2009 International Conference on Advanced Information Networking and Applications Workshops

A New Approach to Model Web Services' Behaviors based on Synchronization

Zakaria Maamar
Zayed University, Dubai, U.A.E
zakaria.maamar@zu.ac.ae

Quan Z. Sheng
The University of Adelaide, Adelaide, Australia
qsheng@cs.adelaide.edu.au

Hamdi Yahyaoui
KFUPM, Dhahran, Saudi Arabia
hamdi@kfupm.edu.sa

Jamal Bentahar
Concordia University, Canada
bentahar@encs.concordia.ca

Khouloud Boukadi
École des Mines, Saint-Etienne, France
boukadi@emse.fr

Abstract

This paper introduces a novel approach for modelling and specifying behaviors of Web services. This approach excludes Web services from any composition scenario and sheds the light on two types of behaviors: control and operational. The control behavior illustrates 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. To synchronize both behaviors at run-time, conversational messages are developed and permit conveying various details between these two behaviors. A prototype showing the use of these conversational messages is presented in this paper as well.

1. Introduction

Web services are gaining momentum in academia and industry as they aim at fulfilling the promise of developing loosely-coupled business processes [10], [11]. This momentum is witnessed from the widespread adoption of Web services in multiple R&D projects and application domains. Moreover, this is witnessed from the variety of standards for Web services. These projects and standards are to a certain extent all concerned with the obstacles that hinder automatic composition of Web services. Simply put, composition handles the situation of a user's request that cannot be satisfied by any single, available Web service, whereas a composite Web service obtained by combining available Web services may be used.

However, despite the recent advances in Web services, the verification of their design correctness is still lagging behind. This verification should occur first at the component level by excluding Web services ("isolated") from any composition scenarios, and then at the composition level by connecting the already-verified Web services ("certified") together. Service engineers should check and track a design correctness prior to letting Web services implement real business applications. To this end, engineers need to monitor execution status, identify behavior patterns, detect design errors such as deadlocks, reachability, and termination, and offer appropriate solutions. Restricting Web services to process just requests of users or peers without allowing these Web services to check their execution states will reduce, and probably undermine, their participation opportunities in complex business applications [3].

In a dynamic environment like the Internet, Web services should be given the opportunity to control and adjust their behaviors according to the status of this environment. We expose this status through messages that Web services exchange on different matters such as peers they interact with, operations they execute, events they handle, and exceptions they raise. Our proposal to "keep an eye" on a Web service consists of separating its behavior into two types: *control* and *operational*. With the control behavior, we illustrate the business logic that underpins the functioning of a Web service, i.e, how the functionality (e.g., `CarRental`) of a Web service is achieved. With the operational behavior, we regulate in a step-by-step way the progress of executing the control behavior (i.e., business logic) by stating the actions that need to be taken and the constraints that need to be put on this progress. It is noted that the operational behavior is public and application-independent, and thus common to almost all Web services. Contrarily, the control behavior is private and specific to a given Web service.

The existence of the control and operational behaviors complies with the "*separation of concerns*" design principle [6] that the object-oriented domain promotes through data and operation separation. This separation not only eases the design (in terms of development, maintenance, etc.) of Web services, but makes the verification of the correctness of this design possible by synchronizing the two behaviors. This is doable by checking if the operations implemented through the business logic (the control behavior) respect the properties stated in the operational behavior. For example, if the operational behavior allows Web services to be compensated, it will be possible to check if the compensation actions are correctly integrated into the Web service design. Another example is the verification of the possible terminations. Synchronization can reveal that the control behavior terminates at some states that are not allowed by the operational behavior, which could result in an inconsistent Web service.

Both behaviors are designed in a loosely-coupled way. In this paper, control and operational behaviors are modelled with finite state machines [5]. Other formalisms (e.g., Petri-Nets) could be used without any impact on how these behaviors are modelled, verified, synchronized, and deployed.

The requirement of maintaining a Web service in a consistent state all the time stresses the necessity of coordinating, i.e., synchronizing, the control and operational behaviors using a set of conversational messages. The use of conversations in Web services is widely acknowledged in the literature [1], [2], [4], [9]. However none of the efforts reported so far examined the value-add of conversations to model and verify the behaviors of "isolated" Web services before they get engaged in complex business applications. These applications have strict requirements in terms of reliability, efficiency, and availability, which raises the importance of guaranteeing the proper functioning of Web services and verifying some critical properties like deadlock, liveness, and safety. We devise conversational messages in a way that various details are conveyed between behaviors. For instance, a message from the operational to the control behaviors indicates the nature of event, e.g., temporal, that triggered the Web service. We associate the necessary conversational messages with performatives (like in FIPA-ACL), which we classify into two categories: *activation* performatives support the operational behavior initiate the execution of the control behavior of a Web service, and *outcome* performatives support the control behavior report the status of this execution to the operational behavior for further actions that will later, affect this control behavior.

The way we engineer Web services goes beyond the use of conversational messages as a simple interaction means. Indeed, compared to the existing initiatives on Web services conversations, this is the first work in which the operational and control behaviors are made accessible ("interactible") to each other through conversations. It is noted that some of these initiatives focus on either the operational behavior (e.g., [8]) or the control behavior (e.g., [2]) of Web services, while other initiatives do not even acknowledge the existence of these behaviors. Our contributions in this paper are manifold: (i) models that separate operational and control behaviors of Web services, (ii) mechanisms that support the synchronization of both behaviors using conversational messages, (iii) definition of performatives that implement these conversational messages, (iv) verification of the correctness of sequences of conversational messages, and last but not least (v) development of a framework that realizes all the proposed concepts.

The rest of the paper is organized as follows. Section 2 discusses the approach to engineering "isolated" Web services. Section 3 presents how control and operational behaviors of Web services are synchronized through appropriate conversational messages. Implementation is reported in Sections 4. Conclusions are drawn in Section 5.

2. The Engineering Approach

Our engineering approach consists of three steps. The *mapping* step identifies which states in the operational behavior match which states in the control behavior of a Web service. It is assumed that finite state machines of both behaviors already exist prior to carrying out this step. The *specification* step works out the structure and details of the conversational messages to exchange between both behaviors. These messages are the result of the mapping step. And the *synchronization* step deploys the conversational messages between both behaviors and follows up their exchange at run-time.

In the rest of this paper, a Web service denoted by *WeatherWS* is used for illustration purposes. Its functionality is to return a 5-day weather-forecast report on a certain city for a certain day. *WeatherWS* highlights the challenges that service engineers face when working out some of the following operations: how do both behaviors progress hand-in-hand, how does the operational behavior "remotely" reflect some changes on the control behavior and *vice-versa*, how do both behaviors implement the conversational messages they receive, how are the soundness and completeness properties of a Web service verified through both behaviors, and last but not least how to build well-formed sequences of conversational messages between both behaviors?

2.1. Control and Operational Behaviors

The *control behavior* shows the business logic that underpins the functioning of a Web service. A business logic is domain-application dependent (e.g., real estate domain) and changes from one scenario to another according to various requirements such as user (e.g., minimum age to submit a loan application) and legal (e.g., VAT rate). In contrast, the *operational behavior* guides the execution progress of the business logic of a Web service. The operational behavior relies on a number of states (`activated`, `not-activated`, `done`, `aborted`, `suspended`, and `compensated`)[1] that are reported in the transactional Web services literature [8], [12] and to a certain extent common to all Web services regardless of their functionalities, origins, and locations. We adopt finite state machines to model the control and operational behaviors of a Web service. This modelling is interleaved with formal definitions and illustrations.

1. Other states like `hold` and `tentative commit` could be added to the operational behavior [7], but the ones suggested in Fig.1 (*b*) are expressive enough to represent the functioning of any system.

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 $l \in \mathcal{L}$. We qualify these transitions as *intra-behavior* (the rationale of this qualification is given later). $\Box$

A Web service's control and operational behaviors are instances of the Web service's behavior. 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.

In Fig. 1, the control and operational behaviors rely on a set of finite sequences of states and transitions. These sequences are called *paths* and defined as follows:

Definition 2 (Path in Web Service Behavior): A path $p^{i \to j}$ in a Web service's behavior $\mathcal{B}$ is a finite sequence of states and transitions starting from state s^i and ending at state s^j and denoted as follows: $p^{i \to j} = s^i \overset{l^i}{\to} s^{i+1} \overset{l^{i+1}}{\to} s^{i+2} \ldots s^{j-1} \overset{l^{j-1}}{\to} s^j$ such that $\forall k \in \{i, j-1\} : (s^k, l^k, s^{k+1}) \in \mathcal{T}$ (here exponents in state names are for notational purposes, only). $\Box$

We consider only finite paths as they are enough to capture states and transitions that we need to specify the mappings between operational and control behaviors.

Example 1:. *Let l^1 and l^2 be* `start` *and* `failure` *in Fig. 1 (b).* `not-activated` $\overset{l^1}{\to}$ `activated` $\overset{l^2}{\to}$ `aborted` *is a path in WeatherWS's operational behavior.*

Fig. 1 shows how a Web service could be modelled using independent control and operational behaviors that will be brought together at a later stage. The former behavior has a business-logic flavor, while the latter behavior has an execution flavor. By not separating these two behaviors, a service engineer can either merge them into one, or simply ignore one of them for example the operational behavior. In the first case, the resulting model will be cumbersome, unfocused, and difficult to verify. States of different natures like `access-failed` and `compensated` will be combined and the number of possible transitions between these states could explode. In the second case, one aspect of the design will be ignored (here execution) and no verification could be conducted. The states in the control behavior do not tell much if a Web service is either activated or compensated since all these states represent the activation of this Web service.

2.2. Both Behaviors Brought Together

We discuss how the operational behavior guides the execution of a Web service along with its control behavior. In fact, the states that a Web service takes on in the operational behavior, e.g., `done`, will make this Web service take on other appropriate states in the control behavior, e.g., `report-delivered`, and *vice-versa*. The process of taking on states and connecting both behaviors means establishing correspondences between the respective states of these behaviors. These correspondences implement the *mapping* step of our engineering approach and result in forming *conversation sessions*. The idea of this mapping is to associate a given state in the operational behavior with a set of possible paths in the control behavior. To this end, we define a mapping operation as follows:

Definition 3 (Mapping Operation): Let $\mathcal{P}_{co}$ be the set of paths in the control behavior of a Web service starting by any state in this behavior. The mapping operation is defined using the following function: $Map : \mathcal{S}_{op} \to 2^{\mathcal{P}_{co}}$. $\Box$

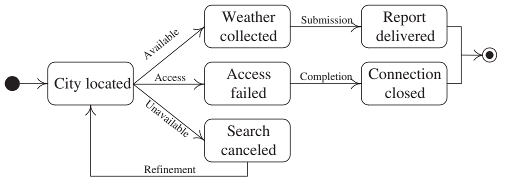
The mapping function Map associates each state in the operational behavior with a set (possibly empty) of possible paths in the control behavior ($2^{\mathcal{P}_{co}}$ is the power set of $\mathcal{P}_{co}$). The performance of this function is service engineer-based (Section 4). The intuition behind this definition is to simulate the execution of the operational behavior by the control behavior. Consequently, the Web service is correctly designed if such a simulation is possible.

Example 2:. *Fig. 2 is a mapping example in WeatherWS where* `activated` *state in the operational behavior is associated with different paths in the control behavior. One of these paths is:* `city-located` $\overset{l^1}{\to}$ `weather-collected` $\overset{l^2}{\to}$ `report-delivered` *where* $l^1 =$ `available` *and* $l^2 =$ `submission`.

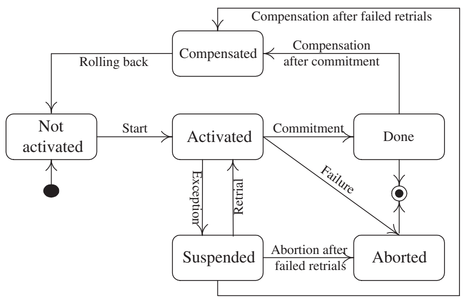
Interactions between operational and control behaviors are specified as part of the exercise of working out the conversation sessions. The purpose here is to keep both behaviors synchronized and hence, complete the *specification* step of our engineering approach. By specification, we mean how and when a state in the operational behavior communicates with other states in the control behavior and *vice-versa*. This communication is illustrated via the transitions between this state and the associated paths that the mapping function Map produces and via the specification operation we define as follows:

Definition 4 (Specification Operation): Let $\mathcal{P}_{co}$ be the set of paths in a control behavior starting by any state in this behavior, and $\mathcal{L}_S$ be the set of labels associated with the transitions between operational and control behaviors. The specification operation is defined through the following two functions:
$Spec : \mathcal{S}_{op} \to 2^{\mathcal{L}_S \times \mathcal{P}_{co} \times \mathcal{L}_S}$ and $Next : \mathcal{S}_{op} \times \mathcal{P}_{co} \to \mathcal{L}_{op} \times \mathcal{S}_{op}$. $\Box$

The specification function $Spec$ associates each state s_{op} in the operational behavior with a (possibly empty) set of triples. Each triple contains: (i) the label of the transition from s_{op} to the first state in the control behavior of a mapped



(a) Control behavior



(b) Operational behavior

Figure 1. *WeatherWS*'s control and operational behaviors

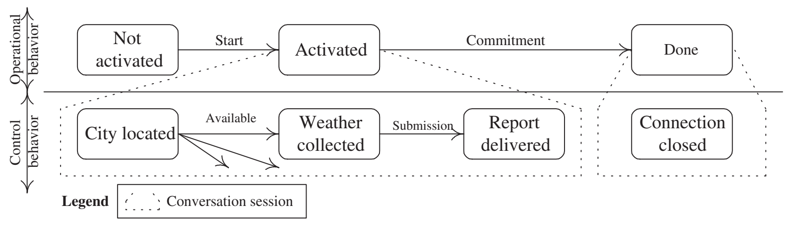


Figure 2. Example of operational and control behaviors mapping in *WeatherWS*

path, (ii) the mapped path itself, and (iii) the label of the transition from the last state in the control behavior of the mapped path back to s_{op}. We qualify transitions that connect states in independent finite state machines as *inter-behavior*. The partial function $Next$ associates a given state in the operational behavior and its mapped path in the control behavior with the next state to take on in the operational behavior and its associated transition label. The intuition behind this definition is to illustrates the simulation steps of the Web service execution by connecting its operational behavior to its control behavior from both sides.

Definition 5 (Soundness and Completeness): A Web service is sound and complete iff $Spec$ and $Next$ functions make each path from the initial state to a final state in the control behavior correspond to a path from the initial state to a final state in the operational behavior (soundness) and vice-versa (completeness). □

In Fig. 3, 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 transition, the execution of *WeatherWS* commences 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 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.

Cases a) and b) show how transitions between states in the operational behavior of a Web service are affected by the states and paths that this Web service takes on and executes respectively in the control behavior. *WeatherWS* moves from `activated` state to either `done` or `aborted` state based on information obtained out of the control behavior using either `report-delivered` state in $path_1$ or `connection-closed` state in $path_2$. The state to take on in the operational behavior, which is the function $Next$ returns, depends on the details that inter-behavior transitions carry over. 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 finite state machine (Definition 1).

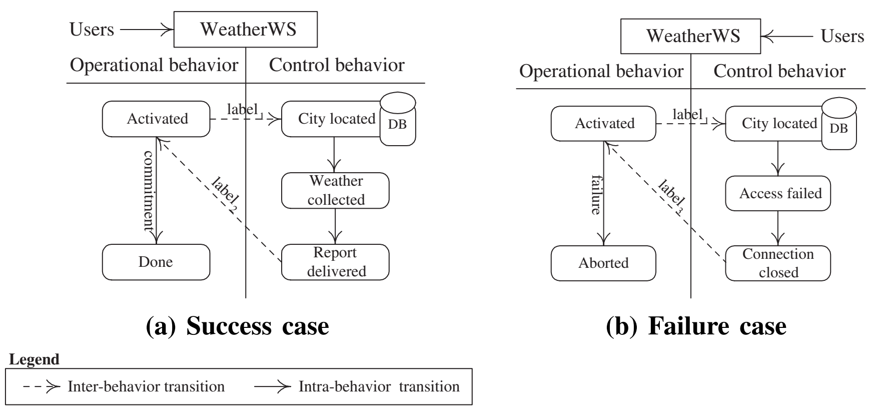


Figure 3. Synchronization of *WeatherWS*'s control and operational behaviors

*Definition 6 (***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.
- $\mathcal{L}_{op\rightarrow co}$ 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)). $\square$

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

*Definition 7 (***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:

- $s_{op} \in S_{op}$, $it_{op\rightarrow co} \in \mathcal{IT}_{op\rightarrow co}$, $it_{co\rightarrow op} \in \mathcal{IT}_{co\rightarrow op}$, $p_{co} \in \mathcal{P}_{co}$;
- $(Lab(it_{op\rightarrow co}), p_{co}, Lab(it_{op\rightarrow co})) \in Spec(s_{op})$. $\square$

This definition gives us the whole elements we need to specify the synchronization between the control and operational behaviors.

3. Synchronization of Both Behaviors

The *synchronization* step of our engineering approach is concerned with how inter-behavior transitions, like those in Fig. 3, are structured and executed. These transitions are the cornerstone of the conversation sessions between operational and control behaviors. In this section, we propose the list of conversational messages that implement inter-behavior transitions and then discuss how such messages are put together to form sequences of conversational messages. During the exercise of binding operational states to control paths, conversation sessions are identified (Fig. 2). These sessions characterize sequences of conversational messages (or sequences of inter-behavior transitions). A sequence of conversational messages is an ordered list of messages that are put together in a consistent way. Capturing such sequences generates the *execution traces* of a Web service and turns out to be very useful for post-analysis activities. An execution trace helps review all the actions that a Web service performed from operational and control perspectives. For instance, *delay* messages can be examined so that similar and frequent cases in the control behavior are addressed. In addition, $fail$ messages might indicate a reliability problem in the control behavior requiring corrective actions. The following are some possible sequences of messages where . stands for "next":

- *sync.success* (resp. *sync.fail*) refers to synchronization followed by success (resp. failure).
- *sync.delay.syncreq.sync.success* refers to synchronization followed by delay then by request of re-synchronization then by synchronization then by success.

All possible sequences of conversational messages can be represented using a combination of if-then rules.

Definition 8 (Sequences of Conversational Messages): Let n be the number of conversational messages exchanged during a session. Also, let $m_1, \ldots, m_7$ be conversational

Table 1. Messages implementing inter-behavior transitions

#	Message type	Performative Category	Description
1.	*sync*	Activation	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*	Outcome	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*.

messages of Table 1, and $m_i(t)$ $(i \in \{1, \dots 7\})$ be a conversational message sent at time t from a state in the operational behavior (resp. the control behavior) to a state in the control behavior (resp. the operational behavior). All possible sequences of conversational messages can be represented using a combination of rules of the form:

$$\forall t \in \{0, \dots, n-2\}, m_i(t) \Rightarrow \bigvee_{j \in \mathcal{J}} m_j(t+1)$$

where $\mathcal{J} \subseteq \{1, \dots, 7\}$, $i \neq j$, $m_i(0) = sync \vee m_i(0) = ping$, $m_i(n-1) = success \vee m_i(n-1) = fail \vee m_i(n-1) = ack$. $\Box$

The right side of the rule defines the possible continuations after the input m_i. Using these rules we specify some conditions that would make sequences of messages *well-formed*. For instance, to avoid deadlock situations like *sync.delay.delay...*, we put some restrictions on the way these sequences are developed. Examples of some restrictions are as follows:

- Each *sync* message in a sequence should be followed by either a *success*, *fail*, or *delay* message for the sake of synchronization matching. Formally, *sync*$(t) \Rightarrow$ *success*$(t+1) \vee$ *fail*$(t+1) \vee$ *delay*$(t+1)$.
- Each *ping* message in a sequence should be followed by an *ack* message for the sake of synchronization matching. Formally, *ping*$(t) \Rightarrow$ *ack*$(t+1)$.

A well-formed sequence of conversational messages offers some benefits that make the engineering of Web services sound and complete. For instance, it would be possible to (i) determine if a Web service's operational and control states were properly executed (e.g., a Web service is not indefinitely waiting for an *a*ck), (ii) to detect errors at design-time (e.g., having a *sync* without a corresponding synchronization matching transition), and (iii) to verify Web services' non-functional properties (e.g., *delay* messages submitted upon response-time verification).

Theorem 1: A Web service in which all the possible sequences of conversational messages between control and operational behaviors terminate by success, fail or ack is sound and complete.

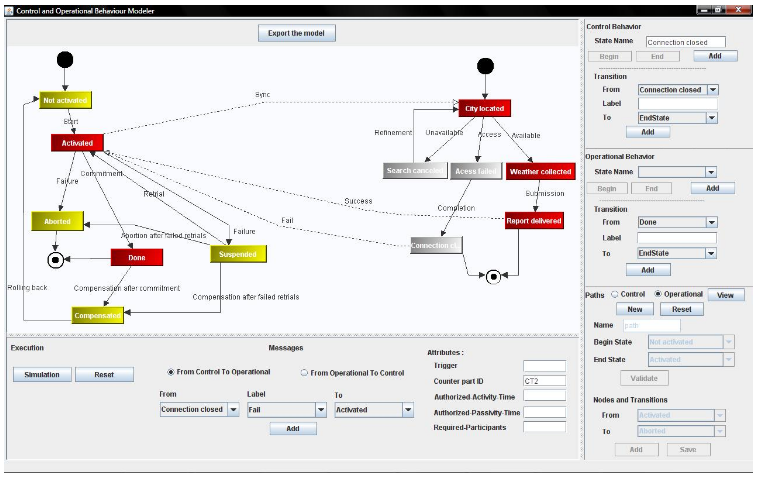
The rationale of this theorem is to check if a Web service is correctly designed. On the one hand, Definition 5 suggests that a Web service is sound and complete if the use of $Spec$ and $Next$ functions permits correspondence of its behaviors. On the other hand, the theorem provides a mechanism to validate this correspondence by simply checking the sequences of conversational messages between these behaviors. Techniques such as model checking and equivalence checking could be used for this purpose, but this is outside this paper's scope.

4. Implementation

The current prototype permits to define ("isolated") Web services' control/operational behaviors and inter-behavior transitions. The technologies we used are: XML to define behaviors and conversational messages, JDK1.6 to program conversation, control, and operational behavior managers, W3C.dom to process XML files, and Eclipse3.2 to integrate all the different modules into a CASE-like Web services development platform. The main modules in the prototype are as follows. Service designers access the system via a Web-based user interface. The `ControlBehaviorModeler` and the `OperationalBehaviorModeler` let designers specify the control and operational behaviors of a Web service, respectively. The `ConversationModeler` takes the behavior specifications of a Web service and the different allowed conversation specifications (i.e., inter-transitions and message sequences between two types of behaviors) to execute them. All these specifications are translated into XML documents, by the corresponding modeler, for subsequent processing. The `ServiceManager` provides functionalities for discovering and executing Web services. Together with the `ConversationController`, the execution of a Web service can be tracked and analyzed (if necessary) according to its conversation definition (e.g., whether the messages are received and sent in an appropriate order). Fig. 4 suggests an overview of the prototype's capabilities. Upon user request reception, *WeatherWS* moves from `not-activated to activated` states in the operational behavior (i.e., the left statechart diagram). In this latter state, *WeatherWS* submits a *sync* message along with necessary details to `city-located` state in the control behavior (i.e., the right statechart diagram). After execution, `report-delivered` state returns a *success* message back to `activated` state in the operational behavior.

5. Conclusion

In this paper we discussed how Web services could be engineered based on their control and operational behaviors.



Messages

(● From Control To Operational) ○ From Operational To Control

From: Connection closed
Label: Fail
To: Activated

Attributes :
Trigger:
Counter part ID: CT2
Authorized-Activity-Time:
Authorized-Passivity-Time:
Required-Participants:

Control Behavior
State Name: Connection closed
Transition
From: Connection closed
Label:
To: EndState

Operational Behavior
State Name:
Transition
From: Done
Label:
To: EndState

Paths ○ Control (● Operational)
Name: path
Begin State: Not activated
End State: Activated
Nodes and Transitions
From: Activated
To: Aborted

Figure 4. Web service behaviors specification

The former shows the business logic of the functionality that a Web service offers to users and peers, and the latter guides the execution progress of this control behavior through the actions that can be taken and the constraints that need to be satisfied. To the best of our knowledge, this is the first work in which similar behaviors are combined and made accessible to each other ("interactible") through a set of conversational messages. Some research projects focus on either the operational behavior [12] or the control behavior of Web services [2], [4], [9], while other projects do not even differentiate between both behaviors. Combining control and operational behaviors raised various challenges such as how to model them, how to synchronize them, how to maintain their consistency, and how to track them. We addressed these challenges through models that separate operational and control behaviors of Web services, mechanisms that support the synchronization of both behaviors using conversational messages, performatives that implement these conversational messages, and last but not least the verification of the correctness of the sequences of conversational messages. Additional challenges as future work were identified by for example binding the operational behavior of a Web service to some transactional properties. These properties limit the actions that are included in the control behavior of this Web service.

References

[1] L. Ardissono, A. Goy, and G. Petrone. Enabling Conversations with Web Services. In *Proceedings of The 2nd International Joint Conference on Autonomous Agents & Multi-Agent Systems (AAMAS'2003)*, Melbourne, Australia, 2003.

[2] B. Benatallah, F. Casati, and F. Toumani. Web Service Conversation Modeling, A Cornerstone for E-Business Automation. *IEEE Internet Computing*, 8(1), January 2004.

[3] K. P. Birman. Like It or Not, Web Services Are Distributed Objects. *CACM*, 47(12), December 2004.

[4] W. De Pauw, R. Hoch, and Y. Huang. Discovering Conversations in Web Services Using Semantic Correlation Analysis. In *Proceedings of The International Conference on Web Services (ICWS'2007)*, Salt Lake City, Utah, USA, 2007.

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

[6] Y. Kambayashi and H. F. Ledgard. The Separation Principle: A Programming Paradigm. *IEEE Software*, 21(2), March/April 2004.

[7] B. Limthanmaphon and Y. Zhang. Web Service Composition Transaction Management. In *Proceedings of the 14th Australasian Database Conference (ADC'2004)*, Dunedin, New Zealand, 2004.

[8] M. Little. Transactions & Web Services. *CACM*, 46(10), October 2003.

[9] Z. Maamar, Q. Z. Sheng, and B. Benatallah. Towards a Conversation-Driven Composition of Web Services. *Web Intelligence and Agent Systems*, 2(2), 2004.

[10] B. Medjahed and Y. Atif. Context-based Matching for Web Service Composition. *Distributed and Parallel Databases, Springer*, 21(1), January 2007.

[11] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 40(11), November 2007.

[12] 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.