Formal Semantics of Web Services Composition based on Colored Petri Nets and Graph Grammars

Sofiane Chemaa, Raida Elmansouri, and Allaoua Chaoui
MISC Laboratory, Department of Computer Science, Faculty of Engineering, University Mentouri Constantine, Algeria

Abstract: A web service is an application that permits exchanging information, interaction, and interoperability between different elements in a distributed environment. The web service composition aims to coordinate many web services to satisfy a particular need and bring a new functionality. The graph transformation encompasses all techniques that allow to pass from a formalism to another using the graph theory. In this paper, we propose an approach that deals with the formal semantics of complex web services composition. Colored Petri nets (CPNs) are used to model each web service. We use graph grammars and ATOM3 tool to generate automatically a CPN for a complex web service composition.

Keywords: Web services, Web services composition, Colored Petri nets, Graph grammars, ATOM3 tool.

1. INTRODUCTION

Technology is, now, looking for distributed applications to develop and increase its domains. Web services are distributed components that provide functionality to applications through a network. They can be used by applications written in different languages, and performed in different platforms on different systems. The concept of web services is essentially based on standard internet protocols, such as UDDI (Universal Description, Discovery and Integration) [6], WSDL (Web Service Description Language) [5], and SOAP (Simple Object Access Protocol) [4]; which offer solutions for description, publication, discovery, and interoperability of web services. However, the complex composition of web services is not guaranteed by them, therefore, different approaches have been proposed to solve this problem. The models represented in these works are expressed in different formalisms, the proposals of [9], [21], and [8] represent Petri net based models, [15] and [3] present web service models based on semantic annotations, however [14] introduces graph transformations used by the authors in order to model and compose web services.

In this paper, we propose a CPN-based approach to capture the semantics of complex web services composition. This approach uses Graph grammars [1] and ATOM3 tool [17] to implement the process of transformation. To this end, we have defined a meta-model for CPN [12], and a graph grammar with seven rules for performing the web services composition. More precisely, starting from two or three web services models, the process of composition allows to generate automatically a CPN for a compound web service. Seven operations of composition are implemented: the empty service, the sequence service, the mutual exclusion service, the iteration service, the arbitrary sequence service, the parallel service, and the discriminator service.

The remainder of this paper is organized in eight sections. In addition to the introductory section, the second section presents the modeling of web services and their specification using colored Petri nets. The third section is devoted to the algebra for composing Web services. The fourth section constitutes a brief overview of the model transformation subject. In the fifth section, we present a meta-model for colored Petri nets; as well as the formal definition of all used operators and the presentation of composition rules using the tool ATOM3. In the sixth section, we illustrate our approach through an example. In the next section, we outline some related work. Finally, we discuss some prospects and work in progress.

2. WEB SERVICES AS COLORED PETRI NETS

A Petri net [18] is a means for modeling the behavior of Discrete Event Dynamic Systems. This is a directed bipartite graph with two types of nodes: places are represented by circles and transitions are represented by rectangles. The arcs of the graph connect places and transitions in such a way that places can only be connected to transitions and vice versa. It is not permissible for places to be connected to other places, or for transitions to be connected to other transitions. Colored Petri Nets (CPN) are an important abbreviation of ordinary Petri nets; they constitute a very elegant representation of systems with a high level abstraction. The main idea of colored Petri nets is that the tokens can be of different types. The color of token represents the type of information but not the content of information. Thus, two tokens with the same color may be different. For more elaborate introduction to Petri nets and colored Petri nets, the reader is referred to [18], [19], and [12]. The use of visual modeling techniques such as colored Petri nets in the design of complex Web services is justified by many reasons. For example, visual representations provide a high-level yet precise language, which allows to
express and reason about concepts at their natural level of abstraction.

Web services are assimilated to a distributed system; that consists of a set of loosely coupled modules, which communicate through messages exchange. The behavior of a web service is defined by a set of operations [21]. Thus, modeling Web services using CPN is straightforward. Every operation in the service is modeled by a transition; the state of the web service is modeled by the position of tokens in the Petri net and the arrows between places and transitions are used to specify causal relations.

In the following, we give some formal definitions about Service net and Web service.

2.1. Service net

\( SN = (\Sigma, P, T, A, C, G, E, M_0, i, o, l) \) is a service net where:

- \( CPN = (\Sigma, P, T, A, C, G, E, M_0) \) is a colored Petri net where:
  - \( \Sigma \) is a finite set of non-empty types, also called color sets.
  - \( P \) is a finite and non-empty set of places.
  - \( T \) is a set of transitions.
  - \( A \) is a set of directed arcs \( A \subseteq (P \times T) \cup (T \times P) \) (the flow relation).
  - \( C \) is a color function, it is defined from \( P \) into \( \Sigma \) (i.e., \( C \) is a function that associates a subset of \( \Sigma \) to each place).
  - \( G \) is a guard function, it is defined from \( T \) into expressions.
  - \( E \) is an arc expression function, it is defined from \( A \) into expressions.
  - \( M_0 \) is the initial marking.
- \( i \) is the input place with \( i = \{ x \in P \cup T | (x, i) \in A \} = \emptyset \).
- \( o \) is the output place with \( o = \{ x \in P \cup T | (o, x) \in A \} = \emptyset \).
- \( l : P \rightarrow O \cup \{ \tau \} \) is a labeling function where \( O \) is a set of operation names and \( \tau \) is a silent operation.

The Silent operations are transition firings that cannot be observed. They are used to distinguish between external and internal behavior of the services.

2.2. Web Service

A Web service is a tuple [9]

\( S = (NameS, Desc, Loc, URL, CS, SN) \) where:

- \( NameS \) is the name of the service used as its unique identifier.
- \( Desc \) is the description of the provided service. It summarizes what functionalities the service offers.
- \( Loc \) is the server in which the service is located.
- \( URL \) is the invocation of the Web service.
- \( CS \) is a set of the component services of the Web service, if \( CS = \{ NameS \} \) then \( S \) is a basic service, otherwise \( S \) is a Composite service.
- \( SN = (\Sigma, P, T, A, C, G, E, M_0, i, o, l) \) is the service net modeling the dynamic behavior of the service.

The concept of service net and Web service being presented, we show, in the next section, how Web services can be incrementally composed. We recall that we use colored Petri nets as a means to offer a flexible and powerful algebra.

3. WEB SERVICE COMPOSITION

Given two or more Web services, each with a specific task, they sometimes cooperate in order to achieve a new task. This will result in a new value added service. For example a service of hotel booking can collaborate with a Web mapping service, like Google Maps API to show the customers the location of hotels. The collaboration of these services generates a composed Web service which performs the original individual tasks, as well as a new one.

In the following, we present an algebra that combines existing Web services for building more complex ones. We will take empty service, Sequence, Alternative, Iteration, Arbitrary Sequence, and Parallel as basic constructs. We, also, define one more developed construct which is the Discriminator.

3.1. Composition constructs

The BNF-like notation below describes a grammar defining the set of services that can be generated using algebra’s operators.

\[ S:: = \varepsilon | X | S \mid S | S \mid S \mid S \mid S \mid S \mid (S \mid S) \]

where:

- \( \varepsilon \) is the Zero Service (or empty service), i.e a service which performs no operation.
- \( X \) is a constant service. It consists of a service performing operation that cannot be split into sub-operations. This service is called Atomic.
- \( S_1 \mid S_2 \) represents a composite service that performs one service immediately followed by another, i.e \( \mid \) is a sequence operator.
- \( S_1 \mid S_2 \) represents a composite service that can reproduce either the behavior of \( S_1 \) or \( S_2 \), i.e \( \mid \) is an alternative (or a Mutual Exclusion) operator.
- \( S \) represents a composite service where one service is successively executed multiple times, i.e \( \) is an iteration operator.
- \( S_1 \iff S_2 \) is a composite service that performs any arbitrary sequence of the services \( S_1 \) and \( S_2 \), i.e \( \iff \) is an unordered sequence operator.
- \( S_1 \mid S_2 \) represents a composite service which performs the two services \( S_1 \) and \( S_2 \) at the same time and independently. The resulting service waits until the end of execution of \( S_1 \) and \( S_2 \), i.e \( \mid \) is a parallel operator.
- \((S_1 \square S_2) \implies S_3\) is a composite service that waits the outputs from \( S_1 \) and \( S_2 \). The first to response among \( S_1 \) and \( S_2 \) will be sequentially executed with the subservice \( S_3 \), i.e \( (\square) \implies \) is a discriminator operator. Note that \( S_1 \) and \( S_2 \) are performed in parallel and without communication.
4. MODEL TRANSFORMATION

The transformation between models is a process that converts a model to another model. This task requires a set of rules that define how the source model has to be analyzed, and transformed into other elements of the target model. The transformation engine takes the source model in input; execute the rules of transformation; and, finally, generate the target model in output. Graph grammars are used for model transformation. They are composed of production rules; each having graph in their left and right sides. Rules are compared with an input graph, called host graph. If a matching is found between the left hand side of a rule and a subgraph in the host graph, then the rule can be applied and the matching subgraph of the host graph is replaced by the right hand side of the rule. Furthermore, rules may also have a condition that must be satisfied, so that the rule to be applied, and actions to be performed, when the rule is executed. A rewriting system iteratively applies matching rules in the grammar to the host graph, until no more rules are applicable [13].

If the source and target meta-models are identical, the transformation is called as endogenous; otherwise it is called as exogenous [16]. If the level of abstraction does not change, the transformation is called as horizontal transformation. If the level of abstraction does change, the transformation is called as vertical transformation.

5. OUR APPROACH

In order to compose web services modeled by CPNs, we have used a horizontal endogenous transformation approach.

In this section, we present a meta-model for Colored Petri nets, whereupon we give the formal semantics of each operator and the composition rules implemented in AToM3 tool.

5.1. Colored Petri nets Metamodelling

To build a graphical editor for models in AToM3, we must define a meta-model for them. The formalism used in this work is the entity relationship diagram; the constraints are expressed in Python code. For modeling colored Petri nets in AToM3, we proposed two classes linked by two associations, as shown in Figure (1). We also associated with each element of the meta-model a graphical representation.

Class “CPNPlace”: represents the colored Petri net places. It has four attributes: Name, Type, Color set, and Tokens as shown in the figure (2). The attribute “Type” indicates the place type: Input-Place, Normal-Place or Output-Place, The attribute “Color Set” allows to define the color domain of each place, and finally, the attribute “Tokens” represents the initial marking of the place. The constraint “TypeOfPlace” gives the graphical appearance appropriate to each type of place, a Normal place is represented by a blue circle, an Input place is represented by a yellow circle and an Output place is represented by a double circle.

Class “CPNTransition”: represents the colored Petri net transitions. It has four attributes: Name, Operation, Type, and Guard. The attribute “Operation” is used to associate to each transition an action, the attribute “Type” indicates the
transitions type either Silent-transition or Normal transition and finally the attribute “Guard” represents the guard function of the transition (the transition can be Guarded or non-Guarded). The constraint “TypeOfTransition” provides the graphical appearance appropriate to each type of transition. A normal transition is represented by an empty rectangle, while a silent transition is represented by a black rectangle.

Association “CPNPL2TR”: links the class “CPNPlace” with the class “CPNTransition”. It is displayed by an arrow labeled by the attribute “Inscription”.

Association “CPNTR2PL”: links the class “CPNTransition” with the class “CPNPlace”. It is displayed by an arrow labeled by the attribute “Inscription”.

5.2. Graph grammar for generating a CPN of web service composition

In this part we present a formal definition of the composition operators in terms of services net, and the different composition rules using the tool ATOM3.

1) Empty Service: The empty service \( \varepsilon \) is a service that performs no operation. It is used for technical and theoretical reasons.

The empty service is defined as \( \varepsilon = (\text{NameS}, \text{Desc}, \text{Loc}, \text{URL}, \text{CS}, \text{SN}) \) where:

- \( \text{NameS} \) is the name of the new service.
- \( \text{Desc} \) is the description of the new service.
- \( \text{Loc} \) is the location of the new service (may be at the same server as one of the two component services).
- \( \text{URL} \) is the invocation of the new service.
- \( \text{CS} = \{ \text{Empty} \} \).
- \( \text{SN} = (\emptyset, p, p, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset) \).

2) Sequence: The sequence operator allows the construction of a service composed of two services executed one after the other. This construction is used, when a service should wait the execution result of another one before starting its execution. The service \( S_1 \triangleright S_2 \) is defined as \( S_1 \triangleright S_2 = (\text{NameS}, \text{Desc}, \text{Loc}, \text{URL}, \text{CS}, \text{SN}) \) where:

- \( \text{NameS} \) is the name of the new service.
- \( \text{Desc} \) is the description of the new service.
- \( \text{Loc} \) is the location of the new service (may be at the same server as one of the two component services).
- \( \text{URL} \) is the invocation of the new service.
- \( \text{CS} = \text{CS}_1 \sqcup \text{CS}_2 \).
- \( \text{SN} = (\Sigma, \text{P}, T, \text{A}, \text{C}, \text{G}, \text{E}, \text{M}_0, \text{I}, \text{O}, \text{L}) \) where:

\[
\begin{align*}
\Sigma &= \Sigma_1 \cup \Sigma_2; P = P_1 \cup P_2; T = T_1 \cup T_2 \cup \{ st_1 \} \, A = A_1 \cup A_2 \cup \{ (st_1, st_1), (st_1, i_2) \}, \, C = C_1 \cup C_2, \, G = G_1 \cup G_2, \, E = E_1 \cup E_2, M_0 = M_{01} \cup M_{02}, \, i = i_1, \, o = o_2, l = l_1 \cup l_2 \cup \{(st_1, \tau)\}.
\end{align*}
\]

The second service depends on the output of the first service, therefore, \( C(i_2) \subseteq C(o_1) \), must be satisfied, otherwise, the second service cannot be enabled, and the new composite service cannot work.

We have proposed the rule, shown on the figure 4 (a), to implement the sequence construct \((S_1 \triangleright S_2)\). The places 1 and 3 represent respectively the initial places of the services \( S_1 \) and \( S_2 \), while 2 and 4 represent respectively the final places of these services.

3) Alternative: Given two services \( S_1 \) and \( S_2 \), the alternative operator reproduces either the behavior of \( S_1 \) or \( S_2 \), but not both.

The service \( S_1 \triangleright S_2 \) is defined as \( S_1 \triangleright S_2 = (\text{NameS}, \text{Desc}, \text{Loc}, \text{URL}, \text{CS}, \text{SN}) \) where:

- \( \text{NameS} \) is the name of the new service.
- \( \text{Desc} \) is the description of the new service.
- \( \text{Loc} \) is the location of the new service (may be at the same server as one of the two component services).
- \( \text{URL} \) is the invocation of the new service.
- \( \text{CS} = \text{CS}_1 \sqcup \text{CS}_2 \).
- \( \text{SN} = (\Sigma, \text{P}, T, \text{A}, \text{C}, \text{G}, \text{E}, \text{M}_0, \text{I}, \text{O}, \text{L}) \) where:

\[
\begin{align*}
\Sigma &= \Sigma_1 \cup \Sigma_2; P = P_1 \cup P_2; T = T_1 \cup T_2 \cup \{ st_1 \}, \, A = A_1 \cup A_2 \cup \{ (st_1, i_2), (st_2, i_2), (st_1, st_2), (st_2, o_2) \}, \, C = C_1 \cup C_2, \, G = G_1 \cup G_2, \, E = E_1 \cup E_2, M_0 = M_{01} \cup M_{02}, \, l = l_1 \cup l_2 \cup \{(st_1, \tau), (st_2, \tau), (st_3, \tau)\}.
\end{align*}
\]

We have proposed the rule, shown on the figure 5 (a), to implement the iteration construct \( \text{S}_1 \). The places 1 and 2
represent respectively the initial place and the final place of the service $S_1$.

5) Arbitrary Sequence: The arbitrary sequence operator specifies the execution of two services that must not be executed concurrently. This construct is useful when there are no benefits to execute services in parallel.

The service $S_1 \Leftrightarrow S_2$ is defined as $S_1 \Leftrightarrow S_2 = (NameS, Desc, Loc, URL, CS, SN)$ where:

- **NameS** is the name of the new service.
- **Desc** is the description of the new service.
- **Loc** is the location of the new service (may be at the same server as one of the two component services).
- **URL** is the invocation of the new service.
- **CS** = $C_{S_1} \cup C_{S_2}$.
- **SN** = $(\Sigma, P, T, A, C, G, E, M_0, i, o, l)$ where:

$$\Sigma = \Sigma_1 \cup \Sigma_2, P = P_1 \cup P_2 \cup \{i, o, n_{p_1}, n_{p_2}, n_{p_3}, n_{p_4}, n_{p_5}\}, \ T = T_1 \cup T_2 \cup \{s_{t_1}, s_{t_2}, s_{t_3}, s_{t_4}, s_{t_5}, s_{t_6}\}, \ A = A_1 \cup A_2 \cup \{(i, s_{t_1}), (s_{t_1}, n_{p_1}), (s_{t_1}, n_{p_2}), (s_{t_1}, n_{p_3}), (s_{t_2}, n_{p_4}), (s_{t_3}, n_{p_5}), (s_{t_4}, n_{p_6}), (s_{t_5}, n_{p_7}), (s_{t_6}, n_{p_8})\}, \ C = C_1 \cup C_2, G = G_1 \cup G_2, \ E = E_1 \cup E_2, \ M_0 = M_{01} \cup M_{02}, \ l = l_1 \cup l_2 \cup \{(s_{t_1}, \tau), (s_{t_2}, \tau), (s_{t_3}, \tau), (s_{t_4}, \tau), (s_{t_5}, \tau), (s_{t_6}, \tau)\}.$$

We have proposed the rule, shown on the figure 5 (b), to implement the arbitrary sequence construct ($S_1 \Leftrightarrow S_2$). The places 1 and 3 represent respectively the initial places of the services $S_1$ and $S_2$, while 2 and 4 represent respectively the final places of these services.

![Figure 5. Iteration (a) and Arbitrary Sequence (b)](image)

6) Parallel: Given two services $S_1$ and $S_2$, the parallel operator builds a composite service, performing the two services ($S_1$ and $S_2$), in parallel and without interaction between them. The accomplishment of the resulting service is achieved, when the two services are completed.

The service $S_1 \parallel S_2$ is defined as $S_1 \parallel S_2 = (NameS, Desc, Loc, URL, CS, SN)$ where:

- **NameS** is the name of the new service.
- **Desc** is the description of the new service.
- **Loc** is the location of the new service (may be at the same server as one of the two component services).
- **URL** is the invocation of the new service.
- **CS** = $C_{S_1} \cup C_{S_2}$.
- **SN** = $(\Sigma, P, T, A, C, G, E, M_0, i, o, l)$ where:

$$\Sigma = \Sigma_1 \cup \Sigma_2, P = P_1 \cup P_2 \cup \{i, o\}, \ T = T_1 \cup T_2 \cup \{s_{t_1}, s_{t_2}\}, \ A = A_1 \cup A_2 \cup \{(i, s_{t_1}), (s_{t_1}, i_1), (s_{t_1}, i_2), (s_{t_2}, o_1), (s_{t_2}, o_2), (s_{t_2}, o_3)\}, \ C = C_1 \cup C_2, G = G_1 \cup G_2, \ E = E_1 \cup E_2, \ M_0 = M_{01} \cup M_{02}, \ l = l_1 \cup l_2 \cup \{(s_{t_1}, \tau), (s_{t_2}, \tau)\}.$$

We have proposed the rule, shown on the figure 6 (b), to implement the parallel construct ($S_1 \parallel S_2 \Rightarrow S_3$). The places 1, 3, and 5 represent respectively the initial places of the services $S_1$, $S_2$, and $S_3$, while 2, 4, and 6 represent respectively the final places of these services.

7) Discriminator: The main goal of the discriminator operator is to increase reliability and delays of the services through the Web. For customers, best services are those which respond in optimal time, and are constantly available. The composite construct, obtained by applying the Discriminator operator, submits redundant orders to different services performing the same task ($S_1$ and $S_2$ for the first service, which responds to the request, activates the service $S_3$. All other late responses will be ignored.

The service $(S_1 \Box S_2) \Rightarrow S_3$ is defined as $(S_1 \Box S_2) \Rightarrow S_3 = (NameS, Desc, Loc, URL, CS, SN)$ where:

- **NameS** is the name of the new service.
- **Desc** is the description of the new service.
- **Loc** is the location of the new service (may be at the same server as one of the two component services).
- **URL** is the invocation of the new service.
- **CS** = $C_{S_1} \cup C_{S_2} \cup C_{S_3}$.
- **SN** = $(\Sigma, P, T, A, C, G, E, M_0, i, o, l)$ where:

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3, \ P = P_1 \cup P_2 \cup P_3 \cup \{i, o, n_{p_1}, n_{p_2}, n_{p_3}, n_{p_4}, n_{p_5}, n_{p_6}, n_{p_7}, n_{p_8}\}, \ T = T_1 \cup T_2 \cup T_3 \cup \{s_{t_1}, s_{t_2}, s_{t_3}, s_{t_4}, s_{t_5}, s_{t_6}, s_{t_7}, s_{t_8}\}, \ A = A_1 \cup A_2 \cup A_3 \cup \{(i, s_{t_1}), (s_{t_1}, n_{p_1}), (s_{t_1}, n_{p_2}), (s_{t_1}, n_{p_3}), (s_{t_1}, n_{p_4}), (s_{t_1}, n_{p_5}), (s_{t_1}, n_{p_6}), (s_{t_2}, n_{p_7}), (s_{t_2}, n_{p_8}), (s_{t_3}, n_{p_9}), (s_{t_4}, n_{p_{10}}), (s_{t_5}, n_{p_{11}}), (s_{t_6}, n_{p_{12}}), (s_{t_7}, n_{p_{13}}), (s_{t_8}, n_{p_{14}})\}, \ C = C_1 \cup C_2 \cup C_3, G = G_1 \cup G_2 \cup G_3, \ E = E_1 \cup E_2 \cup E_3, \ M_0 = M_{01} \cup M_{02} \cup M_{03}, \ l = l_1 \cup l_2 \cup l_3 \cup \{(s_{t_1}, \tau), (s_{t_2}, \tau), (s_{t_3}, \tau), (s_{t_4}, \tau), (s_{t_5}, \tau), (s_{t_6}, \tau), (s_{t_7}, \tau), (s_{t_8}, \tau)\}.$$

$C(i_3) \subseteq C(o_1)$ and $C(i_3) \subseteq C(o_2)$ must be satisfied.

We have proposed the rule, shown on the figure 6 (b), to implement the discriminator construct $(S_1 \Box S_2) \Rightarrow S_3$. The places 1, 3, and 5 represent respectively the initial places of the services $S_1$, $S_2$, and $S_3$, while 2, 4, and 6 represent respectively the final places of these services.

![Figure 6. Parallel (a) and Discriminator (b)](image)
6. EXAMPLE

The proposed algebra verifies the closure property. This latter ensures that the product of any operation on services is itself a service to which we can apply algebra operators. We are thus able to build more complex services by aggregating and reusing existing services through declarative expressions of service algebra.

In this example, we have 6 simple services modeled with colored Petri nets as shown in figure 7.

- $S_1 (P_1) (P_2)$: converts a value from Terabyte to Bit.
- $S_2 (P_3) (P_4)$: converts a value from Terabyte to Gigabyte.
- $S_3 (P_5) (P_6)$: converts a value from Gigabyte to Kilobyte.
- $S_4 (P_7) (P_8)$: converts a value from Terabyte to Megabyte.
- $S_5 (P_9) (P_{10})$: converts a value from Megabyte to Kilobyte.
- $S_6 (P_{11}) (P_{12})$: converts a value from Kilobyte to Byte.

The figure 8 illustrates a composite service ($S$) modeled with a colored Petri net that takes as input a value, expressed in Terabyte, and calculates its equivalent in Bytes and Bits for output. This service is a sequence of operations on the set of the existing services ($S_1, S_2... S_6$).

$$S = S_1 \circ (S_2 \circ S_3) \circ (S_4 \circ S_5) \circ S_6.$$
modeling the process of Web service composition by a kind of Object-Oriented Petri Nets, but this approach was not formally defined. However, unlike our approach, all of these works are purely theoretical, and have not yet been automated.

Other approaches have used the graph transformation for web services composition [10] and [11]. Compared to all these approaches, our work is at the intersection of the two last. We used Petri nets for the modeling and the graph transformation to make the composition automatically. The intention is to take advantages from the formal treatment, the expressive comfort of Petri Nets, and at the same time to gain benefits from the graph transformation approach.

8. CONCLUSION

In this paper, we have presented a simple yet powerful approach for composing web services modeled by Colored Petri nets using graph grammars and ATOM3 tool. From two or three CPN web services model, our approach generates automatically a CPN for a compound web service, this latter is itself a service to which we can apply algebra operators. To attend this purpose, we have defined a meta-model for CPN and a graph grammar composed of seven rules. We have also illustrated our approach through an example.

In a future work, we plan to extend our approach with advanced operators that can support more complex Web service combination. We may also use reduction techniques to optimize the models before the analysis and the verification of certain properties by using appropriate tools such as INA [2].

References