On context-independent and context-aware cloud services substitutability verification

Sofiane Bourouz* and Nadia Zeghib

LIRE Laboratory,
Constantine 2 – Abdelhamid Mehri University,
Constantine, Algeria
Email: sofiane.bourouz@univ-constantine2.dz
Email: nadia.zeghib@univ-constantine2.dz
*Corresponding author

Abstract: The composition of web services in cloud computing allows making them cooperative in order to satisfy a client request. However, the substitution of one of these services by another must ensure the proper functioning of the new composition. Hence there is a need to develop adequate methods to verify the services substitution. In this aim, we propose, in the present paper, two verification methods. The first one deals with context-independent substitution and sets if a web service can substitute another one. The second one focuses on context-aware substitution and decides the substitutability of web services in a specific composition. These methods use open coloured Petri net (OCNets) as a formal framework for modelling web services and their composition in cloud computing. This model allows the structural analysis of web services’ interfaces. For the behavioural verification, we use services automata, which allow capturing OCNets’ behaviour and checking its preservation.

Keywords: web service; web services substitutability; open coloured Petri nets; OCNets; context-awareness; context-aware; structural verification; behavioural verification; web services composition; composability; cloud services.


Biographical notes: Sofiane Bourouz is currently a PhD student at the Constantine 2 – Abdelhamid Mehri University, Algeria. He received his Master’s in Computer Science in 2010 on multi-agent system for production management. His research interests include formal methods, parallel and distributed systems, web services, programs substitution, service-oriented computing, and Petri nets analysis. He is also interested in programming languages (Java) and systems modelling.

Nadia Zeghib is a Professor at the Constantine 2 – Abdelhamid Mehri University, Algeria. She received her PhD in Computer Science in 2007 on formal specification languages and extended concurrent algebraic high level Petri nets. She is a member of the LIRE laboratory and several projects in the area of formal methods for the development and verification in software engineering. Her current research interests include formal methods, service-oriented computing, mathematical logics, design and verification of complex systems.
1 Introduction

Nowadays, cloud computing becomes one of the most promising technologies by providing online services or software applications, accessible at anytime, anywhere, and from any platform (Fei and Fengjie, 2015). Indeed, cloud computing which is based on service-oriented architecture (SOA), allows overcoming drawbacks of components oriented technologies thanks to its independence from platforms, and its capacity for integration and reuse. Cloud computing contains several categories of services. Software-as-a-service (SaaS) is one of these categories which offer software applications and web services over the internet to meet the needs of its customers (Ching-She and Ibrahim, 2012).

Web services are applications made available on the Internet in order to be published, located, and invoked remotely by customers. They have the characteristic of being self-descriptive and modular, providing a simple programming and applications’ deployment model. The functionalities offered by web services are designed to be jointly used in compositions (Yu et al., 2005).

The composition of web services (Alonso et al., 2004; Benatallah et al., 2005; Claro et al., 2006; Hameurlain, 2013; Srivastava and Koehler, 2003; Yang and Papazoglou, 2003) refers to the proper assembly of already existing services, producing a new well-tailored service. In fact, the composition of web services allows combining their functionalities in order to provide new ones richer and able to satisfy customers’ complex requirements (Alonso et al., 2004). The composed web services can be located in the same cloud, in different clouds, or in a non-cloud, which offers multiple possible combinations to meet the needs of the cloud client.

However, a web services’ composition may become inaccessible or ineffective if one of its web services breaks down, turns out to be unavailable, or requires to be updated. In such cases, a substitution of one or more services of this composition is needed.

The substitution of web services is one of the most important issues in the research area of distributed systems (Bordeaux et al., 2004; Yin and Deng, 2013). It must ensure the preservation of all the functionalities and the initial behaviour of the involved service in this substitution. The verification of this substitution will focus on services to substitute; and this is what makes this verification context-independent (Jin et al., 2012). Substituting service in a composition will modify its structure and behaviour, which could affect its proper functioning. So, the validation of substitution depends on compatibility between substitute service and this composition, hence the notion of context aware (Dey et al., 2001; Pathak et al., 2007; Zhang et al., 2011).

In this paper, we focus on services substitutability verification. On one hand, we propose open coloured Petri nets (OCNets) as a formal framework for modelling and specifying web services located in a cloud, as well as their substitution and their composition. On the other hand, two verification methods, respectively based on
context-independent and context-aware, are proposed, using appropriate algorithms to
determine the validity, the coherence and the correction of such substitution.

The paper is organised as follows. Section 2 explains the principles of substitution
related to the notion of context. Section 3 presents the OC Nets model. Section 4 describes
the verification approach of substituting web services independently of their context of
use. Section 5 presents the process of context aware web services substitutability
verification. Section 6 examines the existing approaches related to the substitution of web
services. Finally, the last section concludes the paper and draws up future work.

2 Web services substitution

In the area of web services, the substitution is considered as one of research challenges
due to its difficulty to be validated. It consists in replacing a web service $S_1$ located in a
composition $C_1$ by another service $S_2$ that will transform the composition $C_1$ to $C_2$, while
ensuring that the new service $S_2$ provides at least the same services initially offered by $S_1$.
Therefore, substituting web services must meet functional requirements that are related to
the services offered to the composition and to the reason of this substitution.

Indeed, the substitution of a web service by another one for an update requires adding
new functions, whereas, ensuring that the composition works again must guarantee that
the substitute service $S_2$ provides the same services as $S_1$.

However, the proper functioning of the new composition $C_2$ is related to the
compatibility between the new service $S_2$ and the environment of initial service
($C_1 - \{S_1\}$). This environment represents the set of web services that are composed
with $S_1$.

Consequently, the verification of web services substitutability aims to guarantee the
proper functioning of the composition by analysing and comparing the services to
substitute, and also to consider the context-independent and context-aware issues.

2.1 Context-independent substitution

This verification method aims to validate the substitution of two web services $S_1$ and $S_2$
(substituted and substitute) in order to compare and to verify the preservation of desired
properties, but without taking into consideration the environment in which the initial
service $S_1$ runs.

In fact, the verification on context-independent is confined to compare web services
$S_1$ and $S_2$ to substitute without considering the composition in which was $S_1$ (Jin et al.,
2012). Therefore, it is limited to the analysis and the comparison of both structures and
behaviours of $S_1$ and $S_2$ without checking the compatibility between the new service $S_2$
and the environment $C_1 - \{S_1\}$.

2.2 Context-aware substitution

Context-aware substitution is related to the fact that the environmental aspect is important
in the substitutability verification process. This involves verifying the compatibility
between the web service $S_2$ and the environment of $S_1$, as well as comparing $S_1$ and $S_2$. In
other words, the web services substitution may be possible in a context but not in another one, which makes the decision of substituting depends on the context.

Figure 1  Web services composition and substitution in the cloud computing

Figure 1 represents a user’s requests of an itinerary from its position to a destination using an application in a cloud. This application uses a set of composite web services located in different clouds. The web service Geo_Pos provides the longitudes and latitudes of both the user’s position and his destination, to the web service GPS-1. The service MAPS retrieves data to draw the different paths on the map. Through the latter, the service GPS-1 receives traffic information from an external web service of Cloud 2. And finally, GPS-1 transmits the itinerary’s information to the user.

We suppose that the web service GPS-1 becomes unavailable, and its substitution by a new service GPS-2 is required. This substitution involves a transformation of the initial composition by ensuring its proper functioning and the preservation of the services initially offered by GPS-1.

The context-independent verification method is applied between the web services GPS-1 and GPS-2. This method affirms if the web service GPS-1 can be substituted by GPS-2 in all contexts. However, the context-dependent method includes all the web services involved in the composition as a composite service, by verifying its composability with the new service GPS-2 and the behaviour of the new composition.

3 OCNet: a formal framework for web services modelling introduction

High-level Petri nets constitute an adequate formalism for modelling and specifying complex, discrete and distributed systems, thanks to their ability to analyse the competition and the behaviour of these systems. The composition and the communication
between web services are performed using their interfaces (Papazoglou, 2004; Papazoglou et al., 2007). This makes the verification of their substitution related to the comparison of these interfaces and the capture of their different aspects.

However, modelling interfaces is not supported by the ordinary high level Petri nets. To overcome this drawback we propose a formal model named OCNet model. An OCNet consists of a coloured Petri net representing the internal part of this OCNet (we will call INNER) which has an interface as an external part, allowing the modelling and analysis of the composition and the exchange of messages between web services (Jin et al., 2012; Stahl and Wolf, 2009; Stahl et al., 2009; Stahl and Van der Aalst, 2014).

The OCNet model allows not only facilitating the verification of web services substitutability, but also the capture of all aspects relating to web services using its graphical representation.

3.1 Definition
An OCNet is a tuple \( N = (P, P_{IO}, T, \Sigma, C, W^-, W^+, m_0, \Omega) \) where:

- \( P \): is a finite and non-empty set of places.
- \( P_{IO} \): is a set of interface places (\( P_{IO} \subset P \)).
- \( T \): is a finite and non-empty set of transitions s.t. \( P \cap T = \emptyset \).
- \( \Sigma \): is a finite and non-empty set of colours.
- \( C : P \cup T \rightarrow \Sigma \) is the colour domain application.
- \( W^- : \) is the backward incidence matrix \( P \times T \).
- \( W^+ : \) is the forward incidence matrix \( T \times P \).
- \( m_0 : \) is the initial marking.
- \( \Omega : \) is a set of final markings of the OCNet.

In the above definition, \( P_{IO} \) represents the interface of the OCNet, while all the remaining elements constitute a coloured Petri net (INNER), which is the inner part.

In the INNER, for each element \( x \in P \cup T \), \( \cdot x \) is the Preset of \( x \), and \( x' \) is the Postset of \( x \), defined as follows:

- \( p \in P \), \( p = \{ t \in T / (t, p) \in W^- \} \)
- \( t \in T \), \( t = \{ p \in P / (p, t) \in W^+ \} \)

A transition \( t \in T \) is enabled for a marking \( m \) and a colour \( c \in C(t) \) iff \( \forall p \in P, m(p) \geq W^-(p, t)(c) \).

The firing of \( t \) for a marking \( m \) and a colour \( c \in C(t) \) gives a new marking \( m' \) defined by:

\[
\forall p \in P, m'(p) = m(p) - W^-(p, t)(c) + W^+(t, p)(c).
\]
3.2 Interface place

Interface places represent the ports allowing the OCNets to be composed with each others. They are channels through which exchanged messages pass (input and output) between OCNets. An interface place is defined by its name and the message that will pass through this place. This definition is given below:

\[ \forall p^i \in P_{IO} \cdot p^i = \{ \text{Name}^i, \text{Msg}^i \} \]

- \(1 \leq i \leq n\) and \(n\) is the number of interface places
- \(\text{Name}^i\): name of the interface place \(p^i\)
- \(\text{Msg}^i\): the message passing through the interface place \(p^i\) and which contains a set of fields. This message is defined as follows:
  - \(\text{Msg}^i = \{ \text{Field}^i_x / 1 \leq x \leq k \}\)
  - \(k\) is the number of fields in the message \(\text{Msg}^i\)
  - \(\text{Field}^i_x\): the field \(x\) of the message \(\text{Msg}^i\).

In a message \(\text{Msg}^i\), each field is represented by its name, and the type of information supported as follows:

- \(\text{Field}^i_x = \{ \text{NameField}^i_x, \text{Type}^i_x \}\)
  - \(\text{NameField}^i_x\): name of the field \(x\)
  - \(\text{Type}^i_x\): type of information in the field \(x\).

3.3 OCNet’s interface

The interface \(P_{IO}\) of an OCNNet \(N\) consists of sets of interface places: \(P_{IN}\) for input places and \(P_{OUT}\) for output places. It is used to represent the set of the provided and the required services of web service \(S\). This interface is mainly used to allow the structural analysis and comparison of web services, as well as verification of substitutability and composability of services. It is defined as follows:

Let us consider \(P_{IO}\) the interface of the OCNNet \(N\). The set \(P_{IO} = P_{IN} \cup P_{OUT}\) denotes the interface of the OCNNet \(N\) where:

- \(P_{IN}\): is a set of input places s.t. \(P_{IN} \subseteq P_{IO} \wedge \forall p_{in} \in P_{IN}: p_{in} = \emptyset\).
- \(P_{OUT}\): is a set of output places s.t. \(P_{OUT} \subseteq P_{IO} \wedge \forall p_{out} \in P_{OUT}: p_{out} = \emptyset\).

This definition concerns generally an atomic web service (e.g., substitute). However, the composition of this last with other web services allows to refine the definition by designating the status of its interface places in this composition: taken (denoted by \(P^t\)) or free (denoted by \(P^f\)).

The aim of this refinement is to represent the interface places that will become internal in the composition, to focus the verification of substitutability and composability on these taken places. Therefore, a new definition of a composed OCNNet’s interface is proposed:

\[ P_{IO} = P_{IN} \cup P_{OUT} \]
The use of OCNet model for web services modeling is motivated by their functional and behavioral similarities. In fact, the main idea can be summarized as follows:

- The web service communication and composition is represented by the set of interface places of the OCNet.
- The exchanged messages (sent and/or received) between web services are represented by colored tokens in the OCNet. Each message value corresponds to a color and message types are specified by color sets.
- The OCNet transitions correspond to the web service activities.
- The control flow relations between activities specified by web service are captured with OCNets token firing rules and the arc inscriptions and transition guard expressions.

We further require that neither the initial nor the final marking mark any interface place \( p \in P_{in} \) and by this way, each web service model is naturally represented by an OCNet.
Figure 2 shows the OCNet $N_1$ model of a web service $GPS-1$ of Figure 1. The internal places (resp. interface places) are represented by white circles (resp. solide circles), while the internal transitions (resp. interfaces transitions) are represented by white rectangles (resp. solide rectangles).

4 Context-independent web services substitutability checking

The verification of context-independent web service substitution consists in comparing two web services to substitute ($S_1$ by $S_2$) without taking into consideration the compatibility between the new service and the environment of the old one.

Figure 3 shows the main steps of the proposed verification method. This verification carries out a comparison based on both structural and behavioural aspects. To show how to proceed in this verification, we apply the different steps illustrated in Figure 3 on the OCNet models of web services to substitute.

This approach has two comparison steps based successively on the structural and the behavioural aspects. The decision of the substitution depends to the validation of the two steps of comparison.

**Figure 3** Main steps of context-independent substitutability checking

![Diagram of structural and behavioural verification]

4.1 Structural verification

This structural verification step aims to compare the interfaces of $S_1$ and $S_2$ based on their OCNets models $N_1$ and $N_2$. It is necessary to determine the type of the relation between their interfaces in order to ensure if they meet the requirements related to the reason for the substitution.

According to the definition of the OCNet’s interface, this last includes a set of interface places. Before presenting interfaces’ comparison process, we must first define the method of comparing two interface places with the same direction (input or output).
4.1.1 Interface places comparison

The comparison of interface places is based on comparing the different elements defining these places. The validation of this comparison must meet several conditions: a place of the substitute service must have the same name and pass at least the same message that the place of the substituted service. We have to know that the semantic aspect of names is not considered in this comparison.

Let us consider $p_i, p_j$, respective interface places of OCNets $N_1$ and $N_2$ where:

\[
\begin{align*}
p_i &= \{\text{Name}_i, \text{Msg}_i\} \land \text{Msg}_i = \{\text{Field}_{i,x} \mid x \in [1..k]\} \land \forall \text{Field}_{i,x} \in \text{Msg}_i : \\
\text{Field}_{i,x} &= \{\text{NameF}_{i,x}, \text{Type}_{i,x}\} \\
p_j &= \{\text{Name}_j, \text{Msg}_j\} \land \text{Msg}_j = \{\text{Field}_{j,y} \mid y \in [1..l]\} \land \forall \text{Field}_{j,y} \in \text{Msg}_j : \\
\text{Field}_{j,y} &= \{\text{NameF}_{j,y}, \text{Type}_{j,y}\}
\end{align*}
\]

- $p_i$ and $p_j$ must have the same direction: $(p_i \in P_{IN1} \land p_i \in P_{IN2}) \lor (p_j \in P_{OUT1} \land p_j \in P_{OUT2})$
- $p_i$ and $p_j$ must have the same name: $\text{Name}_i = \text{Name}_j$
- $p_i$ must have at least the same message as $p_j$: $\forall \text{Field}_{i,x} \in \text{Msg}_i : \exists \text{Field}_{j,y} \in \text{Msg}_j : \text{Msg}_i \subseteq \text{Msg}_j$ where:
  - $\text{NameF}_{i,x} = \text{NameF}_{j,y}$
  - $\text{Type}_{i,x}$ is a subtype of $\text{Type}_{j,y}$ if $p_i$ and $p_j$ are the input places. This condition is justified by the inability of the place $p_j$ to support a type that it cannot decipher, because it will provoke a deadlock.
  - $\text{Type}_{i,y}$ is a subtype of $\text{Type}_{j,x}$ if $p_i$ and $p_j$ are the output places. This requirement has no impact on the service itself, since it is a sending to the outside of the service. However, its non-satisfaction affects the reception place of the composition, and will cause a deadlock of this one.

Let us consider $P_{IN1} = \{p_1, p_2, p_3\}$ and $P_{IN2} = \{p_1', p_2', p_3'\}$ the sets of the input places of the respective services $S_1$ and $S_2$ are defined by:

\[
\begin{align*}
p_1 &= \{\text{Itinerary}, ([a: \text{float}], [b: \text{float}])\} \\
p_1' &= \{\text{Route}, ([a: \text{float}]), [b: \text{float}])\} \\
p_2 &= \{\text{Transport}_\text{Mode}, ([a: \text{string}], [b: \text{string}])\} \\
p_2' &= \{\text{Transport}_\text{Mode}, ([a: \text{string}])\} \\
p_3 &= \{\text{Cost}, ([a: \text{float}])\} \\
p_3' &= \{\text{Cost}, ([a: \text{int}])\}
\end{align*}
\]

The interfaces of $S_1$ and $S_2$ do not verify the structural conditions at least for one of the following reasons:

- The interface places $p_1$ and $p_1'$ have not the same name.
- The message of $p_2'$ has not the field ‘b’.
The field ‘a’ of $p^3$ is not the subtype of the field ‘a’ of $p^{-3}$.

### 4.1.2 OCNets’ interfaces comparison

Substituting the web service $S_1$ must ensure that all services it initially offers to its composition are preserved in the substitute service $S_2$. In other words, OCNet $N_2$ interface must at least have the taken interface places used by $N_1$ in the composition. This condition can be defined as follows:

- $\forall p_{\text{in}} \in P_{\text{IN}1} \exists p_{\text{in}} \in P_{\text{IN}2}: p_{\text{in}} = p_{\text{in}} \wedge P_{\text{IN}1} \subseteq P_{\text{IN}2}$
- $\forall p_{\text{out}} \in P_{\text{OUT}1} \exists p_{\text{out}} \in P_{\text{OUT}2}: p_{\text{out}} = p_{\text{out}} \wedge P_{\text{OUT}1} \subseteq P_{\text{OUT}2}$

The free interface places of $N_1$ determine the type of the relation between interfaces of $N_1$ and $N_2$, and this will be used to check whether the objective of this substitution would be reached (troubleshooting, adding services, update, etc.).

Comparing the sets of interface places leads to four possible cases (Belguidoum and Dagnat, 2007). The relation between the sets of input (resp. output) places is represented by *Interface Relation*, where (*) is the quality of this relation, and takes one of the following values: very weak, weak, strong, and very strong, according to the cases described in Figure 3.

#### Figure 4 Possible relations between two input interfaces, (a) case 1 (b) case 2 (c) case 3 (d) case 4

**Theorem:** Given two web services $S_1$ and $S_2$ modelled respectively by OCNets $N_1 = (P_1, P_{\text{IN}1}, T_1, \Sigma_1, C_1, W_1, W_1^+, m_{\text{in}}, \Omega_1)$ and $N_2 = (P_2, P_{\text{IN}2}, T_2, \Sigma_2, C_2, W_2, W_2^+, m_{\text{in}}, \Omega_2)$ where: $P_{\text{IN}1} = (P'_{\text{IN}1} \cup P'_{\text{IN}2}) \cup (P'_{\text{OUT}1} \cup P'_{\text{OUT}2})$ and $P_{\text{IN}2} = (P'_{\text{IN}2} \cup P'_{\text{IN}2}) \cup (P'_{\text{OUT}2} \cup P'_{\text{OUT}2})$; the structural verification of the substitution of $S_1$ by $S_2$ is true iff: $(P'_{\text{IN}1} = P'_{\text{IN}2}) \wedge (P'_{\text{OUT}1} = P'_{\text{OUT}2})$.

**Proof:** The composition of $S_1$ with a composite $C$ is achieved through their corresponding interface places. In fact only the ports $P_{\text{IN}1}$ and $P'_{\text{OUT}1}$ of $S_1$ communicate with their corresponding Output/OutputC ports of the composite web service $C$. The composition of $S_1$ and $C$ produces a new composite $C_1$. To ensure a correct structural composition of $S_2$ and $C$, the web service $S_2$ must at least communicate with $C$ in the same manner as $S_1$ by providing the same sets of taken places $(P'_{\text{IN}1} = P'_{\text{IN}2}) \wedge (P'_{\text{OUT}1} = P'_{\text{OUT}2})$.

Figure 4 shows four possible cases of relations we could have by comparing input places. These cases are obtained only after validating the first condition concerning taken places.
• **Case 1** (very weak): the set of input (resp. output) places of the OCNet $N_2$ is included in the set of input (resp. output) places of the OCNet $N_1$: $P_{IN2} \subseteq P_{IN1}$ (resp. $P_{OUT2} \subseteq P_{OUT1}$).

• **Case 2** (weak): the OCNets $N_1$ and $N_2$ share the set of the taken input (resp. taken output) places but not the free input (resp. free output) places: $[(P_{IN1} \cap P_{IN2} = P'_{in1}) \land (P_{IN2} \setminus P'_{in1})] \land [((P_{OUT1} \cap P_{OUT2} = P'_{out1}) \land (P_{OUT2} \setminus P'_{out1} = P'_{out})].$

• **Case 3** (strong): the set of input (resp. output) places of the OCNet $N_1$ is included in the set of input (resp. output) places of the OCNet $N_2$: $P_{IN1} \subseteq P_{IN2}$ (resp. $P_{OUT1} \subseteq P_{OUT2}$).

• **Case 4** (very strong): the sets of input (resp. output) places of the OCNets $N_1$ and $N_2$ are equal: $P_{IN1} = P_{IN2}$ (resp. $P_{OUT1} = P_{OUT2}$).

The interface relation of two OCNets is defined by the weakest relation obtained by comparing input places and comparing output ones. The combination of the various possible cases (input and output) gives sixteen cases presented in Table 1. The columns represent relations between the sets of input places, while the lines are reserved for relations between the sets of output ones. Crossing lines and columns define the relation between interfaces of two OCNets.

To define the relation between interfaces, we propose an algorithm `interface-relation-checking` that uses input interface-relation and output interface-relation functions for each interface type.

```
Algorithm Interface-relation-checking

Function Interface-relation (in: PIN1, PIN2; out: Relation1)
Function Interface-relation (in: POUT1, POUT2; out: Relation2)
Take the weakest relationship between Relation1 and Relation2
End

Function Interface-relation (in: PIN1, PIN2; out: Relation1)
Begin
Switch (PIN1, PIN2)
Case (PIN2 \subseteq PIN1)
   Relation1 \leftarrow Very Weak; Break;
Case (PIN1 \cap PIN2 = P'_{in1}) \land (PIN2 \setminus P'_{in1} = P'_{in1})
   Relation1 \leftarrow Weak; Break;
Case (PIN1 = PIN2)
   Relation1 \leftarrow Strong; Break;
Default
   Return False;
End
```
Table 1 Interfaces relations’ possibilities

<table>
<thead>
<tr>
<th>Output</th>
<th>Case 1</th>
<th>Case 2</th>
<th>Case 3</th>
<th>Case 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>POUT2 (\subseteq) POUT1</td>
<td>Very weak</td>
<td>Very weak</td>
<td>Very weak</td>
<td>Very weak</td>
</tr>
<tr>
<td>POUT1 (\cap) POUT2 = P(_{\text{out}}^1)</td>
<td>Very weak</td>
<td>Weak</td>
<td>Weak</td>
<td>Weak</td>
</tr>
<tr>
<td>POUT1 (\subseteq) POUT2</td>
<td>Very weak</td>
<td>Weak</td>
<td>Strong</td>
<td>Strong</td>
</tr>
<tr>
<td>POUT1 = POUT2</td>
<td>Very weak</td>
<td>Weak</td>
<td>Strong</td>
<td>Very strong</td>
</tr>
</tbody>
</table>

4.1.3 Example

Let us consider the substituted web service \(GPS_1\) modelled by the OCNet \(N_1\) of Figure 2, and let us consider \(N_2\), \(N_3\), and \(N_4\) the OCNets models of the respective web services \(GPS_2\), \(GPS_3\) and \(GPS_4\) that are a set of candidates’ web services to substitute \(S_1\). Figure 5 represents the OCNets models of these substituting candidates for a structural comparison with \(N_1\).

Figure 5 OCNets candidates for substituting \(N_1\), (a) OCNet \(N_2\) (b) OCNet \(N_3\) (c) OCNet \(N_4\)
Figure 5  OCNets candidates for substituting $N_1$, (a) OCNet $N_2$ (b) OCNet $N_3$ (c) OCNet $N_4$
(continued)
The sets of interface places of $N_1$ are defined as follows:

$P'_{int} = \{\text{Itinerary, Paths, Traffic\_Info}\}$ and $P'_{out1} = \{\text{Transport\_Mode}\}$

$P'_{out1} = \{\text{MAP\_Position, Distance, Trip\_Time}\}$ and $P'_{out1} = \{\text{Trip\_Cost}\}$

The comparison between $N_1$ interface and OCNets’ ones proposed in Figure 5 leads us to the following results:

- $N_1$ with $N_2$: $(P_{IN1} \cap P_{IN2} = P'_{int}) \land (P_{OUT2} \subseteq P_{OUT1}) \Rightarrow \text{weak} \land \text{very weak}$

Then the interface relation is very weak. The structural comparison is true, and we can proceed to the next checking step.

- $N_1$ with $N_3$: the interface place itineray of $S_2$ has not the same structure of the interface place itineray of $S_1$. Consequently the structural comparison fails, and $S_1$ cannot be substituted by $S_2$.

- $N_1$ with $N_4$: $(P_{IN1} \cap P_{IN4} = P'_{int}) \land (P_{OUT1} = P'_{out1}) \Rightarrow \text{weak} \land \text{very strong}$

Then the interface relation is weak. The structural comparison is verified and we can pass to the next step of checking.

### 4.2 Behavioural verification

The behaviour of web services is a key factor to validate their substitution (Chen and Huang, 2010). It presents the different states that a web service can take and the transition from a state to another.

In our approach, we propose the use of service automata (Stahl and Wolf, 2009) to capture the behaviour of an OCNet. This model is a reachability graph of OCNet’s internal part ($INNER$), whose arcs are labelled as follows:

- If the transition is related to an input (resp. output) interface place, it is labelled by ‘?’ (resp. ‘!’), followed by the name of this interface place.
- If the transition is internal, it is labelled by ‘·’.

The reachability graph is generated here using Platform Independent Petri Net Editor v4.3.0 tool (PIPE v4.3.0) (Bonet et al., 2007).

#### 4.2.1 Service automaton

**Definition**

A service automaton $A$ is defined by $A = (S, I_{in}, I_{out}, s_0, \Omega , \delta)$, where:

- $S$: set of states.
- $I_{in}$: set of input channels.
- $I_{out}$: set of output channels, with $I_{in} \cap I_{out} = \emptyset$.
- $I_{in} \cup I_{out} = I_{io}$ is the interface of $A$.
- $\delta \subseteq S \times (I_{io} \cup \{\tau\}) \times S$: a nondeterministic transition relation.
- $s_0 \in S$: initial state.
- $\Omega \subseteq S$: the set of final states.

A service automaton is well suited to describe an OCNet behaviour. The main idea can be summarised as follows:

- The initial state $s_0$ is the initial marking $m_0$ of the OCNet.
- Each state $s_i$ represents one of the possible marking $m^*$ that the OCNet can take.
- The label $x$ of arcs can represent an interface place (input or output) or an internal transition ($\tau$).
- The transition from a state to another is represented in the form $(s, x, s')$.

Figure 6 shows the service automaton $A_1$ of the OCNet $N_1$ of Figure 2.

**Figure 6** Service automaton $A_1$ of the OCNet $N_1$
The service automaton allows capturing the behaviour of an OCNet, which is based on its exchanged messages with other OC Nets of the composition. To compare the behaviour of two web services, we simply compare their services automata in order to verify if the web service $S_2$ behaves like $S_1$.

In a service automaton, the transition from an initial state to one of the final ones is considered as a communication sequence using different interface places. This sequence is represented as a path in the graph of service automaton. Comparing two paths belonging to two services automata $A_1$ and $A_2$ requires comparing their nodes while respecting the sequence in $A_1$ path.

In order to simplify this comparison we apply an abstraction on $A_1$ graph by keeping arcs labelled by taken places only. This abstraction is applied only on the graph of the service automata to replace because the substitute service is atomic, and its places have no status.

Indeed, free places will not affect the functioning of $S_1$, and they will not be taken into consideration in the checking process. Then the validation or not of the behavioural verification depends only on taken places.

Figure 7 Services automata of the candidate substitutes, (a) abstraction of the service automaton $A_1$ (b) service automaton $A_2$ (c) service automaton $A_4$
Figure 7 shows (a) the abstraction of the service automaton $A_1$ of Figure 6, and the two services automata (b) $A_2$ and (c) $A_4$, respectively generated from the OCNets $N_2$ and $N_4$ of Figure 5(a) and 5(c) respectively.

To compare the behaviour of two services automata, we propose a WS checking behavioural substitutability algorithm, which uses several procedures for behavioural verification.

The key idea of this verification is to browse the graphs of two services automata defining all their possible paths as sequences sets, in order to compare these sets. Two sequences are considered similar if the one belonging to $A_2$ contains all nodes of $A_1$ sequence, while respecting the order of these nodes.

**Algorithm** WS behavioural substitutability checking

$A_i$: service automaton of service $S_i$;
$S_0$: initial state of service $S_i$;
$S_0'$: initial state of service $S_2$;
$\text{Sequence}_i$: set of all communication sequences of service $S_i$;
Sub: Boolean;
We suppose that the procedure \textit{Brows paths} allows to browse the graph of a service automaton \( A \) while identifying all possible paths (messages sequences), and putting them into a set, in order to compare this last with those of other services automata.

For instance, the set of all possible paths in the automaton \( A_1 \) of Figure 7(a) is \( \text{Sequence}_1 = \{\text{Seq}_1, \text{Seq}_2, \text{Seq}_3\} \) where:

- \( \text{Seq}_1 = (\text{?Itinerary} \rightarrow \text{!MAP\_Position} \rightarrow \text{?Paths} \rightarrow \text{!Distance} \rightarrow \text{?Traffic\_Info} \rightarrow \text{!Trip\_Time}) \)
- \( \text{Seq}_2 = (\text{?Itinerary} \rightarrow \text{!MAP\_Position} \rightarrow \text{?Paths} \rightarrow \text{?Traffic\_Info}! \rightarrow \text{!Distance} \rightarrow \text{!Trip\_Time}) \)
- \( \text{Seq}_3 = (\text{?Itinerary} \rightarrow \text{!MAP\_Position} \rightarrow \text{?Paths} \rightarrow \text{?Traffic\_Info} \rightarrow \text{!Trip\_Time}! \rightarrow \text{Distance}) \)

The procedure \textit{sequences comparison} allows comparing two sets of messages' sequences belonging to two different graphs, and states if the new service automaton has at least the same sequences of the old one.

\begin{verbatim}
Procedure sequences comparison (in: Sequence1, Sequence2, out: Sub)
Sub \text{\textleft true /* Boolean */}
Sq \text{\textleft false /* Boolean */}
n1: Int /* Sequences' number in Sequence1 : n1 \leq n2 */
n2: Int /* Sequences' number in Sequence2 */
Seq1: a sequence x of service \( S_i \)
a, b: Int
1 \textbf{Begin}
2 \textbf{While} (a \leq n1) \textbf{Do}
3 \hspace{1em} b \text{\textleft 1;
4 \hspace{1em} Sq \text{\textleft false;
5 \hspace{1em} \textbf{While} (b \leq n2) and (Sq = false) \textbf{Do}
6 \hspace{2em} Node(in: Seq1^a, Seq2^b, out: S);
7 \hspace{2em} If (S = true) \textbf{Then}
\end{verbatim}
Proof: Let us consider a web service $S_1$ that must be substituted by another web service $S_2$. The behaviour of each web service is modelled by the respective services automata $A_1 = (S_1, I_{in1}, I_{out1}, s_{01}, \Omega_1, \delta_1)$ and $A_2 = (S_2, I_{in2}, I_{out2}, s_{02}, \Omega_2, \delta_2)$. Then, the behavioural verification of the substitution is true iff $A_2$ has at least all the sequences of $A_1$. This means that all possible execution scenarios of $S_1$ can be performed by $S_2$.

The principle of the procedure sequences comparison is to check if each sequence of $A_1$ (corresponding to an execution scenario) exists in $A_2$.

In this aim, for each sequence $Seq_1^a$ of $A_1$, the procedure explores all the sequences of $A_2$ looking for a corresponding sequence $Seq_2^b$ in $A_2$. If it exists in $A_2$ then, we pass to the next sequence $Seq_1^{a+1}$ of $A_1$ until the validation of the existence of all the sequences of $A_1$ in $A_2$. Otherwise the substitution is not valid. The task of the procedure node is to compare two specific sequences and it is described below.

The procedure node compares two sequences ($Seq_1$ and $Seq_2$) through their nodes. In particular, it checks if the following conditions are met:

1. $Seq_2$ contains at least all nodes of $Seq_1$: $Seq_1 \subseteq Seq_2$.
2. The order of the nodes must be respected.
3. $Seq_2$ may contain more nodes than $Seq_1$ only if they do not match any taken interface places: $Seq_1 \setminus Seq_2 \notin \{P_{in1} \cup P_{out1}\}$.

**Procedure** node (in: $Seq_1$; $Seq_2$; out: $S$)

```
Nd \leftarrow false /* Boolean */
S \leftarrow true /* Boolean */
i \leftarrow 1; /* Index */
j \leftarrow 1; /* Index */
v: Int /* nodes' number in a Seq_1: v \leq w */
w: Int /* nodes' number in a Seq_2 */
Begin
While (i \leq v) Do
```
Proof: The aim of the procedure node is to perform the comparison between two sequences of two services automata. It is based on the comparison of their nodes \((s, x, s')\). Two sequences are said to be substitutable if that of \(A_2\) contains at least all the nodes of \(A_1\). In order to facilitate the comparison, we apply an abstraction of the nodes labelled with the free interface places since they will not influence the functioning of the two web services. The principle of the procedure node is to compare the first nodes \(x_i\) and \(y_j\) of two sequences \(Seq^a_1\) and \(Seq^b_2\). If these nodes are equivalent, we pass to the following nodes \(x_{i+1}\) and \(y_{j+1}\), otherwise we remain on \(x_i\) and we pass to the node \(y_{j+1}\) of \(Seq^2\) as long as \(b \leq w\).

By applying the algorithm WS behavioural substitutability checking on services automata of Figure 7, we obtain that \(A_1\), \(A_2\) and \(A_4\) have a similar behaviour, because \(A_2\) and \(A_4\) contains all the sequences of \(A_1\). This means that GPS-2 and GPS-4 can substitute GPS-1.

5 Context-aware web services substitutability checking

The strength of web services is due to their efficiency to meet complex needs thanks to their composition and reuse. However, the substitution of a web service \(S_1\) belonging to a composition \(C_1 = (S_1 \oplus C)\), by another service \(S_2\) must preserve the services initially offered by \(S_1\). But the composition of \(S_2\) with \(C\) will modify the structure and the behaviour of the composition \(C_1\) by transforming it to \(C_2 = (S_2 \oplus C)\). When the two services \(S_1\) and \(S_2\) verify the context-independent substitutability, the compositions \(C_1\) and \(C_2\) will run properly. In fact, the context-independent substitutability is too strong condition and in general \(S_1\) and \(S_2\) may be substitutable in a context but not in another one.
This leads to say that two web services $S_1$ and $S_2$ are context-aware substitutable, if the compositions $(S_1 \oplus C)$ and $(S_2 \oplus C)$ are context-independent substitutable: $S_1 \approx_{\text{ci-sub}} S_2 \approx (S_1 \oplus C) \approx_{\text{ci-sub}} (S_2 \oplus C)$.

The verification process goes through four steps: the first one is the steps of the context-independent method, previously explained in Section 4. The remaining steps are applied to the old and the new composition, to compare and to verify their structures then their behaviours.

Figure 8 shows the different steps used in the process of context-aware substitutability checking.

**Figure 8** Main steps of context-aware substitutability checking

The substitution of a web service belonging to a services composition would certainly modify the structure of this last, then it will be considered as a new one. A structural comparison between the old and the new composition would be required to validate the substitution of two web services. Before presenting the process of this verification, it is necessary to check whether two web services can be composed.
5.1 Composability checking

As for web services, an OCNet can be composed with another one through their interface places, transforming a set of OCNets into a composite one. This last has an internal part represented by a set of composed OCNets’ INNERs (Hamadi and Benatallah, 2003; Xiong et al., 2010), in addition to the interface places that were transformed into internal places during composition. The interface of this new composite OCNet is the set of OCNets’ interface places which had initially free status.

5.1.1 Interface places composability

The composition of two web services $S_1$ and $C$ respectively modelled by the OCNets $N_1$ and $C$ is accomplished through their interface places with opposite directions (inputs of $N_1$ with outputs of $C$, and output of $N_1$ with input of $C$).

Let us consider $p_{in}^i, p_{out}^i$ ($p_{out}^j, p_{in}^j$) the respective interface places of $S_1$ and $C$ where:

- $p_{in}^i = \{\text{Name}^i, \text{Msg}_i^i\} \land \text{Msg}_i^i = \{\text{Field}_i^x / x \in [1...k]\} \land \forall \text{Field}_i^x \in \text{Msg}_i^i:
  \text{Field}_i^x = \{\text{Name}^{F_i^x}, \text{Type}^{F_i^x}\}$
- $p_{out}^j = \{\text{Name}^j, \text{Msg}_j^j\} \land \text{Msg}_j^j = \{\text{Field}_j^y / y \in [1...l]\} \land \forall \text{Field}_j^y \in \text{Msg}_j^j:
  \text{Field}_j^y = \{\text{Name}^{F_j^y}, \text{Type}^{F_j^y}\}$

$p_{in}^i, p_{out}^j$ are composable iff:

- $\forall \text{Field}_i^x \in \text{Msg}_i^i \exists \text{Field}_j^y \in \text{Msg}_j^j; \text{Msg}_i^i \subseteq \text{Msg}_j^j$ where:
  - $\text{Name}^{F_i^x} = \text{Name}^{F_j^y}$
  - $\text{Type}^{F_i^x}$ subtype of $\text{Type}^{F_j^y}$.

5.1.2 Composition of OCNets

Composing two OCNets $N_1$ and $C$ consists in combining them into a composite OCNet $C_1$ through their composable interface places. The remaining interface places represent the interface of the obtained composite (Hamadi and Benatallah, 2003). The composite’s INNER is a composition of OCNets $N_1$ and $C$ INNERs. In addition to interface places that become internal, it contains initial and final places $P_{\text{INIT}}$ and $P_{\text{FINAL}}$ connected respectively to initial and final transitions $T_{\text{INIT}}$ and $T_{\text{FINAL}}$. These lasts are respectively related to initial and final places of OCNets $N_1$ and $C$ INNERs (Hamadi and Benatallah, 2003).

Based on the work of (Srivastava and Koehler, 2003), the interface of the composition $(N_1 \oplus C)$ can be defined as follows:

$$\text{Comp}_{(N_1,C)} = \text{Comp}_a \cup \text{Comp}_b \text{ s.t.}:$$

- $\text{Comp}_a = P_{\text{in}} \cap P_{\text{out}}^C$
- $\text{Comp}_b = P_{\text{out}}^C \cap P_{\text{in}}$
- $P_{\text{in}} \cap P_{\text{out}}^C = P_{\text{out}}^C \cap P_{\text{in}}^C = \emptyset$. 
Figure 9  Composition of OCNets
Figure 9 shows the composition of the OCNets $N_1$ with the composite $C$ which represents the composition of the OCNets of web services MAPS, Geo-Pos and Traffic, represented in Figures 1.

### 5.1.3 OCNet composite

The composition of two OCNets implies the transformation of their *composable interface* places into *composed internal* ones. However, these lasts do not have the same definition as the usual internal places of the INNER. For this, a post-agglomeration (Feng et al., 2011; Haddad, 1991) is required between interface transitions related to these new places. Consequently, we will obtain new composition’ transitions with the same names of these composed places that we define as follows:

$$\forall t_i \in (P_{IN_1} \cup P_{INC}) \setminus \text{Comp}(N_1, C), \exists t_j \in (P_{OUT_1} \cup P_{OUT}) \setminus \text{Comp}(N_1, C).$$

$$T_{comp} = t_i \oplus t_j \text{ s.t. } T_{comp} : (T_a \times P_{comp} \times T_b) \rightarrow T_{comp} \text{ where } (P_{comp} = \text{Comp}(N_1, C)) \wedge (T_a = P_{comp}) \wedge (T_b = P_{comp}).$$

The interface of the composite is represented by the *free* interface places of the two composed OCNets, s.t.:

- $P_{IN} = (P_{IN_1} \cup P_{INC}) \setminus \text{Comp}(N_1, C)$
- $P_{OUT} = (P_{OUT_1} \cup P_{OUTC}) \setminus \text{Comp}(N_1, C)$.

The composite OCNet $C_1$ obtained by the composition $(N_1 \oplus C)$ can be defined as follows:

Let us consider two OCNets $N_1, C$ that we will compose using the composable places $\text{Comp}(N_1, C)$ s.t.:

$$N_1 = (P_1, P_{IN_1}, P_{OUT_1}, T_1, \Sigma_1, C_1, W_1^+, W_1^-, m_0^1, \Omega_1)$$

$$C = (P_C, P_{INC}, P_{OUTC}, T_C, \Sigma_C, C_C, W_C^+, W_C^-, m_0^C, \Omega_C)$$

The composition $(N_1 \oplus C)$ is a tuple $C_1 = (P_{INIT}, P_{FINAL}, T_{INIT}, T_{FINAL}, P, P_{IN}, P_{OUT}, T, T_{comp}, \Sigma, C, W^+, W^-, m_0, \Omega)$ where:

- $P_{INIT}, P_{FINAL}$: the initial and the final places.
- $T_{INIT}, T_{FINAL}$: the initial and the final transitions.
- $P = P_1 \cup P_C$: is a finite and non-empty set of places s.t. $P_1, P_C$ are respectively the sets of internal places of $N_1$ and $C$.
- $P_{IN}$: is a set of the input interface places s.t. $P_{IN} = (P_{IN_1} \cup P_{INC}) \setminus \text{Comp}(N_1, C)$.
- $P_{OUT}$: is a set of the output interface places s.t. $P_{OUT} = (P_{OUT_1} \cup P_{OUTC}) \setminus \text{Comp}(N_1, C)$.
- $T_{comp'}$: is a finite and non-empty set of composition’ transitions.
- $T$: is a finite and non-empty set of transitions s.t. $T = (T_1 \cup T_C) \setminus \{\text{Comp}(N_1, C) \cup (\text{Comp}(N_1, C))^c\}$. 


On context-independent and context-aware cloud

- Σ: is a finite and non-empty set of colours.
- C: P ∪ T → Σ is the colour domain application.
- W⁻: is the backward incidence matrix P ∪ T.
- W⁺: is the forward incidence matrix T ∪ P.
- m₀: is the initial marking.
- Ω: is a set of final markings of the OCNet C₁.

Figure 10 represents the composite OCNets C₁ obtained by the compositions (N₁ ⊕ C) of Figure 8, and shows the post-agglomeration between places and transitions of the composition. For example, T₁, T'₁ transitions that belong respectively to N₁, C OCNets, and Pᵢtinerary place were reduced into one composition’s transition Tᵢtinerary.

5.2 Structural verification of compositions

Validating the comparison between OCNets N₁ and N₂ interfaces confirms that N₂ contains all taken places of N₁, which implies that the composition N₂ with C uses, at least, the same places.

Compₐ(N₁,C) ⊆ Compₐ(N₂,C) ⇒ (Compₐ ⊆ Compₐ') ∧ (Compₐ ⊆ Compₐ') where:

- Compₐ (resp. Compₐ') designate the places PᵢN₁ and PᵢOUT (resp. PᵢOUT and PᵢNC) that are used for the composition of N₁ with C.
- Compₐ' (resp. Compₐ') designate the places PᵢN₂ and PᵢOUT (resp. PᵢOUT and PᵢNC) that are used for the composition of N₂ with C.

On the other hand, interface places of the compositions (N₁ ⊕ C) and (N₂ ⊕ C) depend respectively on those of N₁ and N₂ OCNets since these latters are composed with the same OCNet C. This means that the relation between (N₁ ⊕ C) and (N₂ ⊕ C) interfaces will have the same properties as between N₁ and N₂. However, this relation can be improved if the composition (N₂ ⊕ C) uses more interface places than the composition (N₁ ⊕ C). This means that N₂ is more compatible with C than N₁.

After studying the different possible cases of composition, we deduced four cases of relations between composition’s places, and each one affects differently the relation between the interface places of (N₁ ⊕ C) and (N₂ ⊕ C).

Table 2: Composable interface places relations

<table>
<thead>
<tr>
<th>Compₐ = Compₐ'</th>
<th>Compₐ ⊆ Compₐ'</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compₐ = Compₐ'</td>
<td>Case 1</td>
</tr>
<tr>
<td>Compₐ ⊆ Compₐ'</td>
<td>Case 3</td>
</tr>
</tbody>
</table>

Based on relations between N₁ and N₂ OCNets’ interfaces presented in Table 1, we can deduce those between (N₁ ⊕ C) and (N₂ ⊕ C) as follows:

- **Case 1**: N₂ uses the same places as N₁ for its composition with C, which implies that the relation between (N₁ ⊕ C) and (N₂ ⊕ C) is the same as between N₁ and N₂. So, Table 3 preserves the same properties of Table 1.
Figure 10  OCNets composites
On context-independent and context-aware cloud

Table 3  Case 1: \((\text{Comp}_a = \text{Comp}'_a) \land (\text{Comp}_b = \text{Comp}'_b)\)

<table>
<thead>
<tr>
<th>Output</th>
<th>(\text{Input})</th>
<th>(P_{\text{IN2}} \subseteq P_{\text{IN1}} ) (very weak)</th>
<th>(P_{\text{IN1}} \cap P_{\text{IN2}} = P_{\text{out1}}') (weak)</th>
<th>(P_{\text{IN1}} \subseteq P_{\text{IN2}}) (strong)</th>
<th>(P_{\text{IN1}} = P_{\text{IN2}}) (very strong)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_{\text{OUT2}} \subseteq P_{\text{OUT1}}) (very weak)</td>
<td>-</td>
<td>Very weak</td>
<td>Very weak</td>
<td>Very weak</td>
<td>Very weak</td>
</tr>
<tr>
<td>(P_{\text{OUT1}} \cap P_{\text{OUT2}} = P_{\text{out1}}') (weak)</td>
<td>-</td>
<td>Weak</td>
<td>Weak</td>
<td>Strong</td>
<td>Strong</td>
</tr>
<tr>
<td>(P_{\text{OUT1}} \subseteq P_{\text{OUT2}}) (strong)</td>
<td>-</td>
<td>Strong</td>
<td>Strong</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(P_{\text{OUT1}} = P_{\text{OUT2}}) (very strong)</td>
<td>-</td>
<td>Very strong</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

For upcoming cases (case 2, case 3, case 4) respectively represented by Table 4, Table 5 and Table 6, the composition \((N_2 \oplus C)\) uses more composable places than \((N_1 \oplus C)\). Therefore, the relation between interfaces of these compositions will be related to the directions of the extra places. These cases are considered as an improvement of the initial composition. The previous relation is obtained by the same method as the one obtained between \(N_1\) and \(N_2\).

- Case 2: the improvement applied on \(\text{Comp}_a\) (input of \(N_2\) and output of \(C\)) will transform the relations represented in columns (weak, strong) into new ones (resp. strong, very strong).

Table 4  Case 2: \((\text{Comp}_a \subseteq \text{Comp}'_a) \land (\text{Comp}_b = \text{Comp}'_b)\)

<table>
<thead>
<tr>
<th>Output</th>
<th>(\text{Input})</th>
<th>(P_{\text{IN2}} \subseteq P_{\text{IN1}}) (very weak)</th>
<th>(P_{\text{IN1}} \cap P_{\text{IN2}} = P_{\text{out1}}') (strong)</th>
<th>(P_{\text{IN1}} \subseteq P_{\text{IN2}}) (very strong)</th>
<th>(P_{\text{IN1}} = P_{\text{IN2}}) (very strong)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P_{\text{OUT2}} \subseteq P_{\text{OUT1}}) (very weak)</td>
<td>-</td>
<td>Very weak</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(P_{\text{OUT1}} \cap P_{\text{OUT2}} = P_{\text{out1}}') (weak)</td>
<td>-</td>
<td>Weak</td>
<td>Very weak</td>
<td>Strong</td>
<td>Strong</td>
</tr>
<tr>
<td>(P_{\text{OUT1}} \subseteq P_{\text{OUT2}}) (strong)</td>
<td>-</td>
<td>Weak</td>
<td>Strong</td>
<td>Very strong</td>
<td>-</td>
</tr>
<tr>
<td>(P_{\text{OUT1}} = P_{\text{OUT2}}) (very strong)</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
- **Case 3**: the improvement is only applied on \( \text{Comp}_b \) (output of \( N_2 \) and input of \( C \)), and that will respectively transform the relations represented in lines from weak and strong into strong and very strong.

- **Case 4**: the weak and strong relations represented in both lines and columns are respectively transformed into strong and very strong. This improvement is applied on \( \text{Comp}_a \) and \( \text{Comps} \).

<table>
<thead>
<tr>
<th>Output</th>
<th>Input</th>
<th>( P_{IN2} \subseteq P_{IN1} )</th>
<th>( P_{IN1} \cap P_{IN2} = P'_{out} ) (strong)</th>
<th>( P_{IN1} \subseteq P_{IN2} ) (very strong)</th>
<th>( P_{IN1} = P_{IN2} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( P_{OUT2} \subseteq P_{OUT1} )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( P_{OUT1} \cap P_{OUT2} = P'_{out} ) (strong)</td>
<td>-</td>
<td>Strong</td>
<td>Strong</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( P_{OUT1} \subseteq P_{OUT2} ) (very strong)</td>
<td>-</td>
<td>Strong</td>
<td>Very strong</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>( P_{OUT1} = P_{OUT2} )</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

In Tables 2–6, the non-empty cells correspond to the possible situations in a structural comparison between two compositions. The relations which are defined by capital letters reflect the improvement provided to interfaces in the composition of \( N_1 \) and \( N_2 \) with \( C \). By comparing composites \( C_1 \) of Figure 9 and the composite \( C_2 \) obtained by composing \( N_2 \) with \( C \), we obtain a weak relation (case 1).

The validation of this verification step allows passing to the next step. However, its non-validation stops the verification process, which causes the impossibility of substituting web services \( GPS-1 \) and \( GPS-2 \). In this step, we deduce that the only reason for the non-substitutability of web services \( S_1 \) and \( S_2 \) is the incompatibility in the composition of \( N_2 \) with \( C \). This fact represents the context on which the verification depends.

### 5.3 Behavioural verification of compositions

Modifying the structure of a composition due to the replacement of one of its services entails a new behaviour to it. This needs to verify the preservation of the initial behaviour by the new composition.

The validation of this last verification step confirms the substitutability of compositions \( (N_1 \oplus C) \) and \( (N_2 \oplus C) \). In this case, we deduce that web services \( S_1 \) and \( S_1 \) are substitutable in this context. However, the non-validation of the behavioural verification allows saying that the web service \( S_1 \) cannot be substituted by the web service \( S_2 \) in this context.

Indeed, the aim of this last verification step is to compare the behaviour of the old and the new composition \( (N_1 \oplus C) \) and \( (N_2 \oplus C) \). Therefore, the services automata \( AC_1 \) and \( AC_2 \) of the composites OC Nets \( C_1 \) and \( C_2 \) are used as models to compare the behaviour of these lasts.

A service automaton \( AC_1 \) of a composite \( C_1 \) is a tuple \( AC_1 = (S, I, s_0, \Omega, \delta) \), where \( I \) is the set of composition channels, and the remaining elements are already defined in Section 4.2.1.
By the last definition, the main idea can be summarised as follows:

- Each state \( s_i \) represents one of the possible marking that a composite OCNet can take.
- The composite’s interface is not considered since the composite’s behaviour depends on interface places used for this composition.
- Passing from a state to another is represented by \((s, x, s')\) where \( x \) (arc’s label) is an OCNet \( C_i \) transition.
- Arcs labelling \( x \) can be a composition’ transition ‘\( T_{COMP} \)’ (obtained by the post-agglomeration) or ‘\( \tau \)’ for the remaining transitions (internal or interface).

Comparing the services automata \( A_{C1} \) and \( A_{C2} \) is limited to arcs labelled with composition’ transitions \( T_{COMP} \), that leads to apply an abstraction on arcs labelled with ‘\( \tau \)’.

The same algorithm (\textit{WS behavioural substitutability checking}) and the same procedures (\textit{Brows paths, sequence comparison, and node}) are used for the behavioural comparison between \( A_{C1} \) and \( A_{C2} \). However, the procedure \textit{node} allows comparing two sequences through their nodes that are labelled with interface places, while the one used in this verification’ step is destined for composition’ transitions. In order that \textit{node} works in this step, we must replace the variable \( P^l_1 \) by \( T_{COMP} \).

The application of the algorithm \textit{WS behavioural substitutability checking} on services automata gives communication sequences between composed services. The comparison of these sequences, confirms that \( A_{C1} \) and \( A_{C2} \) have a similar behaviour, so, we can substitute the composite \( C_1 \) by \( C_2 \). This result leads to say that substituting the web services \textit{GPS-1} by \textit{GPS-2} is possible.

In this verification method, we show that the substitution of a web service belonging to a composition by another web service, does not guarantee the preservation of the proper functioning of the initial composition. This remains valid even if the new web service offers the same services and behaves in the same way as the old one. Indeed, this preservation is related to the compatibility in the new composition and to the conservation of initial behaviour.

Therefore, we can deduce that two web services may be substitutable in a context but not in another. This makes the decision of web services substitutability dependent on the context to which they belong.

6 Related work

In the research area of service oriented computing (SOC), web services composable and substitutability are still hot topics. Investigations in this field are still trying to develop effective methods to overcome these challenges. In (Hamadi and Benatallah, 2003), authors proposed an approach for the composition of web services and verified its reliability. In this aim, they used algebra constructs to express the semantics of web services. Then, they translated it into service nets (SN) in order to represent and verify the composition. This method ensures the preservation of the initial semantic of web services. In Xiong et al. (2010), authors deal with incompatibility problem of web services composition. They proposed composition net (C-net), which is based on Petri
nets, as a formal framework for web services. The principle of this approach is the transformation of an incompatible web service into a new compatible one, by adding information channels to their C-net model. In Jin et al. (2012), the authors proposed an approach for web services substitution that use service workflow nets (SWF-nets) as a model of web services. The verification process consists on extracting temporal transitions order for their comparison, and proving that the composition of two SWF-nets must have no deadlocks. This approach is considered as behavioural compatibility restrictions relax for context-independent substitutability. In Hameurlain (2013), the author proposed C-nets as a compositional framework for defining controllability and substitutability of service protocols that interacting asynchronously. This approach fixed the conditions for protocols controllability and proposed two substitutability relations to show the soundness of the framework. These relations were refined for checking substitutability on context-independent. In Stahl et al. (2009), the authors specified the services by open Petri nets, and presented their behaviour with services automata. In order to decide the services substitution, they proposed three substitutability notions for services, and applied the concept of an operating guideline on them. In another work of Stahl and Wolf (2009), the authors proposed an extension of the concept of operating guideline to characterise all correctly interacting partners of a service. In Li et al. (2011), the authors proposed an approach based on behaviour similarity to verify the substitutability of web services by using coloured Petri nets (PNs). They introduced a formal definition of context-independent similarity between behaviours, and they used a tool (PIPE2) to perform the verification. In Wang et al. (2015), the authors proposed coloured logic Petri nets (CLPNs) for determining the indeterminate data of logic output transitions in LPNs. They proposed an algorithm to generate the reachability tree of this model.

All these works are based on some classes of Petri net as a formal model for web services, while Bordeaux et al. (2014) proposed a labelled transition system (LTS) as a formal framework which is linked to both substitutability types, and the notion of compatibility between two web services. They discussed the web services substitutability types, considering two of them: substitutability in context-dependent and context independent.

In the present work, we suggested two verification methods to validate the web services substitutability. The first one copes with context-independent substitution and sets if a web service can substitute another one. The second one focuses on context-aware substitution and decides the substitutability of web services in a specific composition.

In our approach we use OCNets as a formal framework to model services. Although a large number of the existing works on composition and substitution are based on the use of a class of Petri nets, OCNets are higher level Petri nets model than those used in Hamadi and Benatallah (2003), Hameurlain (2013), Jin et al. (2012), Stahl et al. (2009) and Xiong et al. (2010). OCNets mainly allow specifying the tokens types capturing more faithfully the type of exchanged messages between web services. In fact, the composability and substitutability verification must not only compare the input/output interface places, but also the type of exchanged messages.

Moreover, it is worth mentioning that none of the already presented approaches which are based on Petri nets, considers the two types of services substitutions: context-independent and context-aware. Indeed Hameurlain (2013), Jin et al. (2012) and Li et al. (2011) focus only on context-independent substitution. In this paper we highlight
the fact that a substitution of one service by another one may be valid in a context but not in another, and hence the context-aware substitution deserves a particular attention.

Compared to the work of (Bordeaux et al., 2004) that deals with the two kinds of services substitutions using LTS framework, we use OCNets model to capture the structural as well as behavioural aspects of services, while the use of LTS focuses mainly on behavioural aspects rather than structural ones.

Structurally, an originality of our approach concerns the set of interface places: in almost works such Stahl et al. (2009), the substitute and the substituted services must have the same sets of interface places, while in our approach the substitute may have more interface places. This is more realistic since the substitute service usually offers (at least) more services than those offered by the substituted service.

Behaviourally, and compared to Jin et al. (2012), we do not limit to verify the deadlock freeness, we compare thoroughly the behaviour of the services: all execution scenarios of the substituted service must be performed by the substitute service.

Concerning the behavioural verification, Stahl et al. (2009) used the service automata as model, and they applied operating guidelines which are sufficient for their open net model. In our approach, we used algorithms on the services automata of our OCNets model. The originality of our approach is in the model, and the comparison and verification methods of this model.

The behavioural verification of services substitution is performed on services automata corresponding to OCNets reachability graphs generated using a recent version of the tools of Li et al. (2011) (PIPE v3.4.0).

7 Conclusions

Although the composition of web services consolidates their cooperation and reuse, the substitution of one of them could cause incompatibilities affecting the proper functioning of the entire composition.

This paper proposes two verification methods to decide if a given web service can substitute another one. Each method is related to one of the context’ relations: context-independent and context-aware.

The verification on context-independent is based on checking the substitutability of two web services without considering the composition to which they will belong, unlike context-aware verification which considers the substitutability of this composition, that makes the strength of this method compared to the previous one.

In this paper, we proposed OCNets for modelling ‘web services and their composition’. This model allows analysing and checking the structure of web services’ interfaces and defining the different quality relations that may exist between these interfaces. In addition to preserve the initial functionalities, these relations are considered as a criterion for the decision of a substitution.

We have also proposed the use of service automata for checking the behaviour of web services. This model is generated from OCNets, to capture and compare the web services behaviours.

In order to enrich this work, we plan to extend it on several criteria of verification and comparison (QoS, reliability, time, etc.). We also plan to develop tools to allow automatic substitution verification.
References


On context-independent and context-aware cloud


