Augmenting Web Services Composition with Transactional Requirements

Frederic Montagut
SAP Labs France, Institut Eurecom
805, Avenue du Docteur Maurice Donat
Font de l’Orme, 06250 Mougins, France
frederic.montagut@sap.com

Refik Molva
Institut Eurecom
2229 Route des Cretes
06904 Sophia-Antipolis, France
refik.molva@eurecom.fr

Abstract

Current Web services composition approaches do not take into account transactional requirements defined by designers. The transactional challenges raised by the composition of Web services are twofold: relaxed atomicity and dynamicity. In this paper, we propose a new process to automate the design of transactional composite Web services. Our solution enables the composition of Web services not only according to functional requirements but also to transactional ones defined using the Acceptable Termination States model. The resulting composite Web service is compliant with the consistency requirements expressed by designers and its execution can easily be coordinated using the coordination rules provided as an outcome of our approach.

1. Introduction

Web services composition has been gaining momentum over the last years as it leverages the capabilities of simple operations to offer complex services. These complex services such as airline booking systems result from the aggregation of Web services offered by different organizations. As for all cross-organizational collaborative systems, the execution of composite services requires transactional properties (TP) so that the overall consistency of data modified during the process is ensured. Yet, existing Web services composition systems appear to be limited when it comes to integrate at the composition phase the consistency requirements defined by designers. Composite Web services indeed require different transactional approaches than the one developed for usual database systems [6, 7]. The transactional challenges raised by the composition of Web services are twofold. First, like classical workflow systems, composite services raise less stringent requirements for atomicity in that intermediate results produced by some components may be kept without rollback despite the failure to complete the overall execution of a composite service. Second, composite services are dynamic in that their components can be automatically selected at run-time based on specific requests. To cope with these challenges, the existing approaches only offer means to validate transactional requirements (TR) once a composite Web service has been created [4] but no solution to integrate these requirements as part of the composite service building process.

In this paper, we propose a systematic procedure to automate the design of transactional composite Web services. Given an abstract representation of a process wherein instances of services are not yet assigned to component tasks, our solution enables the selection of Web services not only according to functional requirements but also to transactional ones. In our approach, TR are defined by designers using the Acceptable Termination States model (ATS). The resulting composite Web service is compliant with the defined consistency requirements and its execution can be easily coordinated as our algorithm also provides coordination rules that can be plugged into a transactional coordination protocol. The remainder of the paper is organized as follows. Section 2 introduces the methodology of our approach. In section 3, we present our transactional model. In section 4 we provide details on the termination states of a composite Web service then in section 5 we describe how TR are formed based on the properties of the termination states. The transaction-aware composition process through which transactional composite Web services are designed is detailed in section 6. Finally, section 7 discusses related work and section 8 presents the conclusion.

2. Preliminary definitions and methodology

Consistency is a crucial aspect of composite services execution. In order to meet consistency requirements at early stages of the service composition process, we need to consider TR a concrete parameter determining the choice of the component Web services. In this section we present a high level definition of the consistency requirements and a methodology taking into account these requirements during the composition of Web services.
2.1. Consistent composite Web services

A composite Web service $W_s$ consists of a set of $n$ Web services $W_s = \{s_a\}_{a \in 1..n}$ whose execution is managed according to a workflow $W$ which defines the execution order of a set of $n$ tasks $W = \{t_a\}_{a \in 1..n}$ performed by these services (for the sake of simplicity, we consider that one service executes only one task). The assignment of services to tasks is performed by means of composition engines based on functional requirements. Yet, the execution of a composite service may have to meet $TR$ aiming at the overall assurance of consistency. Our goal is to design a service assignment process that takes into account the $TR$ assurance of consistency. Our goal is to design a service as-


tposite service may have to meet $TR$ whereas some other service does not have any of these capabilities. It is thus necessary to select the appropriate service to execute a task whose execution may be compensated if one of them fails. The chal-

\[\begin{align*}
\text{TP} &= \{\text{Retriable, Compensatable, Pivot}\} \\
\text{Retriable} &\quad \text{If the task has to be re-executed after a failure} \\
\text{Compensatable} &\quad \text{If the task has to be compensated after a failure} \\
\text{Pivot} &\quad \text{If the task is neither compensatable nor retriable}
\end{align*}\]

In this section, we define the semantic specifying the $TP$ offered by services before specifying the consistency evaluation tool associated to this semantic. Our semantic model is based on the "transactional Web service description" defined in [4].

3. Transactional model

In this section, we define the semantic specifying the $TP$ of Web services is presented. This model is based on the classification of computational tasks made in [13, 15] which considers three different types of $TP$. An operation and by extension a Web service executing this task can be:

- Compensatable: the results produced by the task can be rolled back
- Retriable: the task is sure to complete successfully after a finite number of tries
- Pivot: the task is neither compensatable nor retriable

\[\text{TP} = \{\text{Retriable, Compensatable, Pivot}\} \]

These $TP$ allows to define four types of services: Retriable ($r$), Compensatable ($c$), Retriable and Compensatable ($rc$) and Pivot ($p$).

To properly detail the model, we can map the $TP$ with the state of the data modified by the services during the execution of computational tasks. This mapping is depicted in Figure 2. Basically, data can be in three different states: initial (0), unknown (x), completed (1). In the state (0), either the task execution has not yet started initial, the execution has been stopped, aborted before starting, or the execution has been properly completed and the modifications have been rolled back, compensated. In state (1) the task execution has been properly completed. In state (x) either the task execution is not yet finished active, the execution has been stopped, canceled before completion, or the execution has failed. Particularly, the states aborted, compensated, completed, canceled, and failed are the possible final states of execution of these tasks. Figure 3
details the transition diagram for the four types of transactional services. We must distinguish within this model the inherent termination states: failed and completed which result from the normal course of a task execution and the one resulting from a coordination message received during a coordination protocol instance: compensated, aborted and canceled which force a task execution to either stop or rollback. The TP of the services are only differentiated by the states failed, and compensated which indeed respectively specify the retryability and compensatability aspects.

**Definition 3-1.** We have for a given service \( s \):
- failed is not a termination state of \( s \leftrightarrow s \) is retryable
- compensated is a termination state of \( s \leftrightarrow s \) is compensatable

From the state transition diagram, we can also derive some simple rules. The states failed, completed and canceled can only be reached if the service is in the state active. The state compensated can only be reached if the service is in the state completed. The state aborted can only be reached if the service is in the state initial.

### 3.2. Termination states

The crucial point of the transactional model specifying the TP of services is the analysis of their possible termination states. The ultimate goal is indeed to be able to define consistent termination states for a workflow i.e. determining for each service executing a workflow task which termination states it is allowed to reach.

**Definition 3-2.** We define the operator termination state \( ts(x) \) which specifies the possible termination states of the element \( x \). This element \( x \) can be:
- a service \( s \) and \( ts(s) \in \{ \text{aborted, canceled, failed, completed, compensated} \} \)
- a workflow task \( t \) and \( ts(t) \in \{ \text{aborted, canceled, failed, completed, compensated} \} \)
- a workflow composed of \( n \) tasks \( W = (t_a)_{a \in [1,n]} \) and \( ts(W) = (ts(t_1), ts(t_2), ..., ts(t_n)) \)
- a composite service \( W_s \) of \( W \) composed of \( n \) services \( W_s = (s_a)_{a \in [1,n]} \) and \( ts(W_s) = (ts(s_1), ts(s_2), ..., ts(s_n)) \)

The operator \( TS(x) \) represents the finite set of all possible termination states of the element \( x \), \( TS(x) = (ts_k(x))_{k \in [1,j]} \). We have especially, \( TS(W_s) \subseteq TS(W) \) since the set \( TS(W_s) \) represents the actual termination states that can be reached by \( W_s \) according to the TP of the services assigned to \( W \). We also define for \( x \) workflow or composite service and \( a \in [1,n] \):
- \( ts(x, t_a) \): the value of \( ts(t_a) \) in \( ts(x) \)
- \( tscomp(x) \): the termination state of \( x \) such that \( \forall a \in [1,n] ts(x, t_a) = \text{completed} \).

For the remaining of the paper, \( W = (t_a)_{a \in [1,n]} \) represents a workflow of \( n \) tasks and \( W_s = (s_a)_{a \in [1,n]} \) a composite service of \( W \).

### 3.3. Transactional consistency tool

We use the Acceptable Termination States (ATS) [14] model as the consistency evaluation tool for our workflow. ATS defines the termination states a workflow is allowed to reach so that its execution is judged consistent.

**Definition 3-3.** \( ATS(W) \) is the subset of \( TS(W) \) whose elements are considered consistent by workflow designers. A consistent termination state of \( W \) is called an acceptable termination state \( ats_k(W) \) and we note \( ATS(W) = \{ \text{ats}_k(W) \}_{k \in [1,q]} \) the set of Acceptable Termination States of \( W \) i.e. the TR of \( W \).

\( ATS(W) \) and \( TS(W) \) can be represented by a table which defines for each termination state the tuple of termination states reached by the workflow task as depicted in Figure 4. As mentioned in the definition, the specification of the set \( ATS(W) \) is done at the workflow designing phase. \( ATS(W) \) is mainly used as a decision table for a coordination protocol so that \( W_s \) can reach an acceptable termination state knowing the termination state of at least one task. The role of a coordination protocol indeed consists in sending messages to services in order to reach a consistent termination state given the current state of the workflow execution. The coordination decision, i.e. the termination state that has to be reached, made given a state of the workflow execution has to be unique, this is the main characteristic of a coordination protocol. In order to cope with this requirement, \( ATS(W) \) which is used as input for the coordination decision-making process has therefore to verify some properties that we detail later on.

### 4. Analysis of \( TS(W) \)

Since \( ATS(W) \subseteq TS(W) \), \( ATS(W) \) inherits the characteristics of \( TS(W) \) and we logically need to analyze first \( TS(W) \). In this section, we first precise some basic properties of \( TS(W) \) derived from inherent execution rules of a workflow \( W \) before examining \( TS(W) \) from a coordination perspective.

#### 4.1. Inherent properties of \( TS(W) \)

We state here some basic properties relevant to the elements of \( TS(W) \) and derived from the transactional model presented above. \( TS(W) \) is the set of all possible termination states of \( W \) based on the termination states model we chose for services. Yet, within a composite service execution, it is not possible to reach all the combinations represented by a \( n \)-tuple \( (ts(t_1), ts(t_2), ..., ts(t_n)) \) assuming \( \forall a \in [1,n] ts(t_a) \in \{ \text{aborted, canceled, failed, completed, compensated} \} \).

The first restriction is introduced by the sequential aspect of a workflow:

\( (P_1) \) A task becomes activated \( \Leftrightarrow \) all the tasks executed beforehand according to the execution plan of \( W \) have reached the state completed.
(P1) simply means that to start the execution of a workflow task, it is required to have properly completed all the workflow tasks required to be executed beforehand.

Second, we consider in our model that only one single task can fail at a time and that the states aborted, compensated and canceled can only be reached by a task in a given ts_k(W) if one of the services executing a task of W has failed. This means that the coordination protocol is allowed to force the abortion, the compensation or the cancellation only in case of failure of a service. We get (P2):

(P2) if ∃ a, b ∈ [1, n] × [1, j] such that ts_k(W, t_a) ∈ {compensated, aborted, canceled} ⇒ ∃ ! l ∈ [1, n] such that ts_k(W, t_l) = failed.

4.2. Classification within TS(W)

As we explained above the unicity of the coordination decision during the execution of a coordination protocol is a major requirement. We try here to identify the elements of TS(W) that correspond to different coordination decisions given the same state of a workflow execution. The goal is to use this classification to determine ATS(W). Using the properties P1 and P2, a simple analysis of the state transition model reveals that there are two situations whereby a protocol coordination has different possibilities of coordination given the state of a workflow task. Let a, b ∈ [1, n] and assume that the task t_a has failed:

- the task t_a is in the state completed and either it remains in this state or it is compensated
- the task t_a is in the state active and either it is canceled or the coordinator lets it reach the state completed

From these two statements, we define the incompatibility from a coordination perspective and the flexibility.

**Definition 4.1.** Let k, l ∈ [1, j]. ts_k(W) and ts_l(W) are said incompatible from a coordination perspective ⇔ ∃ a, b ∈ [1, n] such that ts_k(W, t_a) = completed, ts_l(W, t_b) = ts_l(W, t_b) = failed and ts_l(W, t_a) = compensated. Otherwise, ts_l(W) and ts_k(W) are said compatible from a coordination perspective.

The value in {compensated, completed} reached by a task t_a in a termination state ts_k(W) whereby ts_k(W, t_b) = failed is called recovery strategy of t_a against t_b in ts_k(W). By extension, we can consider the recovery strategy of a set of tasks against a given task.

If two termination states are compatible, they correspond to the same recovery strategy against a given task. In fact, we have two cases for the compatibility of two termination states ts_k(W) and ts_l(W). Given two tasks t_a and t_b such that ts_k(W, t_a) = ts_l(W, t_b) = failed:

- ts_k(W, t_a) = ts_l(W, t_a)
- ts_l(W, t_a) ∈ {compensated, completed} and ts_l(W, t_a) ∈ {aborted, canceled}

The second case is only possible to reach if t_a is executed in parallel with t_b. Intuitively, the failure of the service assigned to t_b occurs at different instants in ts_k(W) and ts_l(W).

**Definition 4.2.** Let a, b ∈ [1, n]. A task t_a is flexible against t_b ⇔ ∃ k ∈ [1, j] such that ts_k(W, t_b) = failed and ts_k(W, t_a) = canceled. Such a termination state is said to be flexible to t_a against t_b. The set of termination states of W flexible to t_a against t_b is denoted FTS(t_a, t_b).

From these definitions, we now study the termination states of W according to the compatibility and flexibility criteria in order to identify the termination states that follow a common strategy of coordination.

**Definition 4.3.** Let a ∈ [1, n]. A termination state of W ts_k(W) is called generator of t_a ⇔ ts_k(W, t_a) = failed and b ∈ [1, n] such that t_b is executed before or in parallel of t_a, ts_k(W, t_b) ∈ {completed, compensated}. The set of termination states of W compatible with ts_k(W) generator of t_a is denoted CTS(ts_k(W), t_a).

The set CTS(ts_k(W), t_a) specifies all the termination states of W that follow the same recovery strategy as ts_k(W) against t_a.

**Definition 4.4.** Let ts_k(W) ∈ TS(W) be a generator of t_a. Coordinating an instance W_a of W in case of the failure of t_a consists in choosing the recovery strategy of each task of W against t_a and the z_a < n tasks (t_a_i)_{i∈[1,z_a]} flexible to t_a whose execution is not canceled when t_a fails. We call coordination strategy of W_a against t_a the set CS(W_a, ts_k(W), (t_a_i)_{i∈[1,z_a]}, t_a) = CTS(ts_k(W), t_a) − ∪_{i=1}^{z_a} FTS(t_a_i, t_a). If the service s_a assigned to t_a is retrievable then CS(W_a, ts_k(W), (t_a_i)_{i∈[1,z_a]}, t_a) = ∅.

W_a is said to be coordinated according to CS(W_a, ts_k(W), (t_a_i)_{i∈[1,z_a]}, t_a) if in case of the failure of t_a, W_a reaches a termination state in

![Figure 3. Service state diagram according to their Transactional Properties](image)
CS(W_s, ts_k(W_s), (t_a_i)_{i\in[1,z_a]}, t_a). Of course, it assumes
that the TP of W_s are sufficient to reach ts_k(W).
From these definitions, we can deduce a set of properties:

**Theorem 4-5.** W_s can only be coordinated according to a
unique coordination strategy at a time.

**Proof:** Let a ∈ [1, n]. Two termination states ts_k(W) and ts_k(W) generator of t_a are incompatible.

**Theorem 4-6.** Let a, k ∈ [1, n] × [1, j] such that

\[
\begin{align*}
\text{ts}_k(W, t_a) = \text{failed} & \text{ and } ts_k(W) \in TS(W_a) \Rightarrow \exists i \in [1, j] \text{ such that ts}_i(W) \in TS(W_a) \text{ is a} \\
\text{generator of } t_a \text{ compatible with ts}_k(W).
\end{align*}
\]

**Proof:** We define ts_i(W) by: \(ts_i(W, t_a) = \text{failed}, \forall i \in [1, n] - \{a\} ts_i(W, t_i) = ts_k(W, t_i)\) if \(ts_k(W, t_i) \in \{\text{completed, compensated, aborted}\}, ts_i(W, t_i) = \text{completed} \) otherwise.

Given a task t_a the idea is to classify the elements of

\(TS(W)\) using the sets of termination states compatible with the generators of t_a. Using this approach, we can identify the different recovery strategies and the coordination strategies associated with the failure of t_a as we decide which tasks can be canceled.

5. Forming ATS(W)

Defining ATS(W) is deciding at design time the termination states of W that are consistent. ATS(W) is to be inputted to a coordination protocol in order to provide it with a set of rules which leads to a unique coordination decision in any cases. According to the definitions and properties we introduce above, we can now explicit some rules on ATS(W) so that the unicity requirement of coordination decisions is respected.

**Definition 5-1.** Let a, k ∈ [1, n] × [1, j] such that

\[
\begin{align*}
\text{ts}_k(W, t_a) = \text{failed} & \text{ and } ts_k(W) \in ATS(W), ATS(W) \text{ is valid } \iff \exists l \in [1, j] \text{ such that ts}_l(W) \text{ generator of } t_a \text{ compatible with ts}_k(W) \text{ and CTS}(ts_l(W), t_a) - \bigcup_{i=1}^{n} FTS(ts_a_i, t_a) \subset ATS(W) \text{ for a set of tasks } (t_a_i)_{i\in[1,z_a]} \text{ flexible to } t_a.
\end{align*}
\]

The unicity of the termination state generator of a given
task comes from the incompatibility definition and the unicity of the coordination strategy. A valid ATS(W) therefore contains for all ts_k(W) in which a task fails a unique coordination strategy associated to this failure and the termination states contained in this coordination strategy are compatible with ts_k(W). In Figure 4, an example of possible ATS is presented for the simple workflow W_1. It just consists in selecting the termination states of the table TS(W_1) that we consider consistent and respect the validity rule for the created ATS(W_1).

6. Deriving composite services from ATS

In this section, we introduce a new type of service assignment procedure; the transaction-aware service assignment procedure which aims at assigning n services to the n tasks t_a in order to create an instance of W acceptable with respect to a valid ATS(W). The goal of this procedure is to integrate within the instantiation process of workflows a systematic method ensuring the transactional consistency of the obtained composite service. We first define a validity criteria for the instance W_s of W with respect to ATS(W), the service assignment algorithm is then detailed. Finally, we specify the coordination strategy associated to the instance created from our assignment scheme.

6.1. Acceptability of W_s with respect to ATS(W)

**Definition 6-1.** W_s is an acceptable instance of W with respect to ATS(W) ⇔ TS(W_s) ⊆ ATS(W).

Now we express the condition TS(W_s) ⊆ ATS(W) in terms of coordination strategies. The termination state generator of t_a present in ATS(W) is noted ts_k(W). The set of tasks whose execution is not canceled when t_a fails is noted (t_a_i)_{i\in[1,z_a]}.

**Theorem 6-2.** TS(W_s) ⊆ ATS(W) ⇐ ∀ a ∈ [1, n] CS(W_s, ts_k(W), (t_a_i)_{i\in[1,z_a]}, t_a) ⊂ ATS(W).

**Proof:** straightforward derivation from 4-6 and 5-1.

An instance W_s of W is therefore an acceptable one ⇐ it is coordinated according to a set of n coordination strategies contained in ATS(W). It should be noted that if failed \(\notin ATS(W, t_a)\) where ATS(W, t_a) represents the acceptable termination states of the task t_a in ATS(W) then CS(W_s, ts_k(W), (t_a_i)_{i\in[1,z_a]}, t_a) = 0. From 4-6 and 6-1, we can derive the existence condition of an acceptable instance of W with respect to a valid ATS(W).

**Theorem 6-3.** Let a, k ∈ [1, n] × [1, j] such that

\[
\begin{align*}
\text{ts}_k(W, t_a) = \text{failed} & \text{ and } ts_k(W) \in ATS(W). \exists W_s \text{ acceptable instance of } W \text{ with respect to ATS(W) such that } ts_k(W) \in TS(W_s) \iff \exists l \in [1, j] \text{ such that ts}_l(W) \in TS(W_s) \text{ is a generator of } t_a \text{ compatible with ts}_k(W) \text{ in } ATS(W).
\end{align*}
\]

This theorem only states that an ATS(W) allowing the failure of a given task can be used to coordinate a composite service also allowing the failure of the same task ⇐ ATS(W) contains a complete coordination strategy associated to this task, i.e. it is valid.

6.2. Transaction-aware assignment procedure

In this section, we present the procedure that is used to assign services to tasks based on TR. This algorithm uses ATS(W) as a set of requirements during the service assignment procedure and thus identifies from a pool of available services those whose TP match the TR associated to workflow tasks defined in ATS(W) in terms of acceptable termination states. The assignment procedure is an iterative process, services are assigned to tasks one after the other. The assignment procedure therefore creates at each step i a partial instance of W noted W_s^i. We can define as well the set TS(W_s^i) which represents the termination states of
W that the TP of the i services already assigned allow to reach. Intuitively the acceptable termination states refer to the degree of flexibility offered when choosing the services with respect to the different coordination strategies verified in ATS(W). This degree of flexibility is influenced by two parameters:

- The list of acceptable termination states for each workflow task. This list can be determined using ATS(W). This is a direct requirement which specifies the termination states allowed for each task and therefore introduces requirements on the service’s TP to be assigned to a given task: this service can only reach the states defined in ATS(W) for the considered task.

- The assignment process is iterative and therefore, as we assign new services to tasks, TS(W,i) changes and the TP required to the assignment of further services too. For instance, we are sure to no longer reach the termination states CTS(tsk(W), t_a) allowing the failure of the task t_a in ATS(W) when we assign a service (r) to t_a. In this specific case, we no longer care about the states reached by other tasks in CTS(tsk(W), t_a) and therefore there is no TR introduced for the tasks to which services have not already been assigned.

We therefore need to define first the TR for the assignment of a service after i steps in the assignment procedure.

6.2.1. Extraction of TR. From the two requirements above, we define for a task t_a:

- ATS(W,t_a): Set of acceptable termination states of t_a which is derived from ATS(W)

- DIS(t_a, W_i): This is the set of TR that the service assigned to t_a must meet based on the previous assignments. This set is determined based on the following reasoning:

\[(DIS_1): \text{ the service must be compensatable } \iff \text{compensated} \in DIS(t_a,W_i)\]

\[(DIS_2): \text{ the service must be retriable } \iff \text{failed} \notin DIS(t_a,W_i)\]

Using these two sets, we are able to compute

\[\text{Min}_{TP}(s_a,t_a,W_i) = ATS(W,t_a) \cap DIS(t_a,W_i)\]

which defines the TP a service s_a has at least to comply with in order to be assigned to the task t_a at the i + 1 assignment step. We simply check the retrievability and compensatability properties for the set \(\text{Min}_{TP}(s_a,t_a,W_i)\):

- \(\text{failed} \notin \text{Min}_{TP}(s_a,t_a,W_i)\) \(\iff s_a\) has to verify the retrievability property

- \(\text{compensated} \in \text{Min}_{TP}(s_a,t_a,W_i)\) \(\iff s_a\) has to verify the compensatability property

The set ATS(W,t_a) is easily derived from ATS(W). We need now to compute DIS(t_a, W_i). We assume that we are at the i + 1 step of an assignment procedure, i.e. the current partial instance of W is W_i. Computing DIS(t_a, W_i) means determining if (DIS_1) and (DIS_2) are true. From these two statements we can derive three properties:

1. (DIS_1) implies that state compensated can definitely be reached by t_a

2. (DIS_2) implies that t_a can not fail

3. (DIS_2) implies that t_a can not be canceled

The two first properties can be directly derived from (DIS_1) and (DIS_2). The third one is derived from the fact that if a task can not be canceled when a task fails, then it has to finish its execution and reach at least the state completed. In this case, if a service can not be canceled then it can not fail, which is the third property. To verify whether 1., 2. and 3. are true, we introduce the theorems 6-4, 6-5 and 6-6.

**Theorem 6-4.** Let \(a \in [1, n]\). The state compensated can definitely be reached by \(t_a \iff \exists b \in [1, n] - \{a\}\)
verifying (6-4b): \( s_b \) not retriable is assigned to \( t_b \) and \( \exists t_s(k(W)) \in AT S(W) \) generator of \( t_b \) such that \( t_s(k(W), t_a) = \text{compensated} \).

**Proof:** \( \Leftarrow \): Since the service \( s_b \) is not retriable, it can fail and \( t_s(k(W)) \in AT S(W) \) generator of \( t_b \) such that \( t_s(k(W), t_a) = \text{compensated} \) is in \( TS(W_s) \).

\( \Rightarrow \): Derived from (P3) and 6-4.

The two following theorems are proved similarly:

**Theorem 6-5.** Let \( a \in [1, n] \), \( t_a \) can not fail \( \Leftrightarrow \exists b \in [1, n] - \{ a \} \) verifying (6-5b): \( s_b \) not compensatable is assigned to \( t_b \) and \( \exists t_s(k(W)) \in AT S(W) \) generator of \( t_b \) such that \( t_s(k(W), t_b) = \text{compensated} \) or \( t_b \) is flexible to \( t_a \) and \( s_b \) not retriable is assigned to \( t_b \) and \( \forall t_s(k(W)) \in AT S(W) \) such that \( t_s(k(W), t_a) = \text{failed}, t_s(k(W), t_b) \neq \text{canceled} \).

**Theorem 6-6.** Let \( a, b \in [1, n] \) such that \( t_a \) is flexible to \( t_b \), \( t_a \) is not canceled when \( t_b \) fails \( \Leftrightarrow \exists b \in [1, n] - \{ a \} \) verifying (6-6b): \( s_b \) not retriable is assigned to \( t_b \) and \( \forall t_s(k(W)) \in AT S(W) \) such that \( t_s(k(W), t_b) = \text{failed}, t_s(k(W), t_a) \neq \text{canceled} \).

According to the theorems 6-4, 6-5 and 6-6, in order to compute \( DIS(t_a, W_s) \), we have to compute \( t_b \) with each of the \( i \) tasks \( t_i \in W - \{ t_a \} \) to which a service \( s_b \) has been already assigned. This is an iterative procedure and at the initialisation phase, since no task has been yet compared to \( t_a, s_b \) can be (p): \( DIS(t_a, W_s) = \{ \text{failed} \} \).

1. if \( t_b \) verifies (6-4b) \( \Rightarrow \text{compensated} \in DIS(t_a, W_s) \)
2. if \( t_b \) verifies (6-5b) \( \Rightarrow \text{failed} \notin DIS(t_a, W_s) \)
3. if \( t_b \) is flexible to \( t_a \) and verifies (6-6b) \( \Rightarrow \text{failed} \notin DIS(t_a, W_s) \)

The verification stops if \( \text{failed} \notin DIS(t_a, W_s) \) and \( \text{compensated} \in DIS(t_a, W_s) \). With \( \min_{T P}(s_a, t_a, W_s) \), we are able to select the appropriate service to be assigned to a given task according to \( TR \).

**6.2.2. Service assignment process.** Services are assigned to each workflow task based on an iterative process. Depending on the \( TR \) and the \( TP \) of the services available for each task, different scenarios can occur:

(i) services (rc) are available for the task. It is not necessary to compute \( TR \) as such services match all \( TR \).

(ii) only services (p) are available for the task. We need to compute the \( TR \) associated to the task and either pivot is sufficient or there is no solution.

(iii) services (r) and (c) but no (rc) are available for the task. We need to compute the \( TR \) associated to the task and we have three cases. First, (retriebility and compensatability) is required in which case there is no solution. Second, retriebility (resp. compensatability) is required and we assign a service (r) (resp. (c)) to the task. Third, there is no requirement.

The idea is therefore to assign first services to the tasks verifying (i) and (ii) since there is no flexibility in the choice of the service. Tasks verifying (iii) are finally analyzed. Based on the \( TR \) raised by the remaining tasks, we first assign services to tasks with a non-empty \( TR \). We then handle the assignment for tasks with an empty \( TR \). Note that the \( TR \) of all the tasks to which services are not yet assigned are also affected (updated) as a result of the current service assignment. If no task has \( TR \) then we assign the services (r) to assure the completion of the remaining tasks’ execution.

**Theorem 6-7.** The service assignment procedure creates an instance of \( W \) that is acceptable with respect to a valid \( AT S(W) \).

**Proof:** Let \( W_s \) be an instance of \( W \) resulting from the service assignment procedure and a service \( s_a \) assigned to a task \( t_a \) in \( W_s \). The definition 6-1 has to be verified and we therefore consider (A) and (B) (with the notations of 6-2):

(A) \( \forall t_a \in [1, n], \text{failed} \in AT S(W, t_a) \Rightarrow CS(W_s, t_s(k(W), (t_a)_i \in [1, z_a], t_a) \subseteq AT S(W) \)

(B) \( \forall t_a \in [1, n], \text{failed} \notin AT S(W, t_a) \Rightarrow CS(W_s, t_s(k(W), (t_a)_i \in [1, z_a], t_a) \subseteq AT S(W) \)

(A): We suppose that \( \text{failed} \notin AT S(W, t_a) \) then we have two possibilities: \( s_a \) is retriable and \( CS(W_s, t_s(k(W), (t_a)_i \in [1, z_a], t_a) = \emptyset \subset AT S(W) \). \( s_a \) can fail and with 1, 2 and 3 we get \( t_s(k(W)) \in TS(W_s) \) and therefore \( CS(W_s, t_s(k(W), (t_a)_i \in [1, z_a], t_a) \subset AT S(W) \) since \( AT S(W) \) is valid.

(B): We suppose that \( \text{failed} \notin AT S(W, t_a) \) then \( \text{failed} \notin \min_{T P}(s_a, t_a, W_s) \) and \( s_a \) is retriable. Therefore, \( CS(W_s, t_s(k(W), (t_a)_i \in [1, z_a], t_a) = \emptyset \subset AT S(W) \). \( CS(W_s, t_s(k(W), (t_a)_i \in [1, z_a], t_a) \subset AT S(W) \) and \( W_s \) is an acceptable instance of \( W \) with respect to \( AT S(W) \).

**6.3. Coordination of \( W_s \)**

Now, using (A) and (B) defined in the proof of 6-7 and keeping the same notations, we are able to specify the coordination strategy of \( W_s \) against each workflow task. We get indeed the following theorem.

**Theorem 6-8.** Let \( W_s \) be an acceptable instance of \( W \) with respect to \( AT S(W) \). We note \( (t_a)_i \in [1, n] \) the set of tasks to which no retriable services have been assigned. \( TS(W_s) = \{ t_{\text{comp}}(W_s) \} \cup \bigcup_{i=1}^{z_a}(CT S(t_s(k_1(W), t_a)) - \bigcup_{j=1}^{z_a} FTS(t_{n_1(a)}, t_{n_1(a)}) \).

Having computed \( TS(W_s) \), we can deduce the coordination rules associated to the execution of \( W_s \).

**6.4. Example**

We consider the workflow \( W_1 \) of Figure 4. Designers have defined \( AT S_2(W_1) \) as the \( TR \) and the set of available services for each task of \( W_1 \) is specified in the Figure. The goal is to assign services to workflow tasks so that the instance of \( W_1 \) is valid with respect to \( AT S_2(W_1) \) and we apply the presented assignment procedure. We first start to assign the services (rc) for which it is not necessary to compute any \( TR \). \( s_{31} \) which is the only available service (rc) is therefore assigned to task 3. We then try to assign the services (p), and we verify whether \( s_{21} \) can be assigned to task 2. We compute
Min_{TP}(s_a, t_2, W^1_{t_1}) = ATS_2(W_1, t_2) \cap DIS(t_2, W^1_{t_1}).
ATS_2(W_1, t_2) = \{completed, compensated, failed\}
and DIS(t_2, W^1_{t_1}) = \{failed\} as s_{31} the only service already assigned is (re) and the theorems 6-4, 6-5 and 6-6 are not verified. Thus Min_{TP}(s_a, t_2, W^2_{t_1}) = \{failed\} and s_{32} can be assigned to task 2 as it matches the TR. Now we compute the TR of task 1 and we get Min_{TP}(s_a, t_1, W^2_{t_1}) = \emptyset, a service can therefore be assigned to task 1 if it is retrievable, which is the case of s_{11}. Finally, we compute the TR of task 4 and we get Min_{TP}(s_a, t_4, W^3_{t_1}) = \emptyset as theorem 6-5 is verified with the service s_{21}. The service s_{41} can thus be assigned to task 4 as it matches the TR of the task.

7. Related work

Transactional consistency of workflows and database systems has been an active research topic over the last 15 years yet it is still an open issue in the area of Web services [5, 8, 12] and especially composite Web services. Composite Web services indeed introduce new requirements for transactional systems such as dynamicity, semantic description and relaxed atomicity. Existing transactional models for advanced applications [6] are lacking of flexibility to integrate these requirements [3] as for instance they are not designed to support the execution of dynamically generated collaboration of services. In comparison, our solution allows the specification of TR supporting relaxed atomicity for an abstract workflow specification and the selection of semantically described services respecting the defined TR.

Our work is based on [4] which presents the first approach specifying relaxed atomicity requirements for Composite Web services based on the ATS tool and a transactional semantic. Despite a solid contribution, this work appears to be limited if we consider the possible integration into automatic Web services composition systems. It indeed only details transactional rules to validate a given composite service with respect to defined TR. In this approach, TR do not play any role in the component services selection process which may result in several attempts for designers to determine a valid composition of services. On the contrary, our solution provides a systematic procedure enabling the automatic design of transactional composite Web services. Besides, our contribution also defines the mathematical foundations to specify valid ATS for workflows using the defined concept of coordination strategy.

Finally, our solution can be used to augment recent standardization efforts lead in the area of transactional coordination of Web services [1, 11]. Our approach indeed provides adaptive coordination specifications based on the TP of the component services instantiating a given workflow. Existing Web services coordination specifications [9, 10] are indeed not flexible enough as they do not neither allow workflow designers to specify their TR nor take into account the TP offered by Web services.

8. Conclusion

We presented a systematic procedure to automate the design of transactional composite Web services. Our solution enables the selection of component Web services not only according to functional requirements but also to transactional ones. Transactional requirements are defined by designers and serve as an input to define both reliable composite Web services and coordination protocols used to ensure the consistency of their execution. On the one hand this service assignment approach can be used to augment existing Web services composition systems [2] as it can be fully integrated in existing functional match-making procedures. On the other hand, our approach defines adaptive coordination rules that can be deployed on Web services coordination specifications [11] in order to increase their flexibility.

References