A P2P IMPLEMENTATION FOR THE HIGH AVAILABILITY OF WEB SERVICES

Zakaria Maamar[α], Mohamed Sellami[γ], Samir Tata[γ] and Quan Z. Sheng[ε]
[α]*Zayed University, Dubai, U.A.E*
[γ]*Institut TELECOM, Evry, France*
[ε]*The University of Adelaide, Adelaide, Australia*

Keywords: Web service, Community, High availability, Peer-to-Peer.

Abstract: This paper introduces a P2P-based approach to sustain the high-availability of Web services using a similarity-based replication strategies. To this end three strategies known as active, passive, and hybrid, are studied. This approach takes replication one step further by focussing on Web services that offer the same functionality as the original Web service does (i.e., the one to back up). This functionality similarity is built upon communities that gather similarly-functional Web services. To prove the suitability of the selected replication strategy for Web services high-availability, a P2P testbed on top of the JXTA platform is developed.

1 INTRODUCTION

This research work discusses the high availability of Web services using the concept of communities. In a dynamic, open environment like the Internet, it is unlikely that applications built around software components such as Web services can be constantly available at run-time. Multiple reasons could make these applications break down, which triggers corrective plans execution to achieve business-operation continuity.

Despite the increasing popularity of Web services for the development of service-centric applications, there is still room for boosting this popularity by targeting *critical* systems (e.g., medical, nuclear) where **availability** is a central **concern**. Availability, as defined in (Avizienis et al., 2004), is an attribute of dependability that qualifies the readiness of an application. Traditional solutions to achieve availability are mostly based on replication (Juszczyk et al., 2006; Salas et al., 2006). Replication *"refers to the use of redundant resources, such as software or hardware components, to improve reliability, fault-tolerance, or performance. Replication typically involves replication in space, in which the same data is stored on multiple storage devices or the same computing task is executed on multiple devices, or replication in time, in which a computing task is executed repeatedly on a single device"* (Wikipedia Online Dictionary).

Although replication seems to be the trend in the development of highly-available Web services in the last few years (Juszczyk et al., 2006; Salas et al., 2006), this paper shows how we can substitute replica Web services (identical copies of the original Web service) with communities that host Web services semantically equivalent to the original Web service. In agreement with other definitions (Benatallah et al., 2003; Bentahar et al., 2007; Medjahed and Bouguettaya, 2005), we define community as a means to gather Web services with similar functionalities regardless of who developed these Web services, where these Web services are located, and how these Web services function. The high availability of applications built around Web services is then, ensured **not by** replicas of these Web services **but by** a community of Web services that are **similarly functional** to these Web services. Through the use of communities, several immediate benefits are obtained because of this shift in tackling the high-availability concern of Web services. For example, code management between replica Web services in case of changes, is no longer needed since these replica Web services have different codes. A second benefit is the possibility of screening communities when looking for similar Web services instead of traditional registries like UDDI. Although these benefits are appealing to any system developer, some of them are considered as obstacles in the existing replication strategies. For example, any minor code change in an application requires an immediate reflection of this change on all replicas.

Section 2 provides a short literature review on the topic of Web services high-availability. Section 3 summarizes the architecture and functioning of

Maamar Z., Sellami M., Tata S. and Z. Sheng Q. (2009).
A P2P IMPLEMENTATION FOR THE HIGH AVAILABILITY OF WEB SERVICES.
In *Proceedings of the 11th International Conference on Enterprise Information Systems - Software Agents and Internet Computing*, pages 19-24
DOI: 10.5220/0001855500190024

a community of Web services. Our approach based on communities for the high availability of Web services is detailed in Section 4. Section 5 presents some experimental details on the feasibility of our approach. Finally, Section 6 concludes the paper.

2 BRIEF LITERATURE REVIEW

Our literature review identified a good number of research projects, but a few promote the idea of using semantically-equivalent Web services to achieve this high-availability. In (Abraham et al., 2005), Abraham et al. report that little work on availability of Web services exists. As a result, Web services are automatically excluded from the technologies reserved for the development of mission-critical applications. Furthermore, Abraham et al. note that current specifications such as WS-ReliableMessaging and WS-Transaction do not support the availability of Web services. Abraham et al.'s approach suggests an enterprise level gateway and a Web service hub. The former operates at the enterprise level and has a monitoring role, while the latter operates across enterprises and has a selection role.

In (Juszczyk et al., 2006), Web services discovery, replication, and synchronization in ad-hoc networks are considered. This type of networks poses challenges to those who are in charge of supporting Web services high-availability due to dynamic topologies and unpredictable moves of hosts. As a result, when a Web service ceases to exist or changes its location, appropriate information need to be broadcasted to different recipients for instance UDDI registries. Juszczyk et al.'s solution consists of a discovery and registry system that feeds distributed UDDI registries with up-to-date information on available Web services, and a replication and synchronization mechanism that improves Web services reliability.

In (Laranjeiro and Vieira, 2007), the authors examine Web services availability from a fault-tolerance perspective. Their solution promotes the use of alternative Web services that are grouped by functionality. Each group is headed by an adapter (or proxy) that assesses the status of each member Web service in the group using metrics (response time, response correctness, etc.) before either simultaneously or sequentially invoking the alternative Web services. One of the issues that Laranjeiro and Vieira plan to address in the future is the lack of state consistency between alternative Web services in a group. We show in our approach how this lack of state consistency restricts the type of replication strategy to use.

In (Ribeiro et al., 2007), Ribeiro Jr. et al. propose smart proxies as a solution to access replicated Web services. These Web services are semantically equivalent and might be located in different and competing organizations or in the same organization. The smart proxies are adopted to encapsulate a variety of server-selection policies for selecting independent, autonomous Web services and to deploy adapters that bridge potential interface incompatibilities between Web services and clients.

3 COMMUNITY OF WSS

A community can be defined through the functionality of a representative abstract Web service (called master), i.e., without explicitly referring to any concrete Web service that will implement this functionality at run-time (Maamar et al., 2007). A community is established and dismantled with respect to some dedicated protocols. Moreover, a community has a dynamic nature; Web services enter and depart at their convenience. All this happens with respect to other protocols as well (Maamar et al., 2007).

In a community, a special Web service acting as a master Web service leads the community of slave Web services. Interactions between master and slave Web services and the designation of a master Web service outside this paper's scope. Some responsibilities of the master Web service include attracting Web services to the community it leads using rewards, convincing Web services to stay longer in the community, and identifying the Web services to participate in composition scenarios and to ***backup*** the functioning of their peers if needed. Extensive details about communities of Web services and their functioning are given in (Bentahar et al., 2007).

4 OUR APPROACH

Replication is the *de facto* option to tackle the high availability challenge of applications. Simply put, replication means (i) distribute copies of a software application over a network and (ii) make these copies back up the functioning of this application when problems arise. The way the original copy of the application and its replicas function relies on strategies known as *active*, *passive*, and *hybrid* (Wiesmann et al., 2000). For applications built around Web services, we propose in this paper substituting replicas with Web services extracted out of a community. We show that maintaining state-consistency between replicas and reflecting code changes over replicas are to a certain extent no longer needed in our

community-based high availability approach.

4.1 Failure Types and Detection

Several types of failures exist ranging from crashes where a Web service simply stops working to situations where a Web service delivers incorrect results. In this paper, we restrict ourselves to late-timing failures for the sake of illustration. To detect failures, several ways are identified such as test acceptance, watchdog, timeout, and voting (Kim, 2000). Among them two approaches could suit Web services: Watchdog and Timeout.

Since it is unreasonable that Web services generate signals for watchdogs, our approach combines the aforementioned techniques where signals are generated upon watchdogs' requests, i.e., a watchdog pings a Web service for liveness checking. We use timeout to avoid endless wait for feedback. Since watchdogs can be themselves subject to failures as well, our solution makes users act as watchdogs. Communications are supposed to be dependable, i.e., absence of signals' communications that Web services send to users acting as watchdogs are treated as Web services failures as well.

4.2 Replication Strategies Groundwork

The following points are parts of our community-based high-availability approach regardless of the replication strategy (active, passive, or hybrid) to use.

Control *vs.* Operational Flows. Using state charts (Harel and Naamad, 1996), we specify a Web service using two types of flow: control and operational (Fig. 1). Both flows interact with one another like Fig. 2 shows. The control flow describes the business logic that underpins the functionality of a Web service. And the operational flow guides the execution progress of the control flow of a Web service. Because state synchronization in some replication strategies is mandatory , our approach synchronizes only the operational flows of Web services in a community.

To illustrate the use of the control and operational flows, let us use *WeatherWS* as an example. Its functionality is to return a five-day weather forecast for a certain city. Fig. 2 shows how these flows interact together. Two types of transitions are shown: intra-flow (plain lines) and inter-flow (dashed lines). The initiation of *WeatherWS* is indicated with *activated* state in the operational flow.

Because of (*activated,city-located*) inter-flow transition, the execution of *WeatherWS* begins by searching for the requested city using a dedicated

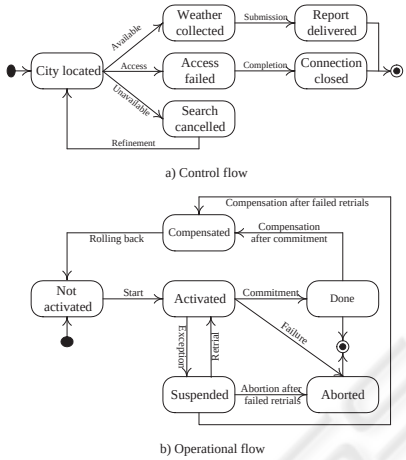

Figure 1: Control and operational flows of *WeatherWS*.

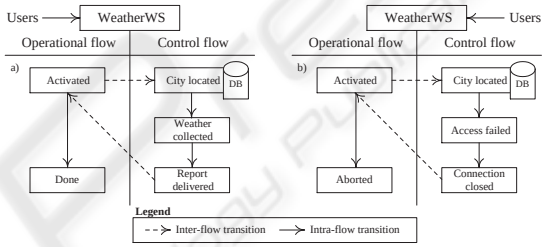

Figure 2: Control and operational flows in interaction

database. This now makes *WeatherWS* take on *city-located* state but this time in the control flow. After carrying out the necessary actions in this state, two cases are identified:

- In case a), everything goes fine and a five-day weather-forecast report is delivered back to the user. This makes *WeatherWS* complete its operation with success by transiting to *done* state in the operational flow.
- In case b), the access to the database fails. This makes *WeatherWS* terminate its operation with failure by transiting to *aborted* state in the operational behavior.

Those two cases illustrate how transitions in the operational flow of a Web service are affected by the states that this Web service binds to in the control flow.

Backup slave Web service. When a user selects a community because of the functionality that satisfies her needs, its respective master Web service is contacted. The master Web service has to identify a specific slave Web service that will implement this functionality. It sends all slave Web services a call for bids to express interest in implementing this functionality. The slave Web services in a community that positively responded to the call for bids are all notified

of their non-selection (except the one that is notified of its selection). The selected slave Web service will take over the role of *primary*, meaning that it will execute the functionality that the user needs.

The extra notification, for non selection, means that the unselected slave Web services might have to take over the role of *backup* to support the primary slave at run-time. These slave Web services can either accept or reject to act as a backup.

4.3 "Twisting" Replication Strategies

In this part of the paper, we discuss how the existing replication strategies can be either adopted or discarded due to the characteristics of the community-based high-availability approach.

4.3.1 *Active-replication strategy*

Active replication requires that all replicas receive and process the same sequence of users' requests in the same order. Result consistency across all replicas is guaranteed because requests are processed in a deterministic way. The main advantage of this strategy is its simplicity and failure transparency. If a replica goes down, the rest of replicas continue running. At the end, results are collected from one of them. The deterministic constraint is the major drawback in this strategy (Wiesmann et al., 2000).

Fig. 3 uses *WeatherWS* to illustrate the interactions in the active-replication strategy. Plain lines correspond to the interactions that take place between users and Web services and between the operational and control flows of *WeatherWS* (primary and backup) and dashed lines correspond to the interactions that take place between the operational flows of *WeatherWS* and backup *WeatherWSs*. Dashed line (1) illustrates *WeatherWS* that forwards a user's request to its backups and initiate their execution. The master Web service provides the list of backup slave Web services to the primary slave. Afterwards, all *WeatherWSs* concurrently process this request. If the primary *WeatherWS* fails the other backup slave Web services continue running. In addition, one of them needs now to be selected to act as a primary so that it can deliver responses to the user.

4.3.2 Passive-replication Strategy

Passive replication suggests that users submit their requests directly to a specific application (usually the original copy), which is the primary backup. This application processes users' requests and regularly sends the rest of backup applications update requests during processing. These latter are on standby waiting to receive these update requests to implement them in order to get their respective behaviors synchronized with the original application's behavior. This strategy is unsuitable for Web services with respect to our approach. The discrepancies in the control flows that exist between the primary backup and other backups prevents the correct execution of this replication strategy. Indeed, actions that the primary backup carries out, can not match the actions that backups have to carry out as well. In addition, when a primary backup fails, the control flow of the new elected primary backup can not be updated because of the disparity with the old primary backup control flow.

4.3.3 Hybrid-replication strategy

Hybrid replication handles the deterministic constraint of the active replication by referring indeterministic decisions to a specific component called leader. The leader makes the choice among several available ones and sends it to all followers. In our approach, indeterminism exists by default since all Web services are expected to return different results. As a result and like the passive-replication strategy, the hybrid-replication strategy is not suitable for Web services with respect to our approach.

5 EXPERIMENTS

To illustrate the feasibility of our Web services high-availability approach using communities, we deployed a P2P based-testbed on top of Sun Microsystems's JXTA platform (Traversat et al., 2003). JXTA is an open source P2P technology that permits simulating virtual overlay networks and grouping peers according to some common interests that these peers express. It was enriched with services known as JXTA services and offers a wide range of predefined P2P facilities including service/peer advertisement/discovery and messaging. The JXTA platform is suitable for implementing our proposed approach; it enables the handling of the dynamic nature of communities like Web services arrival and departure. A community is then viewed as a peer group where each Web service corresponds to a JXTA peer.

Our experimental work implements the active replication strategy. Here, a client peer interacts with a group of peers, i.e., a community, that hosts a set of peers, i.e., Web services. Three JXTA peers have been developed: *MasterServicePeer* acting as a community master, *SlaveServicePeer* acting as community slave, and finally *ClientPeer* acting as a user who interacts with the *MasterServicePeer* to invoke the

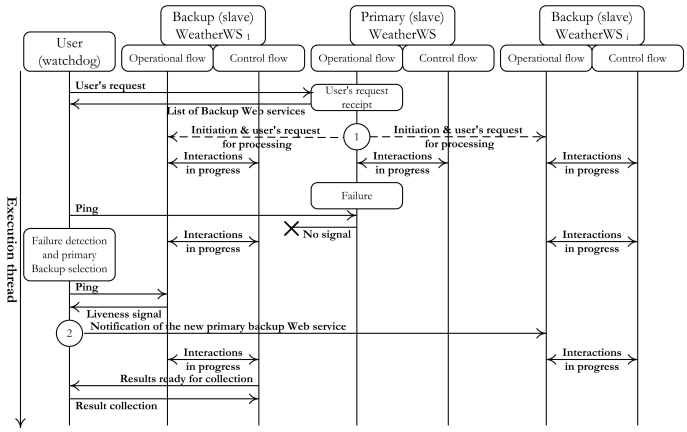

Figure 3: Interactions in active-replication strategy

functionality of a *SlaveServicePeer*. The aforementioned JXTA peers have been deployed over various distributed platforms. Clients' queries to a community are routed to Web services via their representative JXTA peers, creating thus a virtual network on top of the physical network (Fig. 4).

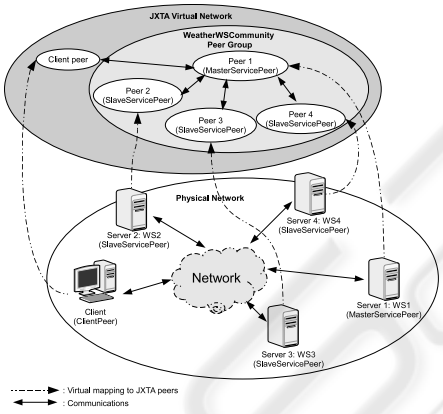

Figure 4: Implementation of our high-availability approach

In our experiments, we created a *MasterServicePeer*, two *SlaveServicePeer*(s) that belong to the community and connected to Web services, and a *ClientPeer*.Fig. 5 uses some screenshots to illustrate the interactions between the peers at run-time. In this figure, the different interactions are numbered. For illustration purposes, *SlaveServicePeer*(s) provide connection to Web services offering weather forecast for a city submitted by a client. The master service peer and slave service peers belong to the same community (JXTA peer group), which we denote by *WeatherWS-Community*. When a client peer contacts the master service peer (Fig. 5, interaction *(i1)*), this one assigns a slave service peer among the available slave service peers *(i2-i3)* and tags the first slave as primary and the rest as backups. This assignment and tagging happen in compliance with the description we provided in Section 4.2. The master service peer sends the name of this primary slave service peer to the client peer *(i4)*. Afterwards, the client peer interacts with the primary slave service peer *(i5)* (i.e., sends "Paris" as input). When the client peer request is received, the primary service peer initiates the backup slave service peers as well so that all slaves are now running in parallel *(i6)*. The names of the initiated backup slave service peers are sent to the client peer *(i7)*. To implement a failure detection mechanism, we made the client peer acts as a Watchdog; it regularly pings the primary slave service peer for the sake of testing its availability *(i8)*. We trigger a failure by simply stopping the primary slave service peer. Upon failure detection due to lack of response, a slave service peer from the backup slaves is elected by the client peer as the new primary slave *(i9)*. The client peer contacts the first slave service peer from the backup slaves list previously received from the ex-primary slave service peer. If the first slave service peer responds, it will be considered as the new primary slave service peer. When it completes its execution, the first slave service peer submits its results to the client peer *(i10)* (i.e., sends the weather forecast of "Paris").

Currently, our Web services are deployed on Web service containers for instance AXIS and JXTA peers bind and convey requests to them. In term of testbed improvement, we are working on using JXTA-SOAP (Amoretti et al., 2008) so that JXTA peers pass on requests to Web services.

6 CONCLUSIONS

In this paper, we looked into the high-availability issue of Web services and put forward solutions that

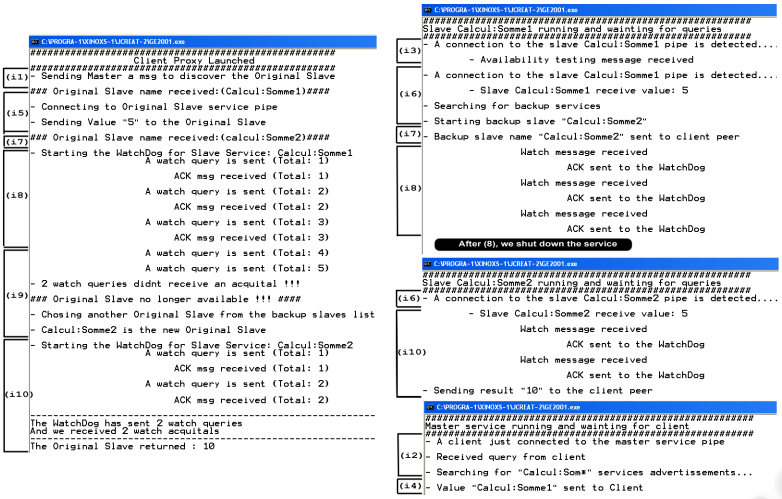

Figure 5: Some screenshots captured at run-time.

promote communities. We proposed a replication based approach for sustaining Web services high availability where replicas are Web services extracted out from a community. Putting the active, passive, and hybrid replication strategies in the context of communities revealed the suitability of the first strategy only. We specified the functioning of similarly-functional Web services using control and operational behaviors. We also demonstrated and deployed the active-replication strategy on top of a JXTA-based tested we developed.

Our future research work will focus on continuing the testbed we started and investigating various issues in terms of (1) how much transactional properties impact the high-availability requirement of Web services from a community perspective, (2) how to assess the semantic impact of replacing a failed Web service with a backup Web service on the progress of a composition scenario in which the failed Web service was taking part and (3) how to select within a community a backup Web service that could maintain the same level of QoS that the failed Web service and/or user requirements in terms of QoS.

REFERENCES

Abraham, S., Thomas, M., and Thomas, J. (2005). Enhancing Web Services Availability. In *ICEBE'2005*.

Amoretti, M., Zanichelli, F., Conte, G., and Bisi, M. (2008). Enabling peer-to-peer Web service architectures with JXTA-SOAP. In *e-Society'08*.

Avizienis, A., Laprie, J. C., Randell, B., and Landwehr, C. (2004). Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1).

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

Bentahar, J., Maamar, Z., Benslimane, D., and Thiran, P. (2007). Using Argumentative Agents to Manage Communities of Web Services. In *WAMIS'2007*.

Harel, D. and Naamad, A. (1996). The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4).

Juszczyk, L., Lazowski, J., and Dustdar, S. (2006). Web Service Discovery, Replication, and Synchronization in Ad-Hoc Networks. In *ARES'2006*.

Kim, K. H. (2000). Issues Insufficiently Resolved in Century 20 in the Fault-Tolerant Distributed Computing Field. In *SRDS'2000*.

Laranjeiro, N. and Vieira, M. (2007). Towards Fault Tolerance in Web Services Compositions. In *EFTS'2007*.

Maamar, Z., Lahkim, M., Benslimane, D., Thiran, P., and Sattanathan, S. (2007). Web Services Communities - Concepts & Operations -. In *WEBIST'2007*.

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

Ribeiro, J. J. G., do Carmo, G. T., Valente, M. T., and C., M. N. (2007). Smart Proxies for Accessing Replicated Web Services. *IEEE Distributed Systems Online*, 8(12).

Salas, J., Pérez-Sorrosal, F., Patiño Martínez, M., and Jiménez-Peris, R. (2006). WS-Replication: A Framework for Highly Available Web Services. In *WWW'2006*.

Traversat, B., Arora, A., Abdelaziz, M., Duigou, M., Haywood, C., Hugly, J.-C., Pouyoul, E., and Yeager, B. (2003). Project JXTA 2.0 Super-Peer Virtual Network. Technical report, Sun Microsystems.

Wiesmann, M., Pedone, F., Schiper, A., Kemme, B., and Alonso, G. (2000). Understanding Replication in Databases and Distributed Systems. In *ICDCS'2000*.