers, C. Feier, C. Bussler, and D. Fensel, "Web Service Modeling Ontology," Applied Ontology, vol. 1, no. 1, pp. 77–106, 2005.