Web Service Selection for Transactional Composition

Yudith Cardinale\textsuperscript{a}, Joyce El Haddad\textsuperscript{b}, Maude Manouvrier\textsuperscript{b,c,*}, Marta Rukoz\textsuperscript{b,c,d,**}

\textsuperscript{a}Universidad Simón Bolívar, Departamento de Computación y T. I., Apartado 89000, Caracas 1080-A, Venezuela
\textsuperscript{b}Université Paris-Dauphine, LAMSADE, CNRS FRE 3234, Place de Lattre de Tassigny, F-75016 Paris, France
\textsuperscript{c}WISDOM, Federation of the three database research teams, LIP6 (Paris 6 Univ.), LAMSADE and CEDRIC (CNAM)
\textsuperscript{d}Université Paris Ouest Nanterre La Défense, 200 Avenue de la République, 92001 Nanterre, France

Abstract

In this work we address the issue of selecting and composing Web Services (WSs) considering functional requirements and transactional properties. We formalize the WS composition problem using the user query, the transactional properties of WSs and the composite WSs definition. We extend Colored Petri Net (CPN) formalism to incorporate transactional WSs properties. We present a CPN-WS selection algorithm that satisfies the user query functional conditions expressed as input and output attributes, and transactional properties expressed as a risk level. The result of our algorithm is a Colored Petri-Net allowing to execute a transactional composite WS.

Keywords: Web Services, Automatic Transactional Composition, Colored Petri Net

1. Introduction

Web Services (WSs) are the most famous implementation of Service-Oriented Architectures (SOA) allowing the construction and the sharing of independent and autonomous software. To ensure efficient implementations of compositions, we have to adequately manage heterogeneity, i.e., diverse functionality (e.g., ticket purchase, payment) and diverse transactional behavior (e.g., compensatable or not) granted by multiple distributed services delivering the same functionality. The interoperation of distributed software-systems is always affected by failures, dynamic changes, availability of resources, and others. These effects are non-functional aspects and are caused by the nature of distributed software-systems [1]. In this context, a service that does not provide a transactional property might be as useless as a service not providing the desired functional results. If the composition is based on simple WSs considering only functional requirements, then it is possible that during the execution, the whole system becomes inconsistent in presence of failures. Selection of transactional WSs allows the system to guaranty reliable composition execution. Indeed, the execution of transactional WSs will leave the system in a consistent state even in presence of failures.

The Transactional WS composition problem has been extensively treated in the literature by using a predefined control structure such as workflows [1, 2, 3], Advanced Transactional Models...
(ATM) [4, 5, 6], or discovering algorithms based on graph representing the input and output relationships among WSs [7]. In workflows, the execution control is defined by the structure of the workflow, while in ATM approaches, it is explicitly defined within the application logic. Both are difficult to maintain and hardly adaptable to different application requirements. In our approach, the execution control is discovered by the user’s query inputs/outputs and the registry containing all possible interactions among WSs. In fact, we use an OWL-S ontology and transactional properties embedded in a Colored Petri Net (CPN) to model and construct a solution. The CPN models the services and their dependencies.

In this work, we propose a hybrid solution that takes advantages of search meta-heuristics techniques to consider functional conditions expressed as input and output attributes, and transactional properties expressed as a risk level. CPN model allows to describe not only a static vision of a system, but also its dynamic behavior; it is expressive enough to capture the semantics of complex WS combinations and their respective interactions. We extend CPN formalism to incorporate transactional WSs properties and adapt a Petri Net unfolding algorithm to perform a Best-First Search which stops when a desired marking, reachable from an initial marking in the CPN, is found. The initial marking represents the input attributes and the risk level provided in the query. The unfolding method is guided by the aggregated transactional properties, therefore, its result corresponds to a composition of WSs that satisfies the functional and risk level user requirements. We formalize the Transactional WS composition problem adopting from [8] the query definition and taking from [3] the definitions of transactional properties of WSs and composite WSs. This article presents our CPN based algorithm.

2. Definitions

In this section we define the framework of our approach by presenting our formal definitions.

Def. 1. Query. Let Onto be the integrated ontology1. A query Q is a 3-tuple (IQ, OQ, RQ), where IQ = {i | i ∈ Onto is an input attribute}, OQ = {o | o ∈ Onto is an output attribute whose value has to be produced by the system}, and RQ is the execution risk, such that: RQ ∈ {R0, R1}. If RQ = R0, then the system guarantees that if the execution is successful, the obtained results can be compensated by the user (i.e., the user can execute another application that semantically undoes the previous one). If RQ = R1, then the system does not guarantee the result can be compensated by the user in case of successful execution. In both execution risk cases, if the execution is not successful, then no result is reflected to the system i.e., nothing is changed on the system.

The result of this query can be obtained by a Composite Web Service (CWS), i.e., a conglomeration of existing WSs interacting together to offer a new value-added service. The WSs composing a CWS are called components. In this article, WSs and CWSs are transactional, using the following definitions:

Def. 2. Transactional WS. Let s be a WS. s is said to be pivot (p) if once it successfully completes, its effects remains forever and cannot be semantically undone. If it fails then it has no effect at all. A completed pivot WS cannot be rolled back. s is compensatable (c) if it exists another WS, s′, which can semantically undo the execution of s. s is retriable (r) if it guarantees a successfully termination after a finite number of invocations. The retriable property can be combined with properties p and c defining pivot retriable (pr) WS and compensatable retriable (cr) WS. Thus, the transactional property (TP) of a WS s is TP(s) ∈ {p, pr, c, cr}.

1Many ontologies could be used and integrated.
that the WS registry is represented by a Petri Net – see for example [9, 10, 11, 12, 13]. In this article, we choose to model it by a Colored Petri-Net whose definition is modified in such way we only focus on transactional composition of WSs satisfying the user’s functional and risk level requirements. Therefore, we define the WS registry can be modeled by a Petri Net – see for example [9, 10, 11, 12, 13]. In this article, we choose to model it by a Colored Petri-Net whose definition is modified in such way we only focus on transactional composition of WSs satisfying the user’s functional and risk level requirements. Therefore, we define the

\[
\text{Def. 4. WSDN.} \quad \text{A WSDN is a 4-tuple } (A, S, F, \xi), \quad \text{where: } A \text{ is a finite non-empty set of places, corresponding to input and output attributes of the WSs in the registry such that } A \subset \text{Onto}; S \text{ is a finite set of transitions corresponding to the set of WSs in the registry;} F : (A \times S) \cup (S \times A) \rightarrow \{0, 1\} \text{ is a flow relation indicating the presence (1) or the absence (0) of arcs between places and transitions defined as follows:}\n\]

\[
\forall s \in S \ (\exists a \in A \mid F(a, s) = 1) \Leftrightarrow \text{a is an input place of } s \text{ and (} \exists a \in A \mid F(s, a) = 1) \Leftrightarrow \text{a is an output place of } s . \quad \xi \text{ is a color function such that } \xi : C_A \cup C_S \text{ with } C_A : A \rightarrow \Sigma_A \text{ is a color function such that } \Sigma_A = \{I, d, ar, c, cr\} \text{ that represents for } a \in A \text{ either the TP of the CWS that can produce it or the user input (I); and } C_S : S \rightarrow \Sigma_S \text{ is a color function such that } \Sigma_S = \{p, pr, d, ar, c, cr\} \text{ that represents the TP of } s \in S.\n\]

\[
\text{The firing of a transition of a WSDN corresponds to the selection of a WS (or CWS), which will participate in the composition of a CWS allowing to answer the user query } Q. \text{ In this article, we only focus on transactional composition of WSs satisfying the user’s functional and risk level requirements. Therefore, we define the marking of a WSDN, the fireable property of a transition, and the firing rules in such way we obtain, at the end, a TCWS, as defined in Def. 3.}\n\]

\[
\text{Def. 5. Marked WSDN.} \quad \text{A marked Web Service Dependency Net is a pair } (\text{WSDN}, M), \text{ where } M \text{ is a function which assigns tokens (values) to places such that } \forall a \in A, M(a) \subseteq \{\emptyset, \text{Bag}(\Sigma_A)\} \text{ where Bag corresponds to a set which can contain several occurrences of the same element. The marking of a WSDN represents the current state of the system, i.e., the set of attributes values produced by the system.}\n\]

\[
\text{Def. 6. Initial Marking (} M_0\text{).} \quad \text{The initial marking } M_0 \text{ depends on the user query } Q \text{ and is defined as:} \forall a \in (A \cap I_o), M_0(a) = \{I\} \text{ and } \forall a \in (A - I_o), M_0(a) = \emptyset.\n\]

\[
\text{Depending on the initial marking, some transitions can be fired. To determine if a transition } s \in S \text{ is fireable, we have to analyze (i) the color of its input places, (ii) its own color, and (iii) the color of the other attribute places. For each } x \in A \cup S \text{ of the WSDN, let } (x) = \{y \in A \cup S : F(y, x) = 1\} \text{ be the set of its predecessors and } (x^*) = \{y \in A \cup S : F(x, y) = 1\} \text{ be the set of its successors. We have the following definition:}\n\]
Def. 7. Fireable transition. A marking \( M \) enables a transition \( s \) iff all its input places contain a token \((\forall x \in (\ast, s), M(x) \neq 0)\) and at least one of the following conditions is verified:
1. \((\forall a \in A, M(a) \in I, \emptyset))\)
2. \((C_2(s) = cr)\)
3. \((C_2(s) = c) \land (\forall s \in (A^{-} s), M(a) \in [0, Bag([I, c, cr]))]\)
4. \((C_2(s) \in \{pr, ar\}) \land (\forall a \in (A^{-} s), M(a) \in [0, Bag([I, ar, cr]))] \land (\forall x \in (\ast, s), M(x) \in Bag([I, ar, cr]))]\)
5. \((C_2(s) \in \{\ast\}) \land (\forall a \in (A^{-} s), M(a) \in [0, Bag([I, cr]))] \land (\forall x \in (\ast, s), M(x) \in Bag([I, ar, cr]))]\)

These fireable conditions are deduced from the automaton of Figure 1 whose states model all the possible transactional properties of a TCWS and whose arc labels model the transactional properties of the component WSs. For example, Condition 3 of Def. 7 is deduced from the arcs leaving from state \( c \) of the automaton, where labels "/"*/", with \( t \in \{c, cr\} \), mean that a compensable WS can only be executed in parallel with compensatable or compensatable retrievable WSs, and labels "/", with \( t \in \sum_s \), mean that a compensable WS can be executed in sequential with any WS. In terms of WSDN: if the color of the transition \( s \) is \( c \) and if all the other places, which are not input places of \( s \), are empty or have tokens \( I, c \), or \( cr \), then \( s \) is fireable whatever the color of tokens in its input places are.

To know the transactional property (TP) of the resulting CWS (composed by all the WSs corresponding to the fired transitions), we defined the color associated with a marked WSDN by:

Def. 8. Color of a Marked WSDN. The color of a marked WSDN (WSDN, \( M \)) is: \( C_M = [I, \emptyset, ar, c, cr] \), \( C_M = I \), when no transition has been fired (i.e., when no WS has been selected and there is no resulting CWS). Otherwise, \( C_M \) represents the TP of the resulting CWS and is updated each time a transition is fired.

Due to a WS should be selected only one time to be a component of the resulting CWS, when a transition is fired, tokens are added to its output places and all the tokens are deleted from its input places (except from places that belong to \( O_Q \)), using the following rules:

Def. 9. Firing rules.\(^2\) The firing of a fireable transition \( s \) for a marking \( M \) defines a new marking \( M' \), denoted as \( M \rightarrow M' \), such that:
1. Tokens are added to the output places of \( s \) depending on the color of \( s \) and on the color of the tokens contained in the input places of \( s \), according to the following rules:
   - if \((\exists x \in (\ast, s), \exists d \in M(x))\), then \(e \in (\ast, s '), M'(y) \leftarrow M'(y) \cup \{d\}\)
   - else if \((\exists x \in (\ast, s), \exists d \notin M(x))\), then \(e \in (\ast, s '), M'(y) \leftarrow M'(y) \cup \{d\}\)
   - else if \((\exists x \in (\ast, s), \exists c \in M(x))\) and \((C_3(s) \in \{pr, ar, c, cr\})\), then \(e \in (\ast, s '), M'(y) \leftarrow M'(y) \cup \{c\}\)
   - else if \((\exists x \in (\ast, s), \exists c \notin M(x))\) and \((C_3(s) \in \{pr, ar, c, cr\})\), then \(e \in (\ast, s '), M'(y) \leftarrow M'(y) \cup \{c\}\)
   - else if \((C_3(s) \in \{pr, ar\})\) and \((C_3(s) \in \{ar, c, cr\})\), then \(e \in (\ast, s '), M'(y) \leftarrow M'(y) \cup \{c\}\)
2. Tokens are deleted from input places of \( s \), if they do not belong to \( O_Q \): \(e \in (\ast - O_Q), M(x) \leftarrow 0\).
3. Color \( C_M \) of the resulting (WSDN, \( M' \)) (see Def. 8) is updated, according to the following rules:
   - if \((C_M = [I, cr])\) and \(C_3(s) = pr\) then \(C_M \leftarrow \emptyset\)
   - else if \((C_M = [I, cr])\) and \(C_3(s) = pr\) then \(C_M \leftarrow \emptyset\)
   - else if \((C_M = [I, cr])\) and \(C_3(s) \in [\emptyset, ar, c, cr]\) then \(C_M \leftarrow C_3(s)\)
   - else \((C_M = [I, cr])\) and \(C_3(s) \in [\emptyset, ar, c, cr]\) then \(C_M \leftarrow C_3(s)\)

\(^2\)These rules are deduced from the transactional rules of CWS defined in [3] (see Fig. 1).

\(^3\)In this case, we are in the following situation: \(x \in (\ast, s), M(x) \in Bag([I, cr])\).
Def. 10. Firing sequence. A firing sequence \( \sigma = \{s_1, \ldots, s_n\} \) is a correct sequence of fired transitions starting from \( M_0 \) iff there are markings \( M_1, \ldots, M_n \) such that \( M_0 \xrightarrow{\sigma_1} M_1 \xrightarrow{\sigma_2} \cdots \xrightarrow{\sigma_n} M_n \); this is denoted as \( M_0 \xrightarrow{\sigma} M_n \).

Note that a firing sequence \( \sigma \) corresponds to a set of fired transitions, that in fact represents the selection of several WSs, \( s_1 \ldots s_n \), which are components of the resulting TCWS, whose transactional property is \( C_{M_0} \).

Def. 11. Reachable marking. A marking \( M \) is reachable if \( \exists \sigma \) such that: \( M_0 \xrightarrow{\sigma} M \).

Since the selection of two WSs which produce the same outputs is not useful, we have the following definition of a cut-off transition:

Def. 12. Cut-off transition. Let \( O_M \) be the set of the produced attributes by a firing sequence \( \sigma \) such that: \( O_M = \{a \in (\bigcup_i \sigma_i) \cap (\bigcup_i \sigma_i) \mid M_0 \xrightarrow{\sigma} M_n\} \). A transition \( s \) is said to be a cut-off transition iff its firing does not change the set of the already produced attributes, i.e., iff: \( M_0 \xrightarrow{\sigma} M_n \xrightarrow{\sigma} M_n+1 \) and \( O_{M_n} = O_{M_n+1} \).

When several transitions are fireable, to select which transition has to be fired, we propose a quality measure of a transition \( s \) which depends on the user query \( Q \) such that:

Def. 13. Quality associated with a transition. The quality of a transition \( s \), called \( \text{Quality}_Q(s) \), is defined as: \( \text{Quality}_Q(s) = g(C_{Q}(s)) \times \text{Max}(1, \text{card}(Q_0 \cap s^e) + 1) \times \text{card}(s^e) \), where \( g : \sum_{a} \rightarrow \mathbb{N} \), a function such that \( g(p) = g(\bar{a}) < g(pr) = g(\bar{a}r) < g(c) < g(cr) \).

Function \( g \) allows to select a transition whose transactional property is the less restrictive. An example of \( g \) is: \( g(p) = g(\bar{a}) = 1, g(pr) = g(\bar{a}r) = 2, g(c) = 3, \) and \( g(cr) = 4 \). \( \text{Max}(1, \text{card}(Q_0 \cap s^e) + 1) \) gives more chance to select transitions producing more required outputs. \( \text{card}(s^e) \) increases the quality to those transitions which will allow more transitions to be fireable.

Our problem consists in discovering and selecting the WSs of the registry whose composition satisfies the functional and the transactional requirements of the user, such that:

Def. 14. The WS Composition Problem-(WSC Problem): Given an user query \( Q \) and a WSDN, the WSC Problem consists in determining a Colored Petri Net \( WS\!DN_{\sigma_Q} \), sub-part of WSDN, created from the firing sequence \( \sigma_Q \), such that: \( M_0 \xrightarrow{\sigma_Q} M_n \) with \( M_0 \), the initial marking (see Def. 6) and \( M_n \), a reachable marking such that: \( \forall a \in (A \cap Q_0), M_F(a) \in [c, cr] \), if \( R_C = R_0 \) and \( \forall a \in (A \cap Q_0), M_F(a) \in [\bar{a}, \bar{a}r, c, cr] \), if \( R_C = R_1 \) and, such that the composition of all the WSs corresponding to the transitions of \( \sigma_Q \) represents a TCWS.

3. Our Approach

We propose an automatic WS selection algorithm to resolve the WSC Problem. The input of our algorithm is the user query \( Q \) and the WSDN, representing dependencies among the WSs of the registry and their input and output attributes. Its output is, if it exists, a Colored Petri Net (CPN), \( WS\!DN_{\sigma_Q} \), representing the set of the transactional WSs selected to participate in the composition. Our CPN-WS selection algorithm is composed by 4 steps. Step 1 verifies the validity of \( Q \). Step 2 identifies the WSs of the registry (i.e., the transitions of the WSDN) that may be useful to produce the outputs of \( Q \) by considering the transactional properties that satisfy the risk level. Step 3 returns a firing sequence \( \sigma_Q \) containing the component WSs of a TCWS required to evaluate \( Q \). Step 4 returns a Colored Petri Net, \( WS\!DN_{\sigma_Q} \), built from \( \sigma_Q \) and allowing to execute the resulting TCWS.

We also propose an algorithm to create the WSDN from the registry. The following sections describe the WSDN creation algorithm and the 4 steps of the CPN-WS algorithm. Note that, due to the space limitation, proofs of the proposed algorithms are not included in the article.
3.1. Creation of WSDN, the Colored Petri Net representing the WS registry

Let $s$ be a WS of the registry, whose input attribute set is $I_s$ and output attribute set is $O_s$. The creation of WSDN is done by adding one service at a time, using Algo. 1.

Algorithm 1: Adding a WS $s$ into a WSDN

**Input:** WSDN, a Web Service Dependency Net (eventually empty) and $s$, a WS of the registry to be added in WSDN

**Output:** WSDN after adding $s$

1. Web service $s$ is mapped to a transition and is added to $S$.
2. All input attributes of $s$ are mapped into places and are added to set $A$ if there does not already exist a place with the same semantic or if it exists with at least one successor:
   
   \[ \forall i \in I_s, \exists \beta \in A \mid (\text{semantic}(i) = \text{semantic}(\beta)) \lor (\exists \alpha \in A \mid (\text{semantic}(i) = \text{semantic}(\alpha)) \land (\alpha \neq \beta)), \text{create a corresponding place } i \in A \text{ and } F(i, s) \rightarrow 1, \]

3. Every output attributes of $s$ are mapped into places and are added to $A$, if there does not already exist a place with the same semantic:
   
   \[ \forall o \in O_s, \exists \beta \in A \mid (\text{semantic}(o) = \text{semantic}(\beta)), \text{create a corresponding place } o \in A \text{ and } F(s, o) \rightarrow 1, \]

4. An arc to $s$ is created from any place having no successor and representing an input attribute of $s$:
   
   \[ \forall \alpha \in (I_s \setminus A) : \text{if } \alpha = 0, \text{then } F(\alpha, s) \leftarrow 1, \]

5. An arc is created from $s$ to each place $o$ of WSDN having the same semantic as an output place of $s$:
   
   \[ \forall \alpha \in A : \text{if } \exists \beta \in o^* (\text{semantic}(\beta) = \text{semantic}(o)), \text{then } F(s, o) \leftarrow 1, \]

6. For all transitions $t$ of WSDN having an output place $o$ with the same semantic as an input place $i$ of $s$, add an arc $(t, i)$:
   
   \[ \forall t \in S : \text{if } (\exists \beta \in o^* (\text{semantic}(\beta) = \text{semantic}(o)) \lor (\exists \alpha \in i^* (\text{semantic}(\alpha) = \text{semantic}(i))), \text{then } F(t, i) \leftarrow 1. \]

Note that, using this algorithm no exclusive transition exists because we duplicate input places (see Item 2 of Algo. 1), then each place of a WSDN has zero or one successor. This is because we want to model the fact that the execution of a WS $s$ cannot forbid the execution of another WS $s'$ which would have the same input attributes ($s$ and $s'$ are independent even if they have the same input attributes). However, the execution of a WS $s$ can allow the execution of other ones when $s$ produces its inputs attributes (see Item 5 of Algo. 1).

We illustrate the execution of this step with the following example. Let us suppose that the registry contains the WSs described in Table 1 and each WS has been added in the WSDN from $s_1$ to $s_9$. First $s_1$ is added to the WSDN, Item 1 of Algo. 1 is executed, thus all places of transition $s_1$ are created (Items 2 to 3 of Algo. 1 are executed). Then $s_2$ is added, its input places are created (Item 2 of Algo. 1), but because a place $PubCod$ already exists (successor of $s_1$), then an arc is created from $s_2$ to this place (Item 5 of Algo. 1). When $s_3$ is added, an arc is added from place $PubCod$ (successor of $s_2$ and $s_2$) because this place has no successor (Item 4 of Algo. 1), and the output place $Title$ is created. On the other hand, when $s_4$ is added, because it already exists a place $PubCod$ (successor of $s_3$ and $s_2$) with a predecessor ($s_3$), then a new place $PubCod$ is created as input of $s_4$ and as output of $s_2$ and $s_2$ (Items 1 and 6 of Algo. 1). Moreover, an output place $ConfCod$ is created as output of $s_4$ (Item 3 of Algo. 1). Figure 2.(a) represents the WSDN created from the WSs of Table 1.

<table>
<thead>
<tr>
<th>WS Name</th>
<th>Input</th>
<th>Output</th>
<th>TP</th>
<th>WS Name</th>
<th>Input</th>
<th>Output</th>
<th>TP</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_1$</td>
<td>AuthorCod,Inst</td>
<td>PubCod</td>
<td>$p$</td>
<td>$s_2$</td>
<td>AuthorName</td>
<td>PubCod</td>
<td>$p$</td>
</tr>
<tr>
<td>$s_3$</td>
<td>PubCod</td>
<td>Title</td>
<td>$p$</td>
<td>$s_4$</td>
<td>PubCod</td>
<td>ConfCod</td>
<td>$p$</td>
</tr>
<tr>
<td>$s_5$</td>
<td>PubCod</td>
<td>ContCod,ContName</td>
<td>$p$</td>
<td>$s_6$</td>
<td>ConfCod</td>
<td>ContName,ContDate</td>
<td>$r$</td>
</tr>
<tr>
<td>$s_7$</td>
<td>Inst</td>
<td>AuthorCod</td>
<td>$p$</td>
<td>$s_8$</td>
<td>ContCod</td>
<td>ContCod</td>
<td>$p$</td>
</tr>
<tr>
<td>$s_9$</td>
<td>AuthorCod</td>
<td>ConfCod</td>
<td>$pr$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Example of WSs of the registry (from [8])

While the creation of the WSDN is independent from $Q$, all steps of the CPN-WS selection algorithm are executed for each $Q$. 

6
3.2. Validation of the user query $Q$

The validation of $Q$ consists in verifying that the registry (represented by a WSDN) contains (i) at least one WS whose input attributes are included in set $I_Q$ given by the user (i.e., a WS can be executed from attributes of $I_Q$), and (ii) one or more WSs allowing to produce all output attributes of $O_Q$ needed by the user. In such case, the selection algorithm can proceed to the next step. Otherwise, it means that it is impossible to find a solution to $Q$.

3.3. Identification of the potentially useful WSs

If $Q$ is valid, then WSs of the registry potentially useful to answer $Q$ can be identify. The identification step, presented in Algo. 2, analyzes the WSDN from the output places corresponding to the output attributes in $O_Q$ and creates a new Colored Petri Net, $WSDN_Q$, by recursively adding all the predecessors of the analyzed places or transitions. This step takes into account the transactional properties in order to select useful WSs that also satisfy the user risk level. If it does not exist at least one input attribute $\in I_Q$ among the input places of $WSDN_Q$ (i.e., the places with no predecessor), then the algorithm returns an error, because $Q$ has no solution. Otherwise, the selection algorithm can proceed to the next step. Algo. 2 is inspired from the yellow coloring step of SAM [7] algorithm.

We illustrate this identification step with the following example: let suppose $I_Q = \{Inst\}$, $O_Q = \{ConfName, ConfDate\}$, $R_Q \subseteq \{R_0, R_1\}$, that the WSDN of the registry is represented by Figure 2.(a), and that all WSs are either compensatable or compensatable retriable. In this scenario, the result of Algo. 2 is the Colored Petri Net $WSDN_Q$ represented in Figure 2.(b). It contains predecessors of $ConfName$ and $ConfDate$ places and recursively their predecessors. Transitions $s_3$ and $s_6$ of WSDN do not appear in $WSDN_Q$ because they are not predecessors of $ConfName$ and $ConfDate$. When the transactional properties of the WSs are those represented in the TP column of Table 1 and $R_Q = R_1$, then only transitions $s_7, s_9, s_4, s_5$ and $s_6$ are selected by Algo. 2.

Algo. 2 returns a CPN $MSDN_Q = (A_Q, S_Q, F_Q, E_Q)$ such that $S_Q = \{s \in S_{R_0} \mid |O_Q \subseteq (\cup_{\sigma \in \sigma_c(s, c) \in c}) \land (\cup_{\sigma \in \sigma_c(s, c) \in c}) \cap I_Q \neq \emptyset \}$ with $S_{R_0}$ defined in Line 1 of Algo. 2. Moreover, $MSDN_Q$ allows to determine all the firing sequences $\sigma$ such that $M \xrightarrow{\sigma} M'$, with $M$ an initial marking such that $(\exists a \in I_Q \mid M(a) = |I|)$ and, $M'$ a final marking such that $(\exists a \in O_Q \mid M(a) = 0)$, and such that the composition of all the WSs corresponding to the transitions of $\sigma$ represents a TCWS.
3.4. Automatic selection of the component WSs

The automatic selection step is presented in Algo. 3. The inputs of this step are the $WSDN_Q$ returned by Algo. 2, the initial and the final markings ($M_0$ and $M_F$), and $O$. The output is, if it exists, a firing sequence $\sigma_Q$ (see Def. 10), corresponding to the TCWS whose components are the WSs of the registry required to evaluate $Q$ and satisfying all the user requirements.

Let suppose that $I_Q = \{\text{Inst}\}$ and $O_Q = \{\text{Conf/Name}, \text{Conf/Date}\}$. Depending on the color of the transitions, the following firing sequences can be returned by Algo. 3: $\{7, 8, 9, 10\}$; $\{57, 58, 59, 61\}$; $\{57, 59, 61\}$; $\{57, 58, 59, 61\}$; $\{57, 59, 61\}$. Note that if $\Sigma_4$
Algorithm 3: Automatic WSs selection

Input: WS $\mathcal{D}(Q) = (\mathcal{A}, \mathcal{S}, \mathcal{F}, \mathcal{G})$ returned by Algorithm 2, $M_0$, the initial marking – see Def. 6, $M_f$, the final marking – see Def. 14, and the user query $Q = (I_Q, O_Q, R_Q)$ – see Def. 1.

Output: $\sigma_Q$: a firing sequence that satisfies $Q$ (i.e., $M_0 \xrightarrow{\sigma_Q} M_f$)

begin
1. $Current \_ \_M \leftarrow M_0$ /* The current marking is the initial one */
2. $New \_M \leftarrow NULL$ /* The new marking obtained after firing a transition */
3. $\sigma_Q \leftarrow \emptyset$ /* The resulting firing sequence is empty at the beginning */
4. $CM \leftarrow 1$ /* The color of the marked Colored Petri Net */
5. $Fireable \leftarrow \{s \in S_Q : s$ is fireable $\}$ see Def. 7 */

repeat
% Select the best fireable transition according to Quality $\sigma_Q$ */
6. Select a fireable transition s. [\forall' \in Fireable, (Quality_Q(s) \geq Quality_Q(s'))] ;
7. if $\neg$ isCutOff(s)/* See Def. 9 */ then
8. Fired $s$ applying firing rules/* See Def. 9 */
9. $O_M \leftarrow O_M \cup \{s'\}$/* Update $O_M$ with successors of $s$ */
10. $New \_M \leftarrow M'$ /* such that $Current \_M \rightarrow M'$ */
11. $Current \_M \leftarrow New \_M$ /* Update current marking with the new marking */
12. $Succ \leftarrow \{s'\}$/* Compute the set of transitions successor of $s$ */
13. $Parallel \leftarrow Fireable \neg Succ \leftarrow \{s\}$/* Set of transitions corresponding to WSs to be executed in parallel with $s$ */
% Add successors of $s$ to Fireable if they verify the fireable conditions - see Def. 7 */
14. repeat
15. Select the first element $w$ of $Succ$;
16. if $(CM \in \text{State and } \text{in } \text{Fig. 1} \}$ then
17. if $w$ is fireable then Fireable $\leftarrow$ Fireable $\cup \{w\}$;
18. until $\text{Fireable } \emptyset$ ;/* Disable fireable transitions which do not verify the fireable conditions anymore */
19. repeat
20. Select the first element $w$ of $Parallel$ ;/* For the following conditions, see arcs labeled by $'\exists' \text{ from State I and } \text{in } \text{Fig. 1} \}$
21. if $(CM \in \text{State } \neg \text{in } \text{Fig. 1} \}$ then
22. if $w$ is fireable then Fireable $\leftarrow$ Fireable $\neg \{w\}$;
23. until $\text{Parallel } \emptyset$ ;
24. if $\text{Fireable } \emptyset$ or the desired marking $M_f$ is not reached (i.e., $M_0 \xrightarrow{\sigma_Q} Current \_M$ with $Current \_M \neq M_f$) then
25. Return $\sigma_Q$; else Return ERROR;
end

is fired, then $s_0$ is cut-off and if $s_0$ is fired, then $s_4$ is cut-off (because WSs $s_4$ and $s_9$ produce the same output attributes). If $s_5$ is fired then $s_4$ and $s_9$ are cut-off (because $(s_4)^* \subset (s_9)^*$ and $(s_9)^* \subset (s_4)^*$). If $s_4$ and $s_9$ are fired or if $s_9$ and $s_6$ are fired then $s_5$ is cut-off (because $(s_5)^* \subset [(s_4)^* \cup (s_9)^*]])$ with $i \in \{4, 9\}$.

Note that if a firing sequence $\sigma_Q$ exists, such that $M_0 \xrightarrow{\sigma_Q} M_f$, and $\sigma_Q$ corresponds to a TCWS satisfying $Q$, then Algo. 3 finds it. On the other hand, if an error is returned, then no firing sequence $\sigma_Q$, satisfying $Q$, exists. Moreover, if a firing sequence $\sigma_Q$ is returned by Algo. 3, then it corresponds to a TCWS satisfying $Q$.

3.5. Creation of the resulting Colored Petri Net

Let suppose that Algo. 3 returns the firing sequence $\sigma_Q = \{s_5, s_1, s_4, s_5, s_6\}$. This firing sequence contains a useless transition, $s_4$. Indeed, $[(s_4)^* \subset (s_5)^*]$ and $M_0 \rightarrow M_f$, with $\zeta = x \rightarrow [\neg s_4]$, but because $s_4$ has been selected before $s_5$, therefore it has not been considered as cut-off. As a consequence, we add another step to our algorithm, in order to eliminate potentially useless transitions of the resulting firing sequence. Moreover, if the Algo. 3 produces the firing sequence
$\sigma_Q = \{s_7, s_1, s_9, s_6\}$, this last step will eliminate transition $s_1$ because in $\sigma_Q$ there is no sequence of transitions starting from $s_1$ and leading to the output places of $O_Q$.

In this sense, the last step of CPN-WS algorithm consists in cleaning $\sigma_Q$, result of Algo. 3, by deleting useless transitions, producing a firing sequence $\sigma'_Q$ such that $M_0 \rightarrow^0 M_F$ and such that the composition of all the WSs corresponding to the transitions of $\sigma'_Q$ represents a TCWS.

The algorithm of our last step is similar to the five first lines of Algo. 2, except that $S_{R_0}$ is replaced by $\sigma_Q$ in the if instructions and except that all transitions $a$, such that it exists another transition $s$ into $\sigma_Q$ verifying $((a^*) \cap A_{s\nu}) \subseteq ((s^*) \cap A_{s\nu})$, are not added into Pred nor into WS DN $\sigma_Q$.

4. Conclusion

In this work, we propose a transactional-driven WSs selection and composition algorithm where functional conditions, expressed as input and output attributes, and composite properties, expressed as a risk level, are considered at the same time to compute adequate transactional WS compositions. We use Colored Petri Net as a formalism to represent composite WSs, adapt a Petri Net unfolding algorithm and perform a Best-First search, guided by the aggregated transactional properties, which stops when the desired marking corresponds to a composition of WSs that satisfies both functional and risk level user requirements. Our future work will focus on QoS-aware selection and on failure recovery of TCWS execution.

5. References