Web Services Synchronization in Composition Scenarios

Hamdi Yahyaoui[1], Zakaria Maamar[2], Jamal Bentahar[3], and Khouloud Boukadi[4]
[1]KFUPM, Dhahran, KSA —- [2]Zayed University, Dubai, U.A.E —– [3]Concordia University, Montreal, Canada—–
[4]École des Mines, Saint-Etienne, France

Abstract

This paper discusses Web services synchronization at the composition level. Synchronization aims at assisting independent parties coordinate their actions and thus, avoid conflicts. Our previous work on synchronization primarily focused on the component level and shed the light on two types of behaviors related to specifying Web services. The control behavior defines the business logic that underpins the functioning of a Web service, and the operational behavior regulates the execution progress of this control behavior by stating the actions to carry out and the constraints to put on this progress. Control and operational behaviors continue to be used to specify composite Web services with respect to the orchestration schemas that these composite Web services have to comply with whether centralized or peer-to-peer. As a result, various types of messages to achieve synchronization are developed per type of orchestration schema. Experiments showing the use of these messages are reported in this paper as well.

Keywords. *Composition, Synchronization, Web service.*

1 Introduction

In [8] and [9], we investigated the synchronization issue of ("isolated") Web services independently of the composition scenarios in which these Web services could take part. This investigation shed the light on two types of behaviors namely *control* and *operational* that were both used to specify and exhibit the functioning of Web services. The control behavior illustrates the business logic of the functionality, e.g., `CarRental`, of a Web service, while the operational behavior frames the progress of executing the business logic of this Web service at run time.

As the literature review points out [3], it is known that the "beauty" of Web services resides in their capacity to be composed into high-level business processes known as composite Web services. Composition is suitable for users' requests that cannot be satisfied by any single, available Web service, whereas a composite Web service obtained by combining available Web services might be used. Composition design and development are bound to a specification that describes, at design time, multiple elements such as execution order of component Web services, data dependencies between component Web services, and corrective strategies in case component Web services raise exceptions. At run time, the composition specification is triggered, which means identifying and invoking component Web services, overseeing their execution, coordinating their actions, and initiating corrective strategies if needed. Different specifications related to Web services composition currently exist such as BPEL (*de facto* standard) and WSCI.

In term of execution, Web services composition can be structured along two types of orchestration [1]: *centralized* or *peer-to-peer* (P2P) (i.e., decentralized). On the one hand, centralized orchestration like its name hints relies on a centralized module (e.g., BPWS4J) that coordinates and tracks all the execution activities related to component Web services in terms of when to invoke them, what to expect out of their invocation, what data they exchange, how to pass on these data, just to cite a few. On the other hand, P2P orchestration excludes the centralized module and promotes direct interactions between component Web services. This makes Web services aware of some of their direct acquaintances during composition, which means the necessity of empowering these Web services with appropriate knowledge and mechanisms in order to support direct interactions. eFlow [2] is an example of Web services-based systems that adopts a centralized orchestration, whereas PCAP [10] is an example of Web services-based systems that adopts a P2P orchestration.

This paper extends the synchronization initiative we report in [9] by leveraging this time our research findings and thoughts from the component to the composition levels. In [5], we applied the separation between control and operational behaviors to model check orchestration-based composite Web services. The control behavior is used to extract the desired properties to be checked in the model of composite Web services captured by the operational behavior. In this paper, our primary objective is to address the following issues per type of orchestration: what synchronization mechanisms are required to set up, what messages implement these mechanisms, how these messages are tracked during synchronization, how synchronization and execution are interleaved, and how the correctness of these messages is proved. In this extended work, Web services are no longer treated as isolated components but as integral components of composition scenarios. Analyzing the synchronization of Web services at the composition level offers some direct benefits. First of all, it would be possible to dissociate the behaviors of Web services at the composite level from the behaviors of these Web services at the component level. Second, it would be possible to track the interactions that occur between the Web services from the component to the composite levels and *vice-versa*. Finally, it would be possible to work out the necessary synchronization mechanisms per type of composition orchestration whether centralized or P2P.

Section 2 discusses the commonalities and differences between the component and composite levels and provides a running scenario. Section 3 reports on the synchronization work that was done at the component level. Section 4 discusses synchronization at the composite level with focus on the P2P schema. Prior to concluding in Section 6, some experimental details are given in Section 5.

2 Background

2.1 Component *vs.* composition levels

In a composition scenario, we classify interactions that involve composite and component Web services into vertical (from composite Web service to component Web service) and horizontal (from component Web service to another component Web service). By establishing an interaction session, the initiator of a message aims at making the recipient of this message behave and take actions according to the content of this message. In the following, we identify the acceptable actions that a message initiator can execute over a potential recipient during vertical and horizontal interactions. The objective of identifying these actions is to facilitate the definition of the relevant synchronization messages that would be suitable per type of interaction.

In vertical interactions, a centralized orchestration of Web services composition is implemented. Here, a composite Web service through the centralized module has the authority to carry out the following actions over a component Web service:

"*Invite*" action makes the composite Web service request the participation of the component Web service in its composition scenario[1];

"*Ping*" action makes the composite Web service check the liveness of the component Web service that accepted its invitation of participation; there is no guarantee that the component Web service is still part of a composition scenario at time of invocation;

"*Trigger*" action makes the composite Web service initiate the execution of the component Web service;

"*Audit*" action makes the composite Web service monitor the performance of the component Web service for assessment purposes; service level agreements motivate the audit exercise;

And, "*retract&invite*" action makes the composite Web service withdraw the component Web service from its composition due to poor performance for example. This yields into searching for another replacement Web service that will be added to this composition.

In horizontal interactions, a P2P orchestration of Web services is implemented. Here, a component Web service has the authority to carry out the following actions over a peer:

"*Invite*" action makes the component Web service request the participation of the peer in the current composition scenario;

"*Ping*" action makes the component Web service check the liveness of the peer that accepted its invitation of participation; there is no guarantee that the peer is still part of a composition scenario at time of invocation;

And, "*trigger*" action makes the component Web service initiate the execution of the peer.

Compared to the vertical interactions in the centralized orchestration, "audit" and "retract" actions are excluded from the horizontal interactions in the P2P orchestration. Essentially, this is due to the challenges that are posed when tracking the performance of Web services and replacing them if needed. Not all providers would like to have their Web services audited by the Web services of other providers for reasons that could be related to security, privacy, competitiveness, etc. In a centralized orchestration, providers do not mutually interact with each other and might not even know that they are parts of the same composition scenario. The absence of "audit" and "retract" actions in a P2P orchestration sheds the light on the necessity of developing appropriate mechanisms that should take into account concerns like privacy and competitiveness. However, these mechanisms do not fall into the scope of this paper.

2.2 Running scenario

Our running scenario concerns a university student who is in the process of organizing a cookout party to celebrate his recent graduation. We identify hereafter the Web services along with their activities that will implement this party's logistics.

CateringWS: searches for and contacts catering companies according to some criteria like allocated budget, num-

[1] Component Web services invitation is discussed in [6].

ber of expected guests, type of cuisine, etc.

GuestWS: sends invitees invitations, keeps track of confirmed invitations, reminds late invitees for confirmation, etc.

PlaceBookingWS: looks for a place to host the cookout party, books the place, completes the necessary paperwork like payment, etc.

WeatherWS: checks weather forecast for the day of the cookout party. In case of bad weather, the party takes place at the student's place.

In our initial synchronization project [9], *state charts* were selected to specify component Web services independently of any composition scenario (Fig. 2 (a)). For the sake of compliance, we continue doing so when modeling the specification of composition scenarios. However states correspond this time to Web services taking part in these scenarios (Fig. 1).

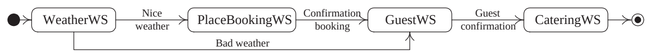


Figure 1. Specification of the cookout party

3 Specification of Web services

3.1 Control and operational behaviors

The *control behavior* shows the business logic that underpins the functioning of a Web service with respect to its functionality. A business logic is domain-application dependent (e.g., healthcare) and changes from one case study to another according to various requirements such as user (e.g., minimum age to submit an application), security (e.g., type of encryption algorithm), etc.

The *operational behavior* guides the execution progress of the business logic of a Web service. To this end, this behavior relies on a specific number of states, which are `activated`, `not-activated`, `done`, `aborted`, `suspended`, and `compensated`. These states are reported in the field of transactional Web services [11] and common to a certain extent to all Web services (and to any software application) regardless of their functionalities, origins, and locations.

As mentioned in Section 1, the control and operational behaviors of a Web service are modeled using state charts. This exercise of modeling is hereafter interleaved with some formal definitions and illustrative examples.

Definition 1 (*Web Service Behavior*). The behavior of a Web service is a 5-tuple $\mathcal{B} = \langle \mathcal{S}, \mathcal{L}, \mathcal{T}, s^0, \mathcal{F} \rangle$ where: $\mathcal{S}$ is a finite set of state names; $s^0 \in \mathcal{S}$ is the initial state; $\mathcal{F} \subseteq \mathcal{S}$ is a set of final states; $\mathcal{L}$ is a set of labels; and $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{L} \times \mathcal{S}$ is the transition relation. Each transition $t = (s^{src}, l, s^{tgt})$ consists of a source state $s^{src} \in \mathcal{S}$, a target state $s^{tgt} \in \mathcal{S}$, and a transition label $l \in \mathcal{L}$. From now on, we qualify transitions in the behavior of a Web service as *intra-behavior*. □

The control and operational behaviors of a Web service are defined as instances of the behavior of this Web service (Definition 1). These two behaviors are denoted by $\mathcal{B}_{co} = \langle \mathcal{S}_{co}, \mathcal{L}_{co}, \mathcal{T}_{co}, s^0_{co}, \mathcal{F}_{co} \rangle$ and $\mathcal{B}_{op} = \langle \mathcal{S}_{op}, \mathcal{L}_{op}, \mathcal{T}_{op}, s^0_{op}, \mathcal{F}_{op} \rangle$, respectively.

Example 1: *Fig. 2 (a) is a state chart of the control behavior of WeatherWS. Several states like* `city-located` *(initial state),* `report-delivered` *(final state), and* `search-canceled`*, and several transitions like (*`city-located`*,* <u>`unavailable`</u>*,* `search-canceled`*) are included in this state chart. In this transition example,* `city-located` *and* `search-canceled` *are the source and target states, respectively, and* <u>`unavailable`</u> *is the transition's label.*

Example 2: *Fig. 2 (b) is another state chart that illustrates this time the operational behavior of WeatherWS. Similar to the control behavior, several states like* `not-activated` *and* `suspended`*, and transitions like (*`compensated`*,* <u>`rolling-back`</u>*,* `not-activated`*) and (*`activated`*,* <u>`failure`</u>*,* `aborted`*) are identified in this state chart.*

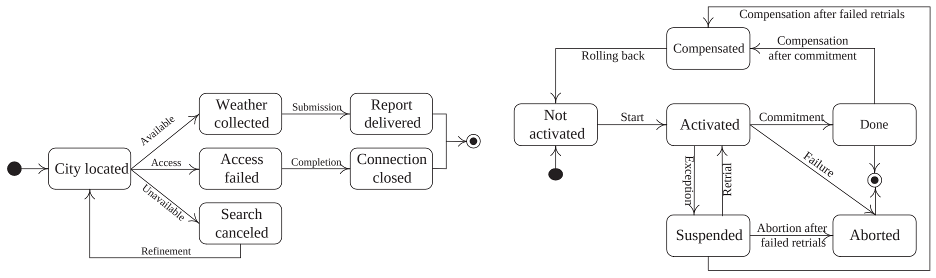
In Fig. 2, the control and operational behaviors of a Web service include different finite sequences that connect states and transitions together. We refer to these sequences as *paths* and define them as follows:

Definition 2 (*Path in Web Service Behavior*). A path $p^{i \to j}$ in the behavior $\mathcal{B}$ of a Web service is a finite sequence of states and transitions starting from state s^i and ending at state s^j and is denoted as follows: $p^{i \to j} = s^i \xrightarrow{l^i} s^{i+1} \xrightarrow{l^{i+1}} s^{i+2} \ldots s^{j-1} \xrightarrow{l^{j-1}} s^j$ such that $\forall k \in \{i, j-1\} : (s^k, l^k, s^{k+1}) \in \mathcal{T}$ (exponents in state names are here given for notational purposes only). □

Example 3: *Let l^1 (resp. l^2) =* `start` *(resp.* `commitment`*) in Fig. 2 (b).* `not-activated` $\xrightarrow{l^1}$ `activated` $\xrightarrow{l^2}$ `done` *is a path in the operational behavior of WeatherWS.*

3.2 Both behaviors in interaction

We pointed out that the operational behavior guides the performance of the control behavior of a Web service. This guidance requires bringing both behaviors together. For instance, `done` state that a Web service takes on in the operational behavior will in return make this



(a) Control behavior **(b) Operational behavior**

Figure 2. *WeatherWS***'s control and operational behaviors**

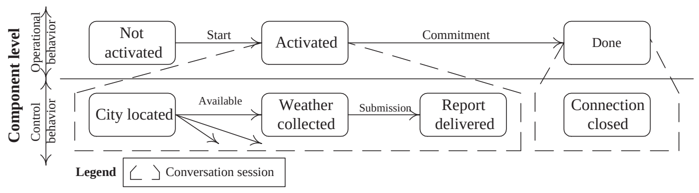


Figure 3. Example of operational and control behaviors mapping in *WeatherWS*

Web service take on other appropriate counterpart states like `weather-collected` and `report-delivered` in the control behavior.

The process of connecting operational and control behaviors together results in establishing *conversation sessions* between the respective states of these two behaviors (Fig. 3). To complete this connection process, a mapping function is defined as follows:

Definition 3 (*Mapping Function*). Let $\mathcal{P}_{co}$ be the set of all paths in the control behavior of a Web service starting from any state in this behavior. Connecting the operational behavior to the control behavior and *vice-versa* occurs using the following mapping function: $Map : \mathcal{S}_{op} \rightarrow 2^{\mathcal{P}_{co}}$, where $2^{\mathcal{P}_{co}}$ is the power set of $\mathcal{P}_{co}$. $\square$

What the he mapping function Map does is to associate each state in the operational behavior with a set (possibly empty) of possible paths in the control behavior.

Example 4: *Fig. 3 is an example of the use of the mapping function in WeatherWS where* `activated` *state in the operational behavior is associated with multiple paths in the control behavior. One of these paths is:* `city-located` $\xrightarrow{l^1}$ `weather-collected` $\xrightarrow{l^2}$ `report-delivered` *where* $l^1 =$ `available` *and* $l^2 =$ `submission`. *A second path for* `activated` *state is given in Fig. 4 (b) as well.*

On top of the mapping function *Map*, interactions between control and operational behaviors require a specification operation that indicates which state in the operational behavior is associated with which set of possible paths in the control behavior along with the "new" transitions that will implement these interactions. The next state to take on in the operational behavior is determined by the executed path in the control behavior and whether this execution was a success or failure. In other words, the specification operation lets the control behavior indicate to the operational behavior what needs to be done next. We define the specification operation as follows:

Definition 4 (*Specification Operation*). Let $\mathcal{L}_S$ be the set of labels associated with the "new" transitions between operational and control behaviors. The specification operation uses the following two functions:
$Spec : \mathcal{S}_{op} \rightarrow 2^{\mathcal{L}_S \times \mathcal{P}_{co} \times \mathcal{L}_S}$ and $Next : \mathcal{S}_{op} \times \mathcal{P}_{co} \rightarrow \mathcal{L}_{op} \times \mathcal{S}_{op}$. $\square$

The specification function $Spec$ associates each state s_{op} in the operational behavior with a (possibly empty) set of triples. A triple contains (i) the label of the transition from s_{op} to the first state in the control behavior of a mapped path, (ii) the mapped path itself $p^{i \rightarrow j}$, and (iii) the label of the transition from the last state in the control behavior of the mapped path back to s_{op} in the operational behavior. We qualify the "new" transitions that connect states in independent state charts as *inter-behavior* (note that intra-behavior transition was used in Definition 1). The partial function $Next$ associates both a given state in the operational behavior and the mapped path in the control behavior with both the next state to take on in the operational behavior and the associated transition label.

Example 5: *Fig. 4 shows the synchronization of WeatherWS's operational and control behaviors where two types of transitions exist: intra-behavior from $\mathcal{T}_{co} \cup \mathcal{T}_{op}$ (plain lines) and inter-behavior (dashed lines, $Labels_{1,2,3}$). Fig. 4 contains $Spec$(`activated`)=$\{$($label_1$, $path_1$, $label_2$),($label_1$, $path_2$, $label_3$)$\}$, $Next$(`activated`,$path_1$)=(`commitment`,`done`), where $path_1$ = `city-located` $\xrightarrow{l^1}$ `weather-collected` $\xrightarrow{l^2}$ `report-delivered` and $Next$(`activated`,$path_2$)=(`failure`,`aborted`) where $path_2$= `city-located` $\xrightarrow{l^3}$ `access-failed` $\xrightarrow{l^4}$ `connection-closed`.*

In Fig. 4, the initiation of *WeatherWS* is shown in the operational behavior with `activated` state. *WeatherWS* takes on this state following receipt of a user's request. Because of (`activated`, $\underline{\texttt{label}_1}$, `city-located`) inter-behavior transitions, the execution of *WeatherWS* begins by using a dedicated database to search for the requested city. This makes *WeatherWS* take on `city-located` state in the control behavior. Afterwards, two cases are identified.

Case a. Everything goes fine and a 5-day weather-forecast report is delivered back to the user. Because of (`report-delivered`, $\underline{\texttt{label}_2}$, `activated`) inter-behavior transition, this makes *WeatherWS* complete its operation with success by transiting from `activated` to `done` states in the operational behavior, i.e., (`activated`, $\underline{\texttt{commitment}}$, `done`) intra-behavior transition.

Case b. The access to the database fails (not like in case a) as the control behavior of *WeatherWS* indicates with `access-failed` and `connection-closed` states. Because of (`connection-closed`, $\underline{\texttt{label}_3}$, `activated`) inter-behavior transition, this makes *WeatherWS* terminate its operation with failure by transiting from `activated` to `aborted` states in the operational behavior, i.e., (`activated`, $\underline{\texttt{failure}}$, `aborted`) intra-behavior transition.

To wrap-up this section, the formal definitions of inter-behavior and conversation session are provided. Needless to propose a formal definition for intra-behavior transition, which is a regular transition in a state chart (Definition 1).

Definition 5 (*Inter-Behavior Transition*). The set of all inter-behavior transitions that connect the operational and control behaviors of a Web service is denoted by $\mathcal{IT}$ where $\mathcal{IT} = \mathcal{IT}_{op\rightarrow co} \cup \mathcal{IT}_{co\rightarrow op}$ such that: $\mathcal{IT}_{op\rightarrow co} \subseteq \mathcal{S}_{IT(op)} \times \mathcal{L}_{op\rightarrow co} \times \mathcal{S}_{IT(co)}$ is the inter-behavior transition relation starting from the operational behavior and ending at the control behavior; $\mathcal{IT}_{co\rightarrow op} \subseteq \mathcal{S}_{IT(co)} \times \mathcal{L}_{co\rightarrow op} \times \mathcal{S}_{IT(op)}$ is the inter-behavior transition relation starting from the control behavior and ending at the operational behavior; $\mathcal{S}_{IT(op)} \subseteq \mathcal{S}_{op}$ is a finite set of state names in the operational behavior that take part in inter-behavior transitions; $\mathcal{S}_{IT(co)} \subseteq \mathcal{S}_{co}$ is a finite set of state names in the control behavior that take part in inter-behavior transitions; and $\mathcal{L}_{op\rightarrow co}$ is a set of inter-transitions' labels from the operational to the control behaviors, and $\mathcal{L}_{co\rightarrow op}$ is a set of inter-transitions' labels from the control to the operational behaviors ($\mathcal{L}_{co\rightarrow op} \cup \mathcal{L}_{op\rightarrow co} = \mathcal{L}_S$ (Definition 4)). $\Box$

Before we define Web service conversation session, we introduce another function known as Lab. This function returns the label of an inter-behavior transition: $Lab : \mathcal{IT} \rightarrow \mathcal{L}_{\mathcal{S}}$.

Definition 6 (*Web Service Conversation Session*). A conversation session between the operational and control behaviors of a Web service is a 4-tuple $\langle s_{op}, it_{op\rightarrow co}, p_{co}, it_{co\rightarrow op}\rangle$ such that: $s_{op} \in S_{op}$, $it_{op\rightarrow co} \in \mathcal{IT}_{op\rightarrow co}$, $it_{co\rightarrow op} \in \mathcal{IT}_{co\rightarrow op}$, $p_{co} \in \mathcal{P}_{co}$; and $(Lab(it_{op\rightarrow co}), p_{co}, Lab(it_{co\rightarrow op})) \in Spec(s_{op})$. $\Box$

4 Synchronization woven into composition

Synchronization is a mechanism by which independent entities coordinate their next actions by agreeing on how, where, and when to carry out these actions. In the rest of this paper, entities correspond to Web services that could be either component or composite. We look into synchronization from two perspectives: *Intra*, which means how the operational and control behaviors in a component/composite Web service are coordinated (Section 4.1), and *Inter*, which means how the operational and control behaviors in separate component Web services are coordinated within the context of the same composite Web service (Section 4.2). Because composition could have either a centralized or a P2P orchestration schema, inter Web-services synchronization is examined from these two types.

4.1 Intra Web-services synchronization

Case of component Web services. The synchronization of component Web services was the main object of our research project in [9], so further details are provided in this reference. Table 1 contains some synchronization messages we developed in order to allow the operational and control behaviors interact with each other.

Case of composite Web services. The synchronization of composite Web services is differently handled from the synchronization of component Web services. This is due to the characteristics of composite Web services that need now to be highlighted through their operational and

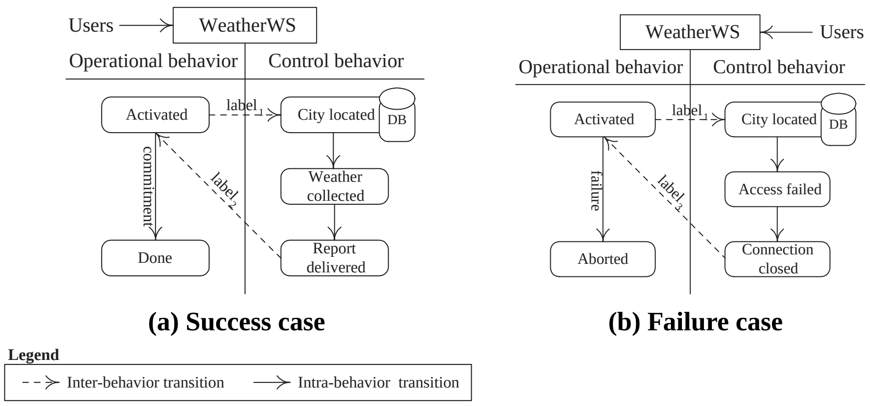


Figure 4. Synchronization of *WeatherWS*'s control and operational behaviors

Table 1. Messages during intra (component) Web-service synchronization

#	Message name	Description
1.	*sync*	Originates from an operational state and targets a control state. The purpose is to trigger the execution of the control states (including the targeted control state) in a conversation session. *sync* is a blocking message, which makes the operational state wait for a notification back from the last control state to execute in this conversation session.
2.	*success*	Originates from a control state and targets the operational state that submitted *sync*. The purpose is to inform this operational state of the successful execution of the control states in a conversation session and to return the execution thread back to this operational state as well. *success* is coupled with *sync*.

control behaviors. These characteristics are as follows. Firstly, the control behavior represents the business logic of a composition scenario and no longer the business logic of a certain component Web service. Secondly, the current definition of the operational behavior (Definition 1) does not tell much about the execution outcome of a composition scenario and if this execution either succeeded or failed. This current definition through states like `activated` and `suspended` is geared towards the needs of the component level, only (Fig. 2 (b)).

Definition 7 (*Composite Web Service Control Behavior*). The control behavior of a composite Web service is a 5-tuple $\mathcal{B}_{co}^{cws} = \langle \mathcal{WS}_{co}, \mathcal{L}_{co}, \mathcal{T}_{co}, \mathcal{WS}_{co}^{0}, \mathcal{F}_{co} \rangle$ where $\mathcal{WS}_{co}$ is a finite set of states that correspond to Web services' names; $\mathcal{WS}_{co}^{0} \subset \mathcal{WS}_{co}$ is the set of initial states that correspond to initial Web services; $\mathcal{F}_{co} \subseteq \mathcal{WS}_{co}$ is the set of final states that correspond to final Web services; $\mathcal{L}_{co}$ is a set of labels; and $\mathcal{T}_{co} \subseteq \mathcal{WS}_{co} \times \mathcal{L}_{co} \times \mathcal{WS}_{co}$ is the transition relation. Each transition $t = (ws^{src}, l, ws^{tgt})$ consists of a source Web service $ws^{src} \in \mathcal{WS}_{co}$, a target Web service $ws^{tgt} \in \mathcal{WS}_{co}$, and a transition label $l \in \mathcal{L}_{co}$. $\square$

Example 6: *Fig. 1 is a state chart of the control behavior of the CookoutParty composite Web service. Several states like* `WeatherWS` *(initial state) and* `CateringWS` *(final state) and several transitions like* (`WeatherWS`, <u>`NiceWeather`</u>, `PlaceBookingWS`) *are included. In this transition example,* `WeatherWS` *and* `PlaceBookingWS` *are the source and target states, respectively, and* <u>`NiceWeather`</u> *is the transition's label.*

Definition 8 (*Composite Web Service Operational Behavior*). The operational behavior of a composite Web service is defined as an instance of the behavior of a Web service (Definition 1) and is denoted by $\mathcal{B}_{op}^{cws} = \langle \mathcal{S}_{op}, \mathcal{L}_{op}, \mathcal{T}_{op}, s_{op}^{0}, \mathcal{F}_{op} \rangle$. $\square$

The purpose of the operational behavior of a composite Web service is (i) to initiate the execution of its specification, which is in fact the control behavior of this composite Web service and (ii) to report on the success or failure of the execution of this specification. As a result, the operational behavior of a composite Web service is a subset of the operational behavior of a component Web service.

Example 7: *Fig. 5 is another state chart that illustrates this time the operational behavior of the CookoutParty composite Web service. In this state chart, the number of states is limited to four, namely* `not-activated`*,* `activated`*,* `done`*, and* `aborted`*, and the number of transitions is limited to three, namely* `start`*,* `commitment`*, and* `failure`*.*

Compared to the six states in the operational behavior of a component Web service (Fig. 2 (b)), the four states in the

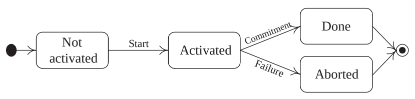


Figure 5. Operational behavior of a composite Web service

operational behavior of a composite Web service (Fig. 5) puts some restrictions on the authorized synchronization messages (like those suggested in Table 1) that can be considered between this operational behavior and its counterpart control behavior. These restrictions are hereafter listed:

1. There is one conversation session between the operational and control behaviors. This session includes the `activated` state in the operational behavior and all the states (i.e., $\mathcal{WS}_{co}$) in the control behavior.

2. Interaction message of type *sync* to come out of the `activated` state in the operational behavior has one recipient, which is the initial state(s) (i.e., Web service) in the control behavior (i.e., $\mathcal{WS}^0_{co}$).

3. Any state (i.e., component Web service) in the control behavior can only submit an interaction message of type *fail* back to the `activated` state in the operational behavior. This restriction is waived for the final state(s) in the control behavior (i.e., $\mathcal{F}_{co}$) that can submit on top of *fail* message another message of type *success* back to the `activated` state in the operational behavior.

4. Interaction message of type *syncreq* is not allowed from the control to the operational behaviors.

4.2 Inter Web-services synchronization

Centralized orchestration is well "embraced" in Web services composition projects. But, a few projects look into the changes that need to be made in Web services standards/specifications like BPEL to smooth the design and development of P2P orchestration. Gowri Nanda et al. note that because performance and throughput are major concerns in enterprise applications, removing the inefficiencies that a centralized control introduces, is required [4]. A BPEL program could be partitioned into independent subprograms that interact with each other without any centralized control. Gowri Nanda et al. propose a technique to partition a composite Web service written as a single BPEL program into an equivalent set of decentralized processes. This technique minimizes communication costs and maximizes the throughput of multiple instances of the input program.

In this paper, we look at inter Web-services synchronization from two perspectives: centralized and **P2P** (focus of this paper). This synchronization aims at initiating the development of composition scenarios and overseeing the execution progress of this development at run-time. As a result, this raises the necessity of enhancing Web services with additional mechanisms based on the needs and requirements of these composition scenarios. For instance, a Web service has now to decide if it would or not take part in a composition scenario subject to carrying out some sort of self-assessment [6]. That was not the case in the intra Web-services synchronization (Section 4.1) where the focus was on how to specify the execution of "isolated" Web services.

We identify the additional mechanisms that should embody Web services along four cases, which we denote by *invitation*, *execution*, *verification*, and *replacement*. These four cases abstract the different types of actions that component and composite Web services carry out during vertical and horizontal interactions. For example, a Web service should submit its performance details to a composite Web service as part of the verification exercise that this composite Web service carries out. In addition, a Web service should not leave its ongoing operations pending in case a composite Web service decides to substitute it as part of the replacement exercise. These additional mechanisms need to be woven into the business logic that underpins the functionality of a Web service. In [7], we elaborate on how this weaving should place in compliance with some design principles like separation of concern and aspect-oriented programming. To keep the paper self-contained on synchronization, enriching Web services with additional mechanisms is excluded.

Case of P2P orchestration. The synchronization of inter Web-services in a P2P orchestration reinforces the existence of the component level, only. Each component Web service that takes part in a composition scenario is associated with an operational and a control behaviors (Fig. 6). The previously proposed definitions for these two behaviors continue to be used (Definition 1). However, new definitions are deemed appropriate for first, the inter-behavior transitions between component Web services and second, the conversation sessions that result out of setting-up these inter-behavior transitions. These new definitions have to be inline with the authorized actions to carry out in a P2P orchestration. These actions are "invite", "trigger", and "ping".

In Fig. 6, the double-arrowed lines (plain and dashed) illustrate where the synchronization of inter Web-services should take place in a P2P orchestration. Numbers associated with these lines represent message chronology. In the P2P orchestration we hereafter adopt, sequential execution of the component Web services is assumed even though concurrent execution could be handled without any substan-

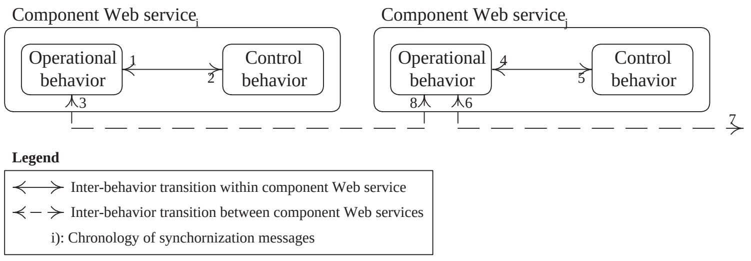

Component Web service$_i$
Operational behavior
Control behavior
Component Web service$_j$
Operational behavior
Control behavior
1
2
3
4
5
6
7
8
Legend
Inter-behavior transition within component Web service
Inter-behavior transition between component Web services
i): Chronology of synchornization messages

Figure 6. Synchronization of behaviors at the composition level – P2P orchestration

tial changes in the new or existing definitions. Plain lines in Fig. 6 represent inter-behavior transitions within the same component Web service (Section 4.1). What is now needed, which is the focus of this part of the paper, is to define the inter-behavior transitions between component Web services. These inter-behavior transitions are represented with dashed lines (3, 6, 7, 8) in Fig. 6.

In a P2P orchestration, the absence of a centralized coordination that would "spread the word" to other component Web services about the execution outcomes of their peers requires some changes in the way these component Web services should behave. For instance, component Web services cannot announce their immediate successful execution until they receive positive feedbacks from their peers in a reverse order. In case of negative feedbacks, these component Web services have to cancel or compensate their execution outcomes and notify their predecessors about the cancelation or compensation actions they have taken, as well. Announcement delay and backward notification have to be reflected on the operational levels of the different component Web services. Like in a centralized orchestration we split `done` state into two states (Fig. 7): `partial done` and `final done`.

- `Partial done` in a component Web service allows to pass on the execution thread to the next peer(s) (Fig. 6, dashed lines 3 and 6).

- `Final done` in a component Web service permits to confirm its successful execution (final completion) following receipt of a positive notification from a successor peer (Fig. 6, dashed lines 7 and 8).

Definition 9 (*Completion Status of Component Web Service in Peer-to-Peer Orchestration*). Let assume a composition scenario of n component Web services. The completion status of a component Web service WS$_i$ in term of either success or failure is dependent on the notification message that WS$_i$ receives from its direct successor component Web service WS$_{i+1}$.

$$Status(WS_i) = \begin{cases} Notify(WS_{i+1}) & i = 1, \cdots, n-1 \\ success \mid failure & i = n \end{cases}$$

It is worth to mention that *success* and *failure* are conversational messages between component Web services. These messages are specified in Table 2.

Fig. 7 illustrates the different conversation sessions that need to be set-up in a P2P orchestration. The focus is on conversation session #2; conversation session #1 is already discussed in Section 4.1. The identification of the inter-behavior transitions that should be included in conversation scenario #2 takes advantage of the set of acceptable actions (e.g., "invite" and "ping") that can be carried out between component Web services. These actions are now woven into the synchronization messages to occur between the respective operational behaviors of the component Web services. The following comments are made on the new operational behavior of a component Web service: (i) `partial done` and `final done` states are added and connected, (ii) `partial done` and `aborted` states are connected, and (iii) `partial done` and `compensated` states are connected as well.

Table 2 summarizes some messages that can be exchanged in a P2P orchestration during inter Web-services synchronization. This table is built upon the messages of Table 1. The description of each message type shows (i) the direction of the bidirectional flow between the operational behaviors of the component Web services, and (ii) the case that corresponds to the actions to perform during horizontal interactions. Interesting to discuss messages #9 and #10, i.e., *confirm* and *cancel*, respectively in Table 2. Both messages are used by component Web services to notify other component Web services that they could either confirm or cancel their execution.

Definition 10 (*Inter-Behavior Transition in Peer-to-Peer Orchestration*). The set of all inter-behavior transitions that connect the operational behaviors of component Web services together in a P2P-orchestration mode (conversation session #2 in Fig. 7) is denoted by $\mathcal{IT}^{(ws_i,ws_j)}$ where $\mathcal{IT}^{(ws_i,ws_j)} = \mathcal{IT}^{(ws_i,ws_j)}_{op\rightarrow op} \cup \mathcal{IT}^{(ws_j,ws_i)}_{op\rightarrow op}$ such that:

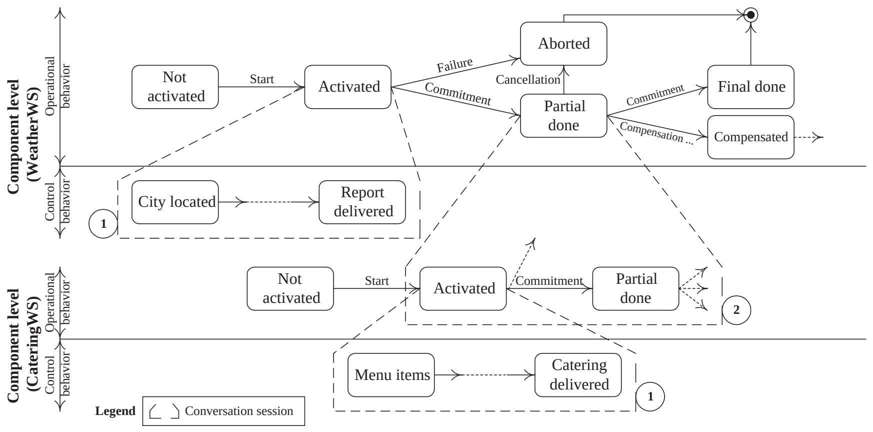


Figure 7. Operational and control behaviors mapping in *CookoutParty* **- P2P orchestration**

$\mathcal{IT}^{(ws_i,ws_j)}_{op\rightarrow op} \subseteq \mathcal{S}_{IT(op)} \times \mathcal{L}^{(ws_i,ws_j)}_{op\rightarrow op} \times \mathcal{S}_{IT(op)}$ is the inter-behavior transition relation starting from the operational behavior of a component Web service (WS_i) and ending at the operational behavior of another component Web service (WS_j). Same definition applies to $\mathcal{IT}^{(ws_j,ws_i)}_{op\rightarrow op} \subseteq \mathcal{S}_{IT(op)} \times \mathcal{L}^{(ws_j,ws_i)}_{op\rightarrow op} \times \mathcal{S}_{IT(op)}$; $\mathcal{S}_{IT(op)} \subseteq \mathcal{S}_{op}$ is a finite set of state names in the operational behavior of a component Web service that take part in inter-behavior transitions; and $\mathcal{L}^{(ws_i,ws_j)}_{op\rightarrow op}$ is a set of inter-transitions' labels from the operational behavior of a component Web service (WS_i) to the operational behavior of another component Web service (WS_j), and $\mathcal{L}^{(ws_j,ws_i)}_{op\rightarrow op}$ is the opposite ($\mathcal{L}^{(ws_i,ws_j)}_{op\rightarrow op} \cup \mathcal{L}^{(ws_j,ws_i)}_{op\rightarrow op} = \mathcal{L}_S$). $\Box$

5 Implementation

To test the viability of the proposed approach, a prototype system was implemented in Java and integrated under Eclipse 3.3 by extending the Web service development platform we developed previously [9]. The prototype consists of four modules:
`ControlBehaviorModeler`,
`OperationalBehaviorModeler`,
`ConversationModeler` and
`SimulationController`.
The `ControlBehaviorModeler` and the `OperationalBehaviorModeler` assist engineers specify the control and operational behaviors of a component or a composite Web service, respectively. In particular, we developed a visual interface for editing Web services' behaviors using state charts. The `ConversationModeler` takes the behavior specifications of a Web service as an input to produce conversation specifications (i.e., inter-transitions and message sequences). It implements functions to support conversations between operational and control behaviors. Specifically, it provides methods for managing conversation instances and triggering transitions. When dealing with a peer to peer synchronization, the `ConversationModeler` manages first, the inter-behavior transitions between component Web services and second, the conversation sessions that result out of setting-up these inter-behavior transitions. Finally, the `SimulationController` tracks and analyzes (if necessary) the execution of a composite or a component Web service according to its conversation definition (e.g., whether the messages are received and sent in an appropriate order).

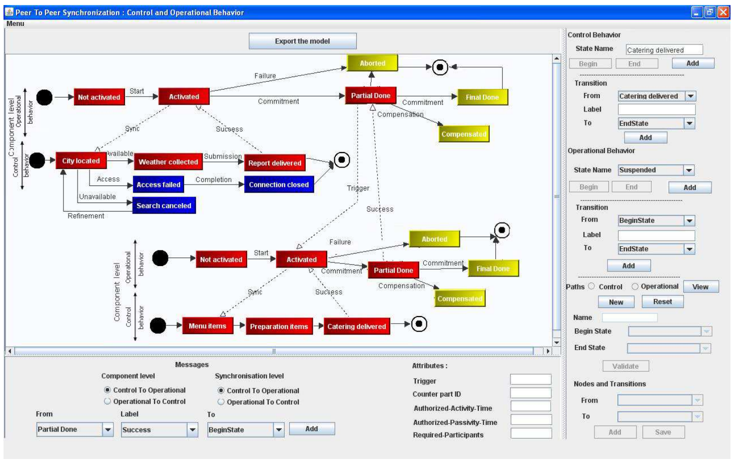
Fig. 8 shows the execution of a component Web service in the case of a peer to peer orchestration. Upon the reception of a user's request, the operational level of the first component Web service (*WeatherWS* in this case) moves from not-activated state to activated state (red color is used to show the execution path). This latter state, submits a `Sync` message to `Citylocated` state in the control behavior to trigger its execution. In a success case, `Reportdelivered` state returns a `Success` message back to the activated state in the operational behavior. Based on this information, the operational behavior moves from activated state to partial done state. This latter state sends a trigger message to invoke the next component Web service (`CateringWS`).

6 Conclusion

We presented in this paper a framework for establishing synchronization between Web services engaged in composition scenarios. Synchronization assists independent parties coordinate their actions and thus, avoid conflicts. This framework extends the research work we carried out on iso-

Table 2. Messages during inter Web-services synchronization - P2P orchestration

#	Message name	Description			
		From	**To**	**In reply to**	**Case**
1.	*invite*	Component WS$_i$ ($\mathcal{B}_{op}$)	Component WS$_j$ ($\mathcal{B}_{op}$)	null	Invitation
2.	*trigger*	Component WS$_i$ ($\mathcal{B}_{op}$)	Component WS$_j$ ($\mathcal{B}_{op}$)	null	Execution
3.	*ping*	Component WS$_i$ ($\mathcal{B}_{op}$)	Component WS$_j$ ($\mathcal{B}_{op}$)	null	Verification
4.	...	...	...	...	...
9.	*confirm*	Component WS$_i$ ($\mathcal{B}_{op}^{ws}$)	1[Component WS$_j$ ($\mathcal{B}_{op}$)](i-1)	success	Execution
10.	*cancel*	Component WS$_i$ ($\mathcal{B}_{op}^{ws}$)	1[Component WS$_j$ ($\mathcal{B}_{op}$)](i-1)	failure	Execution



Messages
Component level: [✓ Control To Operational] Operational To Control
Synchronisation level: [✓ Control To Operational] Operational To Control
From: Partial Done — Label: Success — To: BeginState — Add

Attributes :
Trigger
Counter part ID
Authorized-Activity-Time
Authorized-Passivity-Time
Required-Participants

Control Behavior
State Name: Catering delivered
Begin End Add
Transition
From: Catering delivered
Label:
To: EndState
Add

Operational Behavior
State Name: Suspended
Begin End Add
Transition
From: BeginState
Label:
To: EndState
Add

Paths ○ Control ○ Operational View
New Reset
Name
Begin State
End State
Validate
Nodes and Transitions
From
To
Add Save

Figure 8. An execution of a component Web service in a P2P orchestration

lated Web services (not engaged in any composition) and leverages two types of behaviors related to specifying such Web services. The control behavior defines the business logic that underpins the functioning of a Web service, and the operational behavior regulates the execution progress of this control behavior by stating the actions to carry out and the constraints to put on this progress. Synchronizing Web services through their respective behaviors has revealed that the orchestration schemas, whether centralized or P2P, affect the mechanisms to develop in response to the needs and requirements of the composition scenario that is under consideration. The use of some of these mechanisms per orchestration schema was demonstrated through a prototype. In term of future work, we plan to study the value-add of model checking to the early-detection of design inconsistencies and errors.

References

[1] Benatallah. B., Q. Z. Sheng, Ngu A. H. H., and M. Dumas. Declarative Composition and Peer-to-Peer Provisioning of Dynamic Web Services. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'2002)*, San Jose, CA, US, 2002.

[2] F. Casati and M. C. Shan. Dynamic and Adaptive Composition of E-Services. *Information Systems*, 26(3), 2001.

[3] F. Daniel and B. Pernici. Insights into Web Service Orchestration and Choreography. *International Journal of E-Business Research, The Idea Group Inc.*, 1(2), 2005.

[4] M. Gowri Nanda, S. Chandra, and V. Sarkar. Decentralizing Execution of Composite Web Services. In *Proceedings of 19th Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA'2004)*, Vancouver, British Columbia, Canada, 2004.

[5] M. Kovas, J. Bentahar, Z. Maamar, and H. Yahyaoui. Formal Verification of Conversations in Composite Web Services using NuSMV. In *Proceedings of the 8th International Conference on Software Methodologies, Tools and Techniques (SoMeT09), IOS Press*, Prague, Czech Republic, 2009.

[6] Z. Maamar, S. Kouadri Mostéfaoui, and H. Yahyaoui. Towards an Agent-based and Context-oriented Approach for Web Services Composition. *IEEE Transactions on Knowledge and Data Engineering*, 17(5), May 2005.

[7] Z. Maamar, M. Sheng, D. Benslimane, and H. Yahyaoui. Web Services Interactions: Analysis, Modeling, and Management. *International Journal on Software Engineering and Knowledge Engineering, World Scientific Publishing Co.*, 18(2), March 2008.

[8] Z. Maamar, M. Sheng, H. Yahyaoui, J. Bentahar, and K. Boukadi. A New Approach to Model Web Services' Behaviors based on Synchronization. In *Proceedings of the 23rd IEEE International Conference on Advanced Information Networking and Applications, Symposium on Frontiers of Information Systems and Network Applications*, Bradford, UK, 2009.

[9] Z. Maamar, M. Sheng, H. Yahyaoui, J. Bentahar, and K. Boukadi. Conversation-Oriented Engineering of Web Services. Technical report, College of Information Technology, Zayed University, September 2007.

[10] 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.

[11] W. Yang and S. Tang. A Solution for Web Services Transaction. In *Proceedings of The 2006 International Conference on Hybrid Information Technology (ICHIT'2006)*, Cheju Island, Korea, 2006.