Abstract

Web services technology is a new solution of communication between applications over the Web. Currently, this technology has gained a suitable degree of maturity and a widespread adoption, especially in business community, as it is used to publish easily business functions for remote execution. To provide such capability, service consumers search public business registries, find the requested Web service and then bind that service. For subsequent uses of the same Web service, and to reduce time and effort of its re-invocation, service consumer can save locally binding details of that service in his local business repository. An important issue arising from this situation is how to synchronize conveniently and efficiently such details with original ones whenever the Web service is updated. This fact may disrupt the local binding of that service. To deal with this issue, we propose in this paper an agent-based approach for synchronizing bindings of Web services to maintain their consistency and sustainable use.

1. Introduction

In the vision of a world in which modern enterprises have to accommodate business agility in today’s ever more competitive and global markets, Web services have emerged as a prominent technology to address this critical issue. Indeed, enterprises have become able to expose their existing business processes as Web services that can be easily assembled and interact in a highly standardized manner. Due to this fact, an increasing number of Web services are started to be deployed during the last few years.

As Web services proliferate, the ability to find and bind services of interest across heterogeneous environments becomes a major challenge and raises several concerns such as reliability, availability and performance. Solving these problems is key to the ultimate success of Web services and the solutions become of paramount importance if complex integration models, such as the Service-Oriented Architecture (SOA) [1], are to be successfully deployed.
The key components for the deployment of this architecture are a service provider, a service consumer and a service registry. To find the service they need, service consumers search the service registry where service providers publish the services they offer. Currently, Web services are the suitable solution to develop SOA-based products.

The fundamental standards that make use of Web services are XML [2], WSDL [3], SOAP [4] and UDDI [5]. Although these standards have gained some levels of maturity, Web services technology still has serious lacks for providing guaranties to bind Web services efficiently. One of the main challenges in binding Web services is the fact that users who are looking for relevant Web services will have to devote hours searching through potential service resources independently. Furthermore, the repeated access to the same read-only Web service with the same request may often produce the same response which can be considered a useless work and a waste of time and effort.

Due to these facts and to optimize the binding of Web services, we have proposed in a previous work a Local Repository-Based Framework (LRBF) [6] that provides users with a local access point to bind Web services of interest instead of going through heterogeneous environments (business registries, service portals, search engines, etc.). This framework enables the collection of binding details (WSDL files, binding protocols, endpoints, service operations, functional parameters, etc.) for the most frequently used Web services in Local Web Service Repositories (LWSRs) where users can bind locally these services in subsequent uses.

However, the local binding of Web services presents an inherent weakness that threatens the performance of the LRBF framework. Indeed, as most Web services are unstable (updated, removed, replaced, unavailable, etc.), thereby their binding details saved locally within LWSRs risk to be outdated and inaccurate. To maintain reliable binding of Web services in such environment, we propose in this paper an extension to LRBF with an agent-based model for synchronizing Web service bindings. This model enables the update of locally saved binding details for Web services whenever they are changed.

Although, there have been some efforts attempting to enhance the synchronization issues of Web services, none of them emphasizes the binding aspects. For example, Chainbi et al. have proposed an agent-based framework for autonomic Web services which include synchronization concerns. However synchronization aspects related to their framework have not included binding information but have concerned only QoS properties [7].

The rest of the paper is organized as follows. Section 2 outlines the Web services synchronization issue related to the architecture of the existing framework. In section 3, we describe the proposed agent-based model for synchronizing Web service bindings. Prior to the conclusion in section 5, section 4 reviews a usage scenario.

2. Web Services Synchronization Background

Over the last decade, business registries have been the major building blocks required for successful Web services. Indeed, the UDDI Business Registry (UBR) has emerged as a universal brokerage system that service providers can use to advertise the services they offer whereas service consumers can use to discover services that fit their requirements. However, as Web services proliferate size and magnitude of UBR substantially increase, which may threaten its usefulness. Furthermore, the development of new operating systems, server applications and APIs equipped with built-in functionalities and tools has allowed businesses and companies to create their own Public Business Registries (PBRs) where to publish their home-grown Web services.

Due to these facts, the UBR becomes out of use and hosting this registry is no longer necessary as it was reported from Microsoft, IBM and SAP the original operators of UDDI project [8]. By the closing of UBR along with the increasing demand for Web services, PBRs are becoming the convenient way to find services of interest easily. Service consumers can search PBRs for Web services by categories and tags, or by using a set of advanced search qualifiers. However, the use of either of these methods will always give an incomplete result since the coverage rate in Web services of each PBR is limited to few tens whereas the current number exceeds 28,000 [9]. To fill the gap, hundreds if not thousands of PBRs may exist to cover all the available Web services. Hence, to find bindings of a suitable Web service, users will soon face the challenge of browsing separately a huge number of PBRs which consume time and effort.

To save time and effort of browsing PBRs, service consumers may use Web search engines (e.g. Google) to find easily and quickly binding details of services of interest. However, the information retrieval model of search engines is not suitable for searching Web service bindings.

Due to the fact that UBR, PBRs and Web search engines are considered as the major solutions to find binding
details of Web services and due to their aforementioned limitations we assumed that their interoperability would complement each technology’s strengths.

Motivated by this assumption and to optimize the binding of Web services from heterogeneous environments, the LRBF framework was proposed that focuses on binding services of interest locally. The first level of binding optimization is public, that is, to avoid the separate access of various service resources (especially PBRs) we provided a Public Web Service Repository (PWSR), as a unique access point for users, to discover bindings of Web services before saving them in the PWSR. The second level of binding optimization is local, that is, to avoid the repeated binding to the same Web services from the PWSR (accessed remotely from the server host) bindings of the frequently used Web services are saved locally in the LWSRs (Local Web Service Repositories) of service consumers. This operation is performed by the Local Web Service Crawler Engine (LWSCE).

As we can deduce from the description above, Web service bindings are saved twice. The first time is from PBRs to PWSR for all Web services and the second time is from PWSR to LWSRs for only the frequently used Web services. Therefore, to optimize the binding of Web services within the existing framework three categories of business repositories are needed:

- PBRs hold binding details of services provided by the PBRs’ owners namely the service providers;
- PWSR holds binding details of Web services available in all PBRs;
- LWSRs hold binding details of only the frequently used Web services required by LWSRs’ owners namely the service consumers.

Since Web services may change their bindings, for instance updating their access points, binding protocols, functional and non-functional parameters and so forth, therefore such updates have to be forwarded from PBRs where updates occur to PWSR and LWSRs respectively where bindings of the updated Web services are saved. Otherwise, using outdated binding details from LWSRs may not lead to any result since these details no longer much the original ones. Therefore, the local binding of Web services will not be possible. To sort out this problem, we propose in the next section an agent-based synchronization model to preserve the consistency of Web service bindings over PWSR and LWSRs whenever changes occur in PBRs.

3. An Agent-Based Synchronization Model

The goal of synchronizing Web service bindings within the LRBF is mainly to ensure that business repositories (PBRs, PWSR and LWSRs) see all the changes that have originated by a service provider. An additional goal is that local binding inquiries made at LWSRs yield results consistent to those made at PWSR or PBRs. To meet these goals, we introduce in this section an agent-based model for dynamic synchronization of Web service bindings which will overcome limitations of the LRBF.

In fact, the use of agent paradigm in our proposal is basically argued by the need for autonomy to initiate the synchronization process of Web service bindings between business repositories without the direct intervention of service providers. Previously, Wooldridge has defined a software agent as “a computer system that is situated in some environment, and that is capable of autonomous actions in this environment in order to meet its delegated objectives” [10, 11]. In our case, and according to this definition, the environment that hosts software agents is the LRBF framework and precisely its underlying business repositories whereas the autonomous action to carry out by these agents in this environment is to pick up newly updated service bindings actively from Web service provider, and then to incorporate such updates in business repositories (PBRs, PWSR and LWSRs). Accordingly, the delegated objective to meet by software agents in such environment is to maintain bindings of Web services that are saved locally in LWSRs coherent and synchronized with original ones available in PBRs and thereby to ensure a reliable local binding of Web services whenever they are changed.

Given these facts and according to the infrastructure of the LRBF [6], the architecture of the proposed model for binding synchronization of Web services, shown in Fig 1, involves three participating software agents such as Public Web Service Synchronizer Agent (PWSSA), Universal Web Service Synchronizer Agent (UWSSA) and Local Web Service Synchronizer Agent (LWSSA). Each agent interacts with one particular business repository according to the tasks to be carried out on this repository. Indeed, upon receiving new changes in service binding details from the service provider the PWSSA proceeds to save the updated data in the PBR and then notify the UWSSA to receive
such changes. As soon as the latter agent accepts the reception of changes the PWSSA sends a message incorporating the updated data to the Agent Message Queue (AMQ) of the UWSSA. Whenever a new message is posted in its AMQ the receiver agent picks it up first and then stores the included data in the PWSR. To forward these updates to the local repository of service consumer, the LWSSA periodically requests the UWSSA the new updated data being incorporated in the PWSR. When asked to provide such updates, the UWSSA checks if the requested data is in its AMQ and, in this case, replies the requestor agent with a message that includes the updated binding details. Upon receiving this message, the LWSSA incorporates the included data in the LWSR. The architecture of the proposed model is shown in Fig 1.

![Fig. 1. Architecture of the agent-based synchronization model](image)

As depicted in the above figure, the underlying components of the proposed model involve three software agents (PWSSA, UWSSA and LWSSA), two participating roles (Web service provider and Web service consumer) and three business repositories (PBR, PWSR and LWSR). As we have mentioned previously, each agent interacts with one particular repository to carry out tasks on that repository in response to new updates of service bindings from the Web service provider. Furthermore, agents interact with each other to synchronize such updates between business repositories in order to ensure that PBRs, PWSR and LWSRs see all the changes that have originated by the service provider and thereby the local binding inquiries from LWSRs will yield results consistent to those made at PWSR or PBRs.

To ensure reliable binding synchronization mechanism between business repositories, software agents have to carry out suitable behaviors in response to new requirements of their environment. Typically, we can distinguish among three basic types of behavior such as “One-shot” behaviors that complete immediately, “Cyclic” behaviors that never complete and “Generic” behaviors the complete when a given condition is met [12]. Furthermore, two additional behaviors may be carried out at given points in time namely “Waker” behaviors that complete after a given timeout expires, and the “Ticker” behaviors that never complete waiting a given period after each execution.

Having described the basic types of agent behavior, we move now to analyze which behaviors have to be carried out by PWSSA, UWSSA and LWSSA respectively:

- **PWSSA behaviors:** As mentioned previously, the PWSSA waits for requests from the service provider to update the PBR with new service binding details. A possible design to achieve that is to make the PWSSA execute a one-shot behavior updating the PBR whenever the service provider change service bindings. Moreover, we need to make this agent execute another one-shot behavior forwarding updates to the UWSSA.

- **UWSSA behaviors:** Upon receiving new updates from the PWSSA, the UWSSA proceeds to save them into the PWSR first and then waits for requests from the LWSSA to serve them. A possible design to achieve this is to make UWSSA execute two cyclic behaviors. One dedicated to serve requests for updates reception and the other dedicated to serve requests for updates delivery.

- **LWSSA behaviors:** As described above, to forward updates of service bindings to the local repository of service consumer, the LWSSA periodically requests the UWSSA the new updated data being incorporated in the PWSR.
Therefore, to achieve this task we need to make the LWSSA execute a ticker behavior in which, on each tick, requests new updates from the UWSSA.

In fact behaviors of software agents are carried out upon receiving messages from other agents. Indeed, The format of messages exchanged between agents as specified by the ACL language [13] comprises a number of fields such as the sender of the message, the list of receivers, the communication intention (also called “performative”) and so forth. For the need of our synchronization model we extend this format with new fields. The description of these fields is shown in Table 1.

Table 1. Parameters of the updates message

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td>Denotes the Universal Unique Identifier (36 digits) of the service provider that originates the updates</td>
<td>uuid:d2033110-3aaf-11d5-80dc-002035229c64</td>
</tr>
<tr>
<td>USN</td>
<td>Denotes the Update Sequence Number with which the updates are tagged at their creation</td>
<td>55</td>
</tr>
<tr>
<td>update_Form</td>
<td>Denotes the form of updates intended by the service provider</td>
<td>ADD/Change/Delete/Hide</td>
</tr>
<tr>
<td>update_Entity</td>
<td>Denotes the UDDI entity within which the updates occur</td>
<td>businessEntity/businessService/ bindingTemplate / tmodel</td>
</tr>
<tr>
<td>update_Key</td>
<td>Denotes the UUID key of the updated element</td>
<td>uuid:894b5100-3aaf-11d5-80dc-002035229c64</td>
</tr>
<tr>
<td>update_Att</td>
<td>Denotes the updated attribute of the element</td>
<td>uddi:accessPoint</td>
</tr>
<tr>
<td>update_Value</td>
<td>Denotes the new value of the updated attribute</td>
<td><a href="https://tempuri.com/validateEmail.asmx">https://tempuri.com/validateEmail.asmx</a></td>
</tr>
</tbody>
</table>

According to the form of updates (update_Form parameter) some parameters of the updates message might be omitted when their value are useless. For example, if the service provider hides or deletes the access point of a particular Web service, therefore the update_Value parameter is not required in the message. Furthermore, some parameters are used in conjunction with other parameters to prevent software agents of some behaviors. For instance, if the PWSSA sends twice to the UWSSA an updates message with the same UUID and USN values therefore the latter agent will not accept this message as it was already received. Further details about the processing of updates message by software agents are provided in the next section.

4. Bindings Synchronization’ Usage Scenario

To verify the delivery status of email addresses, companies which relay upon email communications use an email validation Web service to keep list of addresses correct. In this section we use this sample of Web service to show how important it is to maintain its binding details accurate through business repositories to preserve its sustainable use.

In fact as mentioned previously to optimize the binding of such Web service, service consumers need to save its binding details locally within LWSRs for subsequent uses. Among these details we can distinguish; service access points, binding protocols, service operations, functional and non-functional parameters, service description and so forth. One of the most important binding details from the list above is the access point which enables service consumer to bind Web services from his LWSR without the need to access neither the service provider registry nor the PWSR. The access point is a network address, typically a URL, suitable for invoking the Web service.

However, if the service provider has activated a disaster recovery site, by activating new access point for the email validation Web service, most of the local binding calls from service consumers will fail when they try to invoke the Web service at the failed site. Therefore, based on our proposed synchronization model and by updating the service access point the PWSSA will automatically update the PBR of the service provider with the new address (e.g. https://tempuri.com/validateEmail.asmx).

To convey such updates to the PWSR where service consumers can get new bindings of the email validation Web service, the PWSSA sends an updates message to the UWSSA. This message includes in addition to the sender and
receiver identifiers and the communication intention which is in this case a CFP (call for proposal) performative, it includes also the new value of the access point (update_Value) along with a set of additional parameters, as depicted in Table 1, such as service provider identifier (UUID), updates register (USN), updates form (update_Form) and so forth. Upon receiving the updates message within its AMQ, the UWSSA checks whether this message with both the UUID and the USN values has been already received. Depending on the result of this checking, this agent may accept the message and save updates in the PWSR or refuse it. For our example, we assume that the updates message is first time received by the UWSSA and therefore this agent will accept it and store the new service access point in the PWSR while waiting for requests from the LWSSA to serve them.

Having a ticker behavior the LWSSA periodically requests, on each tick (e.g. every second), new updates from the UWSSA. Upon receiving new requests the latter agent servers them by sending the latest received updates message from his AMQ to the LWSSA. Given the fact that these updates may not interest the LWSSA, this agent may accept or refuse them. That is, if the local repository of service consumer already contains the UUID key (update_Key) of the updated access point, therefore the new updates will be accepted and the new address of the Web service will replace the failed one, otherwise the updates will be rejected. Finally, by saving the new service access point in the LWSR the service consumer may bind locally the email validation Web service from the new recovery site of the service provider.

5. Conclusion

In this paper, we have presented an agent-based approach for binding synchronization of Web services. This approach follows our previous work which introduced the architecture of a local repository-based framework for optimizing the binding of Web services. Within this framework, binding details of Web services of interest are saved locally in business repositories of the service consumers in a way to reduce time and effort of re-invoking these services.

However, since Web services are always changing (removed, updated, replaced, etc.) their binding details may need to change too. To address this issue, we have proposed in this work an agent-based synchronization model to update bindings of Web services whenever changes occur in order to keep their use in safe. In future work, we envision dealing with the implementation issues of the proposed model.

References