Towards a Coordination Model for Web Services

Zakaria Maamar[1], Nanjangud C. Narendra[2] and Philippe Thiran[3]

[1] Zayed University, U.A.E
[2] IBM India Research Lab, India
[3] University of Namur, Belgium

Abstract. The increasing popularity of Web services for application integration has strengthened the need for automated Web services composition. For this automation to succeed, the joint execution of Web services requires to be coordinated. Coordination's main use is to solve conflicts between Web services. Conflicts could be on sharable resources, order dependencies, or communication delays. The proposed coordination model tackles these conflicts with three inter-connected blocks defined as conflict, exception, and management. Conflicts among Web services raise exceptions that are handled using appropriate mechanisms as part of the coordination model.

1 Introduction

Over the last few years, the development pace of Web services has been impressive [4]. Several standards have been developed and several projects have been initiated. Some standards concern Web services definition, discovery, and security, and some projects concern Web services composition, personalization, and provisioning. In general, composition of Web services primarily addresses the situation of user requests that cannot be satisfied by any single Web service, whereas a composite Web service obtained by combining available Web services may be used.

Another research venue that is worth pursuing is the coordination of Web services so first, their collective actions are performed in a coherent way and second, their exceptions are handled in a proper way. Although the WS-Coordination specification exists, it does not emphasize the conflicts that could arise between Web services. Web services do not always expose a cooperative attitude when they participate in compositions. For instance, they can compete on sharable resources, which may affect their performance scheduling. Second they can announce misleading information (e.g., QoS) to enhance their participation opportunities in composite Web services. In addition, they can be kept on-hold for long time periods due to data or order dependencies with peers. Our objective is to investigate the way Web services coordination happens over the following three steps: discovery, engagement, and performance. The three steps denote here what we call Web services composition.

It is known that coordination's main use is to solve conflicts between separate components, for example Web services. Depending on the type of conflict, a centralized or distributed (e.g., peer-to-peer) form of coordination can be adopted. In this paper we study both forms of coordination along the discovery, engagement, and performance steps and discuss which form suits which step. Each form of coordination has its pros

Maamar Z., C. Narendra N. and Thiran P. (2006).
Towards a Coordination Model for Web Services.
In *Proceedings of the 1st International Workshop on Technologies for Collaborative Business Process Management*, pages 24-32

and cons. In term of pro, centralized coordination makes the identity of the components only available to the entity in charge of coordination. This is crucial when components want to remain anonymous. For decentralized coordination, the different components know each other so they can reach mutual agreements without going through third parties. In term of con, centralized coordination heavily relies on the normal operation of the entity in charge of coordination, which could turn out to be a bottleneck for this operation. For decentralized coordination, there is no overall picture of the way coordination progresses among the different components. The rest of this paper is organized as follows. We overview some concepts and present a running scenario in Section 2. Sections 3 and 4 present the proposed coordination model for handling exceptions of Web services. The paper concludes in Section 5.

2 Background

2.1 Definitions

The decomposition of Web services composition into three steps namely discovery, engagement, and performance is inline with Burstein et al.'s steps [2]. These steps are to a certain extent run in parallel and thus, require to be monitored for reasons of performance and concurrence. Another conflict could arise while the current conflict is being fixed, unless the execution of all component Web services - including those Web services that are not affected by the current conflict - is suspended.

The discovery step is about identifying the Web services that satisfy users' needs based on user-specified selection criteria [1]. This discovery can be implemented by the use of an UDDI registry on which Web services descriptions are posted so potential users consult these descriptions.

The engagement step is about connecting the component Web services that were identified in the discovery step. The connection, usually known as orchestration, complies with a specification that underlines a business logic (e.g., travel planning). Interesting to stress that Web services interaction highlights messages to exchange, data to supply, acknowledgments to return, dependencies to manage, etc.

The performance step is about running the component Web services on top of resources. Scheduling the execution requests of Web services is prioritized when enough resources are not available to satisfy these requests all at once. A Web service requires resources for different operations like self-assessment prior to participating in compositions, satisfying user needs upon request, and data exchange with other Web services.

2.2 Coordination Form

Coordination can take two different forms: centralized or distributed. The first form of coordination calls for a central component that is in charge of providing a unified model of the coordination, overseeing the operation of other components, interfering in case of conflicts, and restarting execution after exception handling. The second form of coordination calls for a mutual awareness of the components and their respective operations. This alleviates the burden of designing a single coordination component

that will be bound to monitor other components. The components engage in conflict resolution, and cooperate with each other to determine appropriate exception handling procedures. In section 3.2, we discuss the interleaving of coordination and interaction. Some pros and cons of each form of coordination have been discussed in Section 1.

In addition to both forms of coordination, coordination could be either implicit or explicit. On the one hand implicit coordination assumes that participants are aware of some existing pre-defined rules, which they need to abide by. If the contrary happens participants are subject to penalties that depend on the application domain. Traffic regulation is a good example of implicit coordination, where fines sanction the drivers who do not conform to the regulation's policies. On the other hand explicit coordination requires negotiation, voting, or intervention of an authority. This coordination calls for a clear specification of various elements including tasks that are executed, conflicts that can arise, and exceptions that can happen.

2.3 Running Example

Our running scenario is about Amin who is visiting Melissa back in her home city, Oslo. Amin and Melissa agree to meet in a coffee shop, not far from Melissa's office since she finishes work late on that day. Amin has two options to reach the meeting place: by taxi or by bus. Figure 1 illustrates the specification of Amin scenario using a combination of state chart diagrams and service chart diagrams [5].

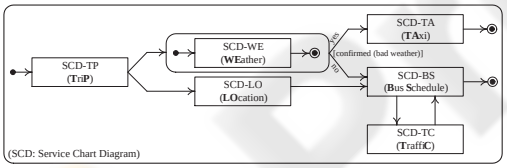

Fig. 1. Specification of Amin scenario.

In case Weather WS forecasts bad weather, a taxi booking is made on behalf of Amin using Taxi WS. Otherwise, i.e., pleasant day, Amin uses public transportation. The location of both Amin's hotel and coffee shop are submitted to Bus Schedule WS, which returns for example the bus numbers Amin has to take. Potential traffic jams force Bus Schedule WS to regularly interact with Traffic WS that monitors the status of the traffic network. This status is fed into Bus Schedule WS so adjustments to bus numbers and correspondences between buses can occur.

Amin scenario yields insight into the multiple challenges that Web services coordination faces. Some of these challenges include: what are the reasons that trigger conflicts between Web services, how to model and track Web services coordination, how to adjust control and data flow among the involved Web services during coordination, which form of coordination, whether centralized or distributed, to adopt according to the progress of Web services composition, how to ensure that Web services comply with an implicit or explicit coordination, and what types of penalties could be developed for the non-complying Web services?

3 Proposed Coordination Model

3.1 Foundations and Operation

For Papazoglou and Georgakopoulos, coordination is ”to control the execution of the component services, and manage dataflow among them and to the output of the component service” [7]. While there is no disagreement on this definition, the way exceptions hinder a composition progress and affect its execution control and dataflow management is not strengthened. Figure 2 presents our proposed coordination model for handling Web services exceptions. The model is made up of 3 blocks: conflict, exception, and management. These blocks form a cycle that starts with conflict and continues next with exception then management, before it returns back to conflict. The coordination model runs along the discovery, engagement, and performance steps of Web services composition. Each step could host the execution of the complete cycle of the three blocks, which calls for a close follow-up of the awareness, handling, and monitoring transitions between these blocks (Figure 2).

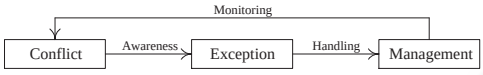

Fig. 2. Proposed coordination model to handle Web services exceptions.

Conflict block is concerned with the problems that preclude the normal operation of Web services as per the composition specification (Figure 1). These problems vary according to the step that features a composition progress. Indeed conflicts during discovery are different from those during engagement or performance. Examples of conflicts per type of step are given later. In the conflict block, we identify the following actions: (1) *conflict detection* for knowing that a conflict is happening is deemed appropriate so actions can be taken; (2) *conflict classification* that identifies the type of conflict whether related to resource, communication failure, etc; (3) *conflict impact assessment* that evaluates how much a conflict impacts the normal progress of composition. This impact is in terms of time duration, execution cost, etc; (4) *conflict correction* that aims at suspending a composition progress, giving room for corrective actions to be executed. This will manifest itself in the exception block.

Exception block follows the conflict block and aims at gearing the computational efforts to put in towards the detected type of conflict. Each conflict has to be separately treated from the specification of composition, which shows the importance of concern separation between exception handling and composition specification execution. In the exception block, we identify the following actions: (1) *exception establishment* that confirms the presence of a conflict by labelling the specification as abnormal. This is derived from the conflict correction action in the conflict block; (2) *exception information propagation* that defines the exception in terms of current active step in specification, participating Web services, pending messages, etc. In addition, the information regarding the exception is propagated to all the participating Web services. This propagation is detailed in Section 3.3.

Management block follows the exception block and aims at fixing the conflict that is associated with an exception, i.e., running exception handling. Management adopts a centralized or distributed form of coordination according to the exception type and the active step of composition. In the management block, we identify the following actions: (1) *management initiation* that begins the coordination work by selecting the appropriate conflict solving strategy according to details obtained out of the exception block.(2) *management tracking* that aims at overseeing the performance of the selected solving strategy in terms of executed actions, fixed conflicts, exchanges messages, etc; *management outcome validation* that permits to finalize the solving strategy by confirming its success or failure and to give back the control to the conflict block.

We stated earlier that the proposed coordination model (Figure 2) runs along the discovery, engagement, and performance steps that make up Web services composition. During Web services discovery, three requirements of type language, function, and architecture need to be satisfied [2]. If this does not happen, conflicts will arise. Language requirements concern expressing the capabilities of Web services and goals of requestors. A potential conflict could be lack of understanding of these capabilities or goals. Functional requirements concern the actions that providers, requestors, and matchmakers will perform. A potential conflict could be the non-performance of the expected actions or the non-compliance with the actions each entity is supposed to perform. Finally, architectural requirements define the advertisements and discovery protocols to be used by Web services providers and requestors, respectively. A potential conflict could be the use of an unknown advertisement or discovery protocol.

During Web services engagement, interactions between Web services providers and requestors take place and result in agreements [2]. Similar to Web services discovery's requirements, the engagement step is featured with functional and architectural requirements. If these requirements are not satisfied, conflicts will arise. Functional requirements concern Web services request formulation, contract preliminaries, contract negotiation, and negotiation. A potential conflict could be differing expectations between service requestor and provider regarding terms of the contract. Architectural requirements concern negotiation protocols, negotiation services, and auditing services. A conflict could be the non-compliance with the commitments made in a contract.

During Web services performance, a.k.a enactment and management in [2], requirements are of types function and architecture. If these requirements are not satisfied, conflicts will arise. Functional requirements concern multiple aspects like choreography interpretation and execution, service-failure handling and compensation, and non-repudiation. A potential conflict could be differing compensation strategies among participating Web services. Architectural requirements concern also multiple aspects like process-scheduling and composition services and policy-monitoring services. A potential conflict could be non-compliance with agreed upon QoS requirements.

3.2 Interleaving Interaction and Coordination

In a Web services composition scenario, the flow of interaction happens in a vertical (between a composite Web service and its component Web services) and horizontal (between component Web services of the same composite Web service) way. Through an interaction, the initiator aims at conveying some information to the recipient, so this

latter can for instance take actions and adapt its behavior, consequently. In the following we discuss the way interaction and coordination are interleaved per way of interaction. This discussion uses Figure 3 and Figure 4. In both figures, plain lines correspond to interactions and dotted lines correspond to conflict detection and resolution operations.

In vertical interactions, a composite Web service has the authority to execute the following actions over a component Web service (Figure 3): invite in order to join composition, trigger in order to initiate execution, audit in order to track performance, and retract in order to replace component (e.g., due to poor performance or temporary unavailability). It is shown in Figure 3 that trigger action implements a centralized orchestration of Web services, which is used in systems like CMI [8]. A centralized orchestration assumes that the connection between the central scheduler of the composite Web service and the component Web services is continually available. It is also shown in this figure that retract action is followed by invite action, so a suitable replacement of the retracted component Web service from the composition is identified and appended into this composition.

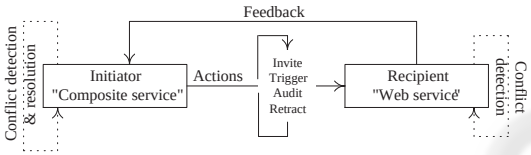

Fig. 3. Interleaving interaction and coordination during vertical interactions.

Coordination in vertical interactions occurs at the composite Web service level. Conflict detection that triggers coordination is carried out in two different places. The first place highlights a component Web service that faces difficulties in completing its operations. A Web service could be put on hold for a long period of time due to occupied resources. As a result, the Web service notifies the composite Web service so appropriate resolution actions can be taken. The notification is represented with feedback in Figure 3. The second place of coordination highlights a composite Web service, which based on the feedbacks it receives from its component Web services, can expect the occurrence of conflicts, such as possible bus scheduling conflicts (caused by traffic jams) in Amin scenario. Thus, the composite Web service decides to take actions prior to conflict occurrence. This is called preventive strategy to conflict occurrence.

In horizontal interactions, a Web service has the authority to carry out the following actions over another peer engaged in the same composition (Figure 4): trigger in order to initiate execution and monitor in order to check peer's liveness. It is shown in Figure 4 that trigger action implements a peer-to-peer orchestration of Web services, which is used in systems like PCAP [9]. It is also shown in this figure that monitor action is followed by trigger action to ensure that a Web service has effectively been triggered. There is no guarantee that a particular Web service is still available at time of request.

Coordination in horizontal interactions occurs at the component Web services level. Conflict detection that triggers coordination is also carried out at the same level. When a component Web service identifies a conflict, it interacts with the component Web services that are part of this conflict. In Amin scenario, bus WS may interact with Traffic

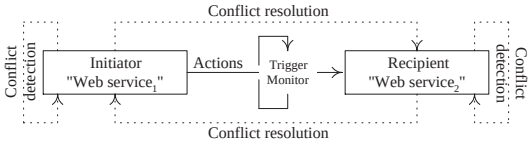

Fig. 4. Interleaving interaction and coordination during horizontal interactions.

WS and Location WS regarding potential traffic jams. The number of these components varies from one to many, which can increase the complexity of resolving conflicts.

It should be noted that triggering action here does not mean that the Web service can only perform one operation; it is assumed that the trigger action would specify the operation needed to be performed. In addition to the aforementioned conflict types, semantic conflicts could be considered [3]. However they are not discussed in this paper.

3.3 Adaptation and Propagation During Exception Handling

As per Narendra et al.'s classification of the tasks in a workflow, a task can fall into one of the following categories [6]: pivot, retriable, or compensatable. We bind to the same classification to our coordination model and consider that a Web service could be:

- Pivot: once the Web service fails, it is neither retried nor compensated. This Web service can only be aborted, not even rolled back. Committing a pivot Web service for execution means that the execution of the entire composition specification needs to be completed. A pivot-Web-service failure means failure of the composition specification, which will need to be aborted and restarted from scratch. In Amin scenario, Trip WS committing to assist Amin in preparing his trip to the coffee shop, is an example of pivot Web service.
- Retriable: the Web service can be retried upon failure, but cannot be compensated. This Web service can also not be rolled back if the retry also fails, and thus, can only be aborted. In Amin scenario, Taxi WS that books a taxi for Amin reports failure after retrying a number of times.
- Compensatable: the Web service is retriable and thus, can be rolled back via its corresponding compensating Web service. In Amin scenario, if the meeting location is changed this will result in forcing Location WS to do some compensation.

Web services in conflict are either rolled back or aborted in the reverse order in which they executed. Two possibilities illustrate the way the reverse order occurs:

1. Web service is retriable. If the retry succeeds, then the traversal stops and the execution of the composition specification resumes. Otherwise, the Web service is aborted and the control moves backward to the previously executed Web service.
2. Web service is compensatable. If the retry succeeds, then the traversal stops and the composition specification resumes. Otherwise, the compensating Web service for the Web service is invoked, and the control moves backward to the previously executed Web service.

4 Illustration via the Running Scenario

Referring to the running scenario in Figure 1, we suggest hereafter three possible conflicts. We assume that Trip WS is of type pivot, Location WS is of type compensatable, and the rest of Web services are of type retriable.

First, taxi scheduling conflict that impedes Amin's ability to take a taxi due to bad weather. Here Taxi WS can retry the execution so an alternate taxi is provided to Amin. In case the retrial fails, Taxi WS will need to simply abort and report failure, forcing Amin to check Bus schedules despite the bad weather.

Second, bus scheduling conflict resulting from traffic jams on the way to the coffee shop. The occurrence of this conflict is due to traffic jams, which forces Bus schedule WS to provide an alternate schedule via a retry. In case this also results in conflict, Bus schedule WS is aborted. Afterwards, Location WS needs to be rolled back via its compensating Web service, and again re-executed to assign a new location for Amin & Melissa to meet.

Third, Bus scheduling conflict because of erroneous location. In case Location WS itself returns an erroneous location, Bus schedule WS needs to be aborted, and Location WS needs to be rolled back using its compensating Web service. Afterwards, Location WS needs to re-execute so it can provide the correct meeting location. This adaptation process can either be coordinated by Trip WS (vertical interactions) or can be self-coordinated by the conflicting Web services themselves (horizontal interactions).

5 Conclusion

In this paper we presented a coordination model for Web services with emphasis on handling exceptions during execution. The model encompasses three blocks: conflict, exception, and management. They are concerned with, respectively, conflict detection and classification; exception derivation and information propagation; and tracking of exception handling. We also discussed the way the coordination model can be implemented in either a centralized or distributed manner.

References

1. V. Agarwal, G. Chafle, K. Dasgupta, N. Karnik, A. Kumar, S. Mittal, and B. Srivastava. Synthy: A System for End To End Composition of Web Services. *Journal of Web Semantics*, 3(4), 2005.
2. M. Burstein, C. Bussler, M. Zaremba, T. Finin, M. N. Huhns, M. Paolucci, A. P. Sheth, and S. Williams. A Semantic Web Services Architecture. *IEEE Internet Computing*, 9(5), September/October 2005.
3. J. Cardoso and A. Sheth. Semantic Web Processes: Semantics Enabled Annotation, Discovery, Composition and Orchestration of Web Scale Processes. In *Proceedings of The 4th International Conference on Web Information Systems (WISE'2003)*, Roma, Italy, 2003.
4. S. Dustdar and W. Schreiner. A Survey on Web Services Composition. *International Journal on Web and Grid Services*, 1(1), 2005.

5. Z. Maamar, B. Benatallah, and W. Mansoor. Service Chart Diagrams - Description & Application. In *Proceedings of The Alternate Tracks of The Twelfth International World Wide Web Conference (WWW'2003)*, Budapest, Hungary, 2003.
6. N. C. Narendra and S. Gundugola. Automated Context-Aware Adaptation of Web Service Executions. In *Proceedings of The 4th IEEE Conference on Computer Systems and Applications (AICCSA'2006)*, Sharjah, U.A.E, 2006.
7. M. Papazoglou and D. Georgakopoulos. Introduction to the Special Issue on Service-Oriented Computing. *Communications of the ACM*, 46(10), October 2003.
8. H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and Composing Service-based and Reference Process-based Multi-Enterprise Processes. In *Proceedings of The 12th International Conference on Advanced Information Systems (CAiSE'2000)*, Stockholm, Sweden, 2000.
9. Q. Z. Sheng, B. Benatallah, Z. Maamar, M. Dumas, and A. H. H. Ngu. Enabling Personalized Composition and Adaptive Provisioning of Web Services. In *Proceedings of The 16th International Conference on Advanced Information Systems (CAiSE'2004)*, Riga, Latvia, 2004.