Web Service Communities: Concepts and Operations

Maamar, Zakaria; Benslimane, Djamal; Thiran, Philippe; Sattanathan, Subramanian

Published in:
International Conference on Web Information System and Technology

Publication date:
2007

Document Version
Early version, also known as pre-print

WEB SERVICES COMMUNITIES
- Concepts & Operations -

Zakaria Maamar[β], Mohammed Lahkim[α], Djamal Benslimane[γ], Philippe Thiran[δ], and Subramanian Sattanathan[δ]

[β]*Zayed University, Dubai, United Arab Emirates*

[α]*King Saud University, Riyadh, Kingdom of Saudi Arabia*

[γ]*Claude Bernard Lyon 1 University, Lyon, France*

[δ]*University of Namur, Namur, Belgium*

Keywords: Community, Contract-Net Protocol, Web Service.

Abstract: This paper discusses the concepts and operations related to the specification and management of a community of Web services, respectively. Web services offering the same functionality are gathered into one community, independently of their origins. The community is led by a master component, which is responsible among others for attracting new Web services to the community, retaining existing Web services in the community, and identifying the Web services in the community that will participate in composite Web services. The identification of these Web services is based on the contract-net protocol.

1 Introduction

For the W3C, a Web service *"is a software application identified by a URI, whose interfaces and binding are capable of being defined, described, and discovered by XML artifacts and supports direct interactions with other software applications using XML-based messages via Internet-based applications*". Nowadays, competition is not limited to goods, services, or software products, but includes systems that offer the current and accurate information. Web services are also not excluded from this competition. Providers develop several Web services that could offer similar functionalities like hotel booking and car rental. To ease and improve the process of Web services discovery in an open environment like the Internet, it is suggested to gather similar Web services[1] into groups known as communities (Benatallah et al., 2003; Medjahed and Bouguettaya, 2005). Although Web services are intensively investigated, the following community-related issues have not been addressed yet: how to initiate, set up, and specify a community of Web services, is the functionality of a Web service the only factor that drives the establishment of a community, and how to specify and manage the Web services that reside in a community?

It is understood that a community of Web services has a dynamic content: new Web services join, others leave, some become temporarily unavailable, etc. All these events need to be closely monitored, otherwise conflicts arise. For example, if a Web service was no longer a member of a community, its peers could assume that the Web service is still in the community. Moreover, Web services do not always expose a cooperative attitude when they join a community. Firstly, they can compete on computing resources, which may affect their performance scheduling. Secondly, they can announce misleading non-functional details to enhance their participation opportunities in composite Web services. Thirdly, they can become malicious when they try to alter other peers' data or behaviors.

In this paper, we do not aim at devising new strategies to select Web services of composite Web services. *We mainly aim at presenting the concepts and operations that are required to specify and manage a community of Web services, respectively*. Concepts and operations related to a community of Web services are discussed in Section 2. The prototype that simulates community management is presented in Section 3. Conclusions are drawn in Section 4.

2 Communities of Web services

2.1 Background

In Longman Dictionary, community is *"a group of people living together and/or united by shared interests, religion, nationality, etc*". When it comes to Web services, Benatallah et al. define community as a collection of Web services with a common func-

[1] Similar Web services and Web services with a similar functionality are interchangeably used.

tionality, although these Web services have distinct non-functional properties like different providers and different QoS parameters (Benatallah et al., 2003). Medjahed and Bouguettaya use community to cater for an ontological organization of Web services that share the same domain of interest (Medjahed and Bouguettaya, 2005). Our definition considers a community as a means to provide a common description of a desired functionality (e.g., `FlightBooking`) without explicitly referring to any specific Web service (e.g., `EKFlightBooking`).

2.2 Architecture

Fig. 1 is the architecture we designed to manage communities of Web services. The components in this architecture include providers of Web services, UDDI registries, and communities of Web services. A community is dynamic by nature. It is established and dismantled according to specific scenarios and protocols, which we discuss in Section 2.3. Two communities of Web services are shown in Fig. 1, which could for example offer `AirfareQuotation` and `HotelBooking` functionalities, respectively.

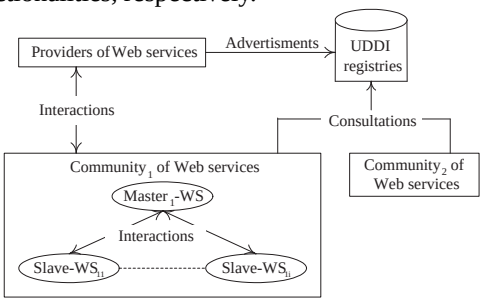


Figure 1: Architecture of Web services communities

The architecture in Fig. 1 has some features which we stress hereafter: (i) the traditional way of defining, announcing, and invoking Web services is still the same although these Web services are now elements of communities, (ii) the functionalities that UDDI registries usually offer to providers and end-users of Web services are still the same, and (iii) the selection of Web services from communities is transparent to users and occur independently of the way these Web services are gathered into communities. A master component always leads a community. The master component could itself be implemented as a Web service for compatibility requirements with other Web services in the community. These Web services are now denoted as slaves and have in common the functionality that labels the community in which they run. The relationship between master and slave Web-services relationship is regulated using the contract-net protocol (Smith, 1980) (Section 2.3).

One of the responsibilities of the master Web-service is to attract Web services to sign up in the community it heads using rewards (Section 2.3). As a result, the master Web-service interacts with the UDDI registries on a regular basis, so it gets to know the latest changes in the content of these UDDI registries such as advertisements of new Web services. Additional responsibilities of the master Web-service are (i) to nominate the Web service that will participate as component in a composite Web service, and (ii) to run the contract-net protocol before nominating this Web service.

In a community, the master Web-service is designated in two different ways. The first way is to have a dedicated Web service that will play the master role during the whole time being of a community. This Web service is independently developed (e.g., by application designer) from other Web services that are advertised in the UDDI registries. The Web service leading a community never participates in any composition. Therefore, this Web service is only loaded with community-management means. The second way is to identify a Web service out of the Web services that already populate a community. This identification could happen on a voluntary basis or after running election between the Web services. Because of the temporary no-participation restriction, the nominated Web service needs to be compensated by other peers. The call for elections among the Web services of a community regularly happens so, the burden on the same Web services leading a community is either minimized or avoided. Because Web services in the second way of designating a master Web-service need to be enhanced with extra functionalities that are specifically dedicated to community management (besides the functionalities these Web services offer), we develop an independent Web service to play the master role.

2.3 Operation

The operation of a community revolves around community development, Web services attraction and retention, and contract-net deployment.

Community development. A community of Web services is primarily established to gather the Web services that have the same functionality. This is a designer-driven activity that occurs in two steps. The first step is to define the functionality, e.g., `FlightBooking`, of the community by binding to a specific ontology. This binding is crucial since providers use different terminologies to describe the functionality of their respective Web services. For example, `FlightBooking` and `FlightReservation` are about the same functionality.

The second step in the establishment of a community is to deploy the master Web-service that leads the community and takes over the responsibilities we

listed in Section 2.2. One of these responsibilities is to invite Web services to sign up in its community by using rewards. It will be shown later that the survivability of a community depends, to a certain extent, on the status of the existing Web services in this community. Another responsibility of the master Web-service is to check the credentials of a Web service before this latter gets admitted in the community. The credentials could be related to QoS, protection mechanisms, etc. Credential checking can boost the security level within a community as well as enhance the trustworthiness level of a master Web-service towards the slave Web-services of its community. We recall that a master Web-service nominates the component Web services for participation in compositions.

Dismantling a community of Web service is also a designer-driven activity that happens upon request from the master Web-service. This latter oversees all the events in a community such as arrival of new Web services, departure of some Web services, identification of Web services to be part of composite Web services, sanctions on Web services due to misbehavior, etc. If the master Web-service notices that first, the number of Web services in the community is less than a certain threshold and second, the number of participation requests in composite Web services that arrive from users over a certain period of time is less than another threshold, the community could be dismantled. Both thresholds are set by the designer. The Web services that will be ejected from a community are invited to join other communities subject to assessing the functionality similarity with other existing communities' functionalities. Table 1 summarizes the role of both thresholds. For example, when the number of Web services in a community is "high" but the number of participation of these Web services in composition is "low", this means that the community has a poor configuration. To remedy to that configuration, some Web services with a low level of participation are ejected from the community and other Web services are invited to join the community.

Web services attraction and retention. Attracting new Web services to a community and retaining the existing Web services in a community fall under the responsibilities of the master Web-service. We discussed how a community could vanish if the number of Web services in this community drops below a certain threshold (Table 1). On one hand, attracting Web services drives the master Web-service to regularly consult the UDDI registries looking for new Web services. These latter could recently have been posted on an UDDI registry or have seen their description changed. Changes in a Web service's description raise challenges at the community level since a Web service may no longer be appropriate for a community. As a result, the Web service is invited to leave the community. When a candidate Web service is identified in an UDDI registry according to its functionality, the master Web-service engages in interactions with its provider (Fig. 1). The purpose of interactions is to ask the provider to register its Web service with the community of this master Web-service. Some arguments that are used during interactions include the high rate of participation of the existing Web services in composite Web services, the short response-time in handling user requests, etc.

On another hand, retaining Web services in a community for a long period of time is a good indicator of the following elements: (i) although the Web services in a community are in competition, they expose a cooperative attitude, and (ii) a Web service is to a certain extent satisfied with its participation rate in composite Web services. Web services attraction and retention shed the light on a third scenario, which concerns Web services being asked to leave a community. A master Web-service could issue such a request upon assessment of the following criteria: (i) the Web service has a new functionality, which does not perfectly match the functionality of the community, and (ii) the Web service is unreliable. In different occasions, the Web service failed to participate in composite Web services due to recurrent operation problems.

Contract-net deployment. In a community, interactions between the master Web-service and the slave Web-services are framed in accordance with the contract-net protocol (Smith, 1980). The main purpose of these interactions is to identify the slave Web-service that will take part in a composite Web service. The Contract-Net protocol (CN) is built upon the idea of contracting and subcontracting jobs between two types of agents known as initiator and participant. At any time an agent can be initiator, participant, or both. The sequence of steps in the contract-net protocol is as follows: (i) initiator sends a call for proposals out to participants in relation to a certain job to carry out; (ii) each participant reviews the call for proposals and bids if interested (i.e., feasible job), (iii) initiator chooses the best bid and awards a contract to that participant, and finally (iv) initiator rejects other bids.

Mapping the contract-net protocol onto the operation of a community of Web services occurs as follows. When a user (through some assistance) selects a community because it has the functionality that satisfies her needs, the master Web-service of this community is automatically contacted. The objective is to identify a specific Web service from the community, which will be in charge of implementing this functionality. The master Web-service sends a call for

Table 1: Community management

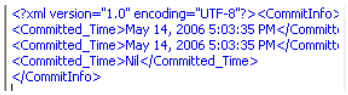
Number of Web services in		Comment	Recommended Action
A community	Compositions		
Small	High	Efficient configuration	Retain Web services and invite new ones
High	Small	Poor configuration	Eject Web services with low participation and invite new ones
Small	Small	Very poor configuration	Dismantle community
High	High	Desired configuration	Maintain same strategy

bids out to all the slave Web-services about the implementation of the functionality ($CN_{Step\ 1}$). The call for bids always comes along with the non-functional criteria that the user sets for selecting Web services like response time and execution cost. Prior to getting back to the master Web-service, the Web services assess their status (Maamar et al., 2006) and check their capacities of meeting these criteria ($CN_{Step\ 2}$). Only the Web services that are interested in bidding contact the master Web-service. This latter screens all the bids before choosing the best one ($CN_{Step\ 3}$, e.g., Web service's execution cost and reliability meeting the user's requirements). The winning Web service is then notified so, it can get ready for execution when requested ($CN_{Step\ 3}$). The rest of the Web services that expressed interest but were not selected, are also notified ($CN_{Step\ 4}$).

3 Prototype development

We developed a prototype to illustrate the contract-net protocol in a Web services community. The prototype uses XML for request and response specification between users and Web services and between master Web-services and slave Web-services, JDK 1.4 for operation performance, and Eclipse 3.1 as an integrated development environment. Currently, the prototype tracks in a community the interactions between a master Web-service and the slave Web-services with emphasis on the slave Web-service that will take part in a composition scenario. For illustration, we considered `WeatherForecast` functionality and a community of four slave Web-service plus the master Web-service. Each slave Web-service can only participate in two compositions at a time.

Initially, `Master-WeatherForecast-WS` leads `WeatherForecast` community. When it receives a user request, `Master-WeatherForecast-WS` broadcasts it to the members of the community. Upon receiving the request, each `Slave-WeatherForecast-WS` assesses its current commitments in terms of time slots prior to showing any interest in the new user-request to `Master-WeatherForecast-WS` (Fig. 2). For instance, $\texttt{Slave-WeatherForecast-WS}_1$ ignores the user request because of its double, ongoing commitments on `May 14, 2006 at 5:03:35 pm`. This does not apply to $\texttt{Slave-WeatherForecast-WS}_{2,3,4}$; they have no commitments so, they are willing to take over the user request.



(a) Weather-1

Figure 2: Commitments of $\texttt{Slave-WeatherForecast-WS}_1$

Once `Slave-WeatherForecast-WSs'` availabilities are known, `Master-WeatherForecast-WS` uses for example the first-response strategy to select a specific slave Web-service. Thus, $\texttt{Slave-WeatherForecast-WS}_2$ receives an accept message and $\texttt{Slave-WeatherForecast-WS}_{3,4}$ receive a not-accepted message. Once the messages are received, each `Slave-WeatherForecast-WS` updates its records.

4 Conclusion

In this paper, we presented our project on Web services communities. A community gathers Web services with similar functionalities. We addressed several aspects including establishing/dismantling a new/existing community, attracting new Web services to an existing community, retaining existing Web services in a community, and regulating the interactions inside a community using the contract-net protocol.

REFERENCES

Benatallah, B., Sheng, Q. Z., and Dumas, M. (2003). The Self-Serv Environment for Web Services Composition. *IEEE Internet Computing*, 7(1).

Maamar, Z., Benslimane, D., and Narendra, N. C. (2006). What Can Context do for Web Services? *Communications of the ACM*, 49(12).

Medjahed, B. and Bouguettaya, A. (March 2005). A Dynamic Foundational Architecture for Semantic Web Services. *Distributed and Parallel Databases*, 17(2).

Smith, R. (1980). The contract Net Protocol: High Level Communication and Control in Distributed Problem Solver. *IEEE Transactions on Computers*, 29.