CHECKING BEHAVIOURAL COMPATIBILITY IN SERVICE COMPOSITION WITH GRAPH TRANSFORMATION

Redouane Nouara and Allaoua Chaoui

Laboratory MISC University Abdel Hamid Mehri Constantine 2,
Constantine Algeria

ABSTRACT

The success of Service Oriented Architecture (SOA) largely depended on the success of automatic service composition. Dynamic service selection process should ensure full compatibility between the services involved in the composition. This compatibility must be both on static proprieties, called interface compatibility which can be easily proved and especially on behavioural compatibility that needs composability checking of basic services. In this paper, we propose (1) a formalism for modelling composite services using an extension of the Business Process (BP) modelling approach proposed by Benatallah et al. and (2) a formal verification approach of service composition. This approach uses the Graph Transformation (GT) methodology as a formal verification tool. It allows behavioural compatibility verification of two given services modelled by their BPs, used as the source graph in the GT operation. The idea consists of (1) trying to dynamically generate a graph grammar R (a set of transformation rules) whose application generates the composite service, if it exists, in this case (2) the next step consist in checking the deadlock free in the resulting composite service. To this end we propose an algorithm that we have implemented using the AGG, an algebraic graph transformation API environment under eclipse IDE.

KEYWORDS

Dynamic Service Composition, Graph Transformation, Service Composition Checking, Modeling Composite Service

1. INTRODUCTION

Service Oriented Architecture (SOA) is an ideal solution to the problems of distributed applications development, characterized by system heterogeneity and low coupling of components, since systems may not be developed by the same teams. Despite the great step made in this field by standardizing protocols of description (WSDL), discovery (UDDI), binding (SOAP) and a series of languages for manipulating services called (WS-*), all researchers and manufacturers are convinced that the success of the SOA approach is inevitably conditioned by a successful automation of dynamic service composition, in which a new service is dynamically created by assembling features of elementary services. In this case, the selection of the composed services is made on the fly. Although software vendors can guarantee the safety of their web services, the development, testing and verification of these web services are independently from other vendors' peers[1]. This raises the problem of composability of services offered by different providers.

Jae-Kwang Lee et al. (Eds) : CCSEA, AIFU, DKMP, CLOUD, EMSA, SEA, SIPRO - 2017
pp. 149– 162, 2017. © CS & IT-CSCP 2017 DOI : 10.5121/csit.2017.70214

For this end, several approaches have been proposed in the literature; generally based on planning tools, semantic extensions of service protocols or formal approaches. All these approaches incorporate the behavioural aspect of the service as part of their specification, in which the service's behaviour is associated with its static interface description (specified as a WSDL document). The specification of external and observable behaviour of services is required to achieve the composition operation because having only a syntactic compatibility level in the interaction interfaces cannot by itself guarantee the success of the interaction between two services[2][3]. The crucial problem that has been raised is whether a given service, selected based on some criteria, which can be functional or non-functional, may be successfully composed with the desired service in terms of interaction interfaces; even if they are not compatible in behavioural aspects.

Checking the composability of services plays an important role in the operation of automatic composition. If the non-formal approaches of composition, based on AI planning tools, have shown their limits at the expense of purely formal approaches, characterized by their mathematical basis [4]. These approaches are therefore ideal candidates that can contribute to solve the problem of checking composability.

Among these formalisms, the GT constitute an adequate tool for solving this kind of problems, due to (1) its pure formal basis (algebraic approach) and (2) it handles graphs which are the formalism generally used for modeling service behavior. However, major approaches proposed for service composition conceal an important aspect which is the modelling of composite services. In these approaches, a global view of services is used, which don't specify the real granularity of services, because services are generally modelled as black boxes or as atomic actions, which do not describe exactly the reality of things. This results in a coarse description of the composite service and an inaccurate specification of the interaction between services. To overcome these problems we propose, in this paper, a modelling formalism (an extension of the BP model) for describing the external and observable behavior of composite services that reflect also the interaction between elementary services, and an approach for checking the behavioural composability between two services using the GT formalism in the form of an algorithm for generating the composite service if it exists.

The article is structured as follows. In section 2 we introduce the concepts and definitions of the graph transformation formalism based on the algebraic approach. A state of the art on the use of graph transformation as a tool in service composition literature is presented in Section 3. Section 4 introduces the Business Process (BP) used as formalism for modelling behavioural services in our approach. In section 5 we detail the proposed service composition checking approach. In section 6 we present the rule generation process, our algorithm and its implementation. Finally we conclude this paper and give some future works directions in Section 8.

2. GRAPH TRANSFORMATION

Graphs offer a very rich mathematical formalism for modeling because they are a natural means for expressing complex system situations on an intuitive level. They are used to model all kinds of system states and specially the behavioural aspect with this mathematical basis. Graphs may be subject to compute operations that check some behavioural properties on system models. What justifies their wide uses in the specification data, diagrams, flow control, for the entities and relationships for UML diagrams[5]. One of these tools is Graph Transformation. Its basic idea is the change of a source graph, into another, result graph by applying some transformation rule(s), similar to Chomsky grammars in formal language theory. GT is used in several areas of computing for model transformation such as modeling and specification of visual processing models according to the MDA (Model Driven Architecture) approach or describing the

concurrency and distribution of systems [4]. In what follows, we present some basic definitions of the algebraic graph transformation used in our work.

2.1. Graphs and Graphs Morphisms

A labelled graph $G = (V, E, s, t, l_V, l_E)$ is a sextuplet with V a finite set of nodes (also called vertices), E a finite set of edges and two functions s and t defined by $s, t: E \to V$, which define the sources and targets of edges respectively. $l_V: V \to L_V$ and $l_E: E \to L_E$ are labelling functions that attribute a node's label from the set LV (respectively edge's label from LE with $L_V \cup L_E = L$ the labelling set. Let be two graphs G1 and G2 defined by $G_i = (V_i, E_i, s_i, t_i, l_{V_i}, l_{E_i})$ with $i \in \{1, 2\}$. A graph morphism f between G1 and G2 is $f: G_1 \to G_2$ with $f = (f_E, f_V)$ consists in two functions $f_V: V_1 \to V_2$ and $f_E: E_1 \to E_2$, that preserves the source and target functions defined by $f_V \circ s_1 = s_2 \circ f_E$ and $f_V \circ t_1 = t_2 \circ f_E$ in [4].

2.2. Algebraic Graph Transformation

The algebraic graph transformation approach is based on pushout constructions used to model the gluing of graphs. In this approach there are two main variants the Single Pushout (SPO) and Double Pushout (DPO). In the latter two gluing constructions are used, where in the first only one construction is used as depicted in Figure 1. The interested reader can find more details in[4].

The operation of transforming a given source graph G to a resulting graph H is done by applying a production rule p, defined in SPO by: $p = L \xrightarrow{r} R$ where L and R are two graphs called the left hand (LHS) and right hand (RHS) of the rule respectively, r is a morphism between L and R as illustrated in Figure 1.

The rule p is applicable if and only if there is a morphism m between the graphs L and G ($m: L \to G$) which takes the form of an image of L in G. The target graph H is constructed by adding the graph R to the graph G, and from the resulting graph the graph L is removed [4]. To prohibit the execution of a rule, some conditions can be added, called Negative Application Condition (NAC), this forbids some graph structure X to be present in the source graph G before or after applying a rule. Formulated by: A NAC(n) on L is a graph $n: L \to X$, a graph morphism $m: L \to G$ satisfies NAC(n) on L iff: $\nexists\, q: X \to G$ such that $q \circ n = m$ [6].

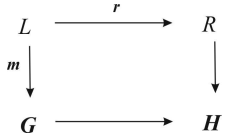


Figure 1. Graph Transformation Principle.

3. RELATED WORK

The use of Model Transformation (MT) in general and especially the GT in the formalization and checking of distributed architectures and service composition as a special case has got very little attention in the literature. Major works proposed in checking service composition uses other formalisms.

In [7] a structural approach is proposed, where composite service is modelled as a kind of Petri Net called Open Net. The service composition checking used done by using results of structure theory of Petri net, in which the necessary and/or sufficient structural conditions are identified for ensuring a behavioural compatibility between two services. Bentahar [1] used a model-checking based approach in order to verify if composite service design meets some desirable behavioural properties. Composite service is modelled based on a separation between two aspects, an operational behaviour illustrates the business logic that a composite service implements and a control behaviour illustrates and states the constraints which the operational behaviour should satisfy. These two behaviours are formally defined using automata-based techniques. Foster[8] propose a checking service composition approach based on verification of properties. Created from design specifications and implementation models; to confirm expected results from the viewpoints of both the designer, modelled in UML, and implementer. The result compiled into the Finite State Process notation (FSP) in order to reason about the concurrent programs. Bultan [9] propose WSAT4; a framework for analysing the interactions among composed Web services modelled as conversations (a sequence of exchanged messages). The composite web service is modelled as a set of peers (elementary services) which communicate with each other using asynchronous messages via a FIFO queue, where each peer is modelled as a state machine. In [10], the Classical Linear Logic (CLL) is used to verify the correctness of web service composition. The process consists of finding a proof for a requested service with available services stated as assumptions. If the proof is found this means a valid composition exists, and then a process calculus realisation of the composite service can be automatically extracted. Hamadi [11] propose an approach that uses Petri nets for modelling composed services, the service composition is done by a merging process of elementary services to a Petri net that models the control flow of the composite service. This approach uses an algebra that specifies different concurrent execution forms between composed services. The most similar work is that of DING [12], where an approach is proposed for the identification of structural conflicts (behavioural incompatibility) in inter-enterprise business process models. This approach is based on an algorithm that employs condition reachable matrix.

4. SERVICE MODELLING FORMALISM

The choice of the formalism used to model the service behaviour constitutes the key element in any approach for service composition. The model should describe as accurately as possible the behaviour of the service and its interaction with its environment. In what follows we present the modelling formalism used in our approach to describe service behaviours whether, elementary or composite.

4.1 Single Service Modeling

We adopt, in this work, the service model proposed by [13] [3] [14] called Roman model. This model captures conversations, the external and observable behavior that a service supports; it is defined as the ordered set of messages exchanged between the service and its client during their interaction. The Roman model uses deterministic finite state automata (DFA), in which the states represent the different phases through which the service passes during its life cycle, and transitions model the events and/or internal actions that occur during service interaction. These transitions are triggered by messages exchanged between the service and its client, which corresponds to (1) an invocation of a service method or a response to the latter, or (2) the advent of an internal event to the service as an expiration of a waiting period. The model has a single initial state and several final states, the transitions labeled by messages are associated with polarities defined by symbols +,- in [13] or ? and ! in [14] that specifies the origin of messages. Polarity + (respectively -) indicates that the message is received (respectively sent) by the service. Each BP is associated with a current state that describes the current state of the BP, initially

following the invocation of the service by the client, it starts at the initial state and at each transition it changes the current state until reaching a final state which indicates the end of the interaction.

To use the graph transformation approach, we formalize the external behavior of services in this article with a graph language notation as mentioned in [4], instead of the automaton notation used in [13] [3]. In order to integrate the BP specificities in a graph model notation, we extended the graph definition by initial and final states. Let A be a BP of a service, formally we use the following definition of $A = (V, E, s, t, l_V, l_E, v_0, F)$ with V and E describe the sets of states and edges respectively, s and t are the start and target functions of edges. The sets lV and lE represent the state and edge labels respectively (with their respective polarities). v_0 the initial state with $v_0 \in V$, F the set of final states (with $F \subset V$).

As an example, Figure 2 describes the modelling of an e-commerce service that manages the order of some goods, with start as initial state and the set *{Cancel, delivery}* as final states. Labels *{login (+), confirm_order (+), payment (+), delivery (-)}* are a sequence of exchanged messages between the service and the client; while the messages depend on the polarity sign. This sequence constitutes a valid conversation, *cancel(-)* is an internal event, automatically generated by the service and sent to the client after the timeout of payment by the customer.

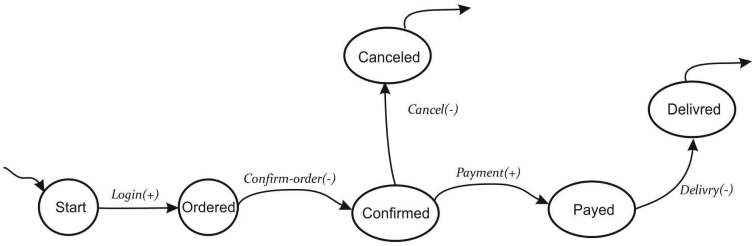


Figure 2. Sample Business Process.

4.2 Composite Service Modeling

The Roman model used to describe service external behaviour is well suited for describing single service behavior. However, it suffers from the lack of concurrence modelling between elementary services in the case of composite service, because DFA formalism has only a single current state describing the entity running at a given time; while in the case of composite services there is a set of services that run in parallel and a fully distributed manner. This drawback inherent from DFA constitutes a big handicap for modelling composite services and specifying the multiple forms of concurrence existing between elementary services. To overcome this obstacle, we propose an extension of the Roman model in order to support the specification of concurrence in a composite service modelling. We use a Multi Current State DFA, for specifying the concurrent execution between elementary services. In this model we formulate a composite service Cs as:

$$C_S = (A, S_0, I)$$

With: *A* a set of service BPs modelling elementary services, S_0 is the set of current active states and I the set of invocation edges. It specifies the execution of composed services and their life cycle progress. *A* is equal to the number of elementary services involved in the composition. When a service calls another one, initially only the caller service has its current state active (in S_0). Each time an elementary service is invoked, its current state (generally the initial state) becomes active and added to S_0. The current state dynamically changes every time the service exchanges messages with accordance to its BP until the end of service execution (reaching a final

state). In this case, the current active state is disabled and removed from S_0. The set *I* describes the interactions between the elementary services. They model either (1) a service invocation or (2) a response from the service following an invocation by another one. Initially, the set *I* is empty and these invocation edges are dynamically created at each service invocation (added to I) and deleted at the end of executions service. The proposed model is a Multi Current State Automata (as many current states as elementary BPs). The composition is carried out following service invocations. Each time a service S_A invokes, from state s_a, another service S_B to state s_b with a message M. This invocation results in the creation of an invocation edge starting from the state s_a to the state s_b and labelled with message M as depicted in Figure 3.

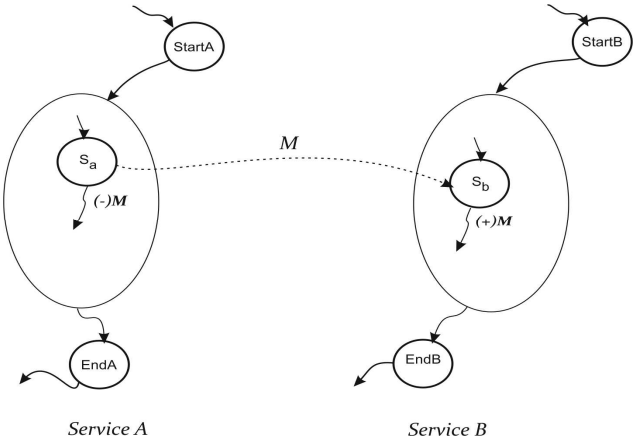


Figure 3. Composite Service Model's.

As an example, let be a service S_A described with its BP shown in Figure 4(a) that interacts with a service S_B (shown in Figure 4(b)) to create a composite service S_c. Initially:

$$C_S = \{(A = \{BP_{SA}\}, S_0 = \{StartA\}, I = \emptyset)$$

Service S_A invokes, from the state A_4 , Service S_B to state $Start_B$, the composition operation creates an invocation edge libelled with the exchanged message m_7 and added to the set I. The created edge connects the state A_4 to $start_B$ as depicted with dotted line in Figure 5. The composite service becomes:

$$C_S = \{(A = \{BP_{SA}, BP_{SB}\}, S_0 = \{A_4, StartB\}, I = \{m_7\})$$

Service S_B responds to Service S_A by sending one of two messages:

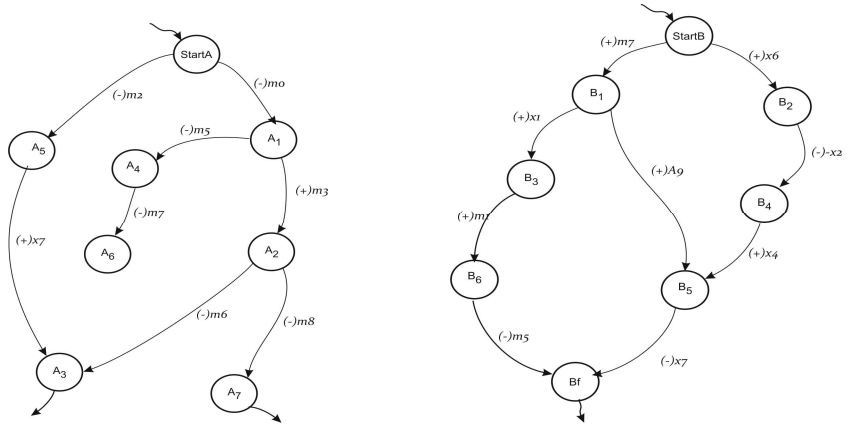
m_5 sent from B_6 to A_1 which creates the edge labeled by m_5 between B_6 to A_1 and the composite service becomes :

$$C_S = \{(A = \{BP_{SA}, BP_{SB}\}, S_0 = \{A_1, B_6\}, I = \{m_7, m_5\})$$

x_7 sent from B_5 to A_5 which result in :

$$C_S = \{(A = \{BP_A, BP_B\}, S_0 = \{A_5, B_5\}, I = \{m_7, x_7\})$$

After this, Service S_B goes to the final state B_f which will complete the operation of composition between the two services.



(a) BP A. (b) BP B.

Figure 4. Example of elementary services composition.

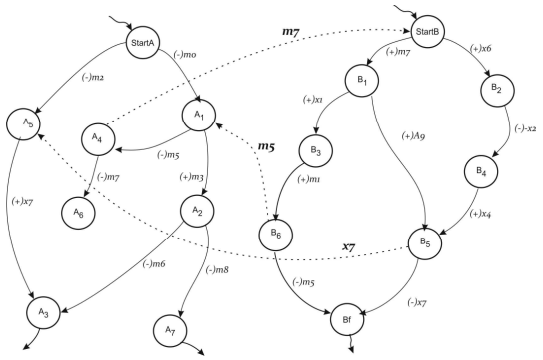


Figure 5. BP of Composite Service.

5. SERVICES COMPOSITION VERIFICATION APPROACH

The proposed approach, for checking service composition, uses GT as verification formalism. Since the latter has the major advantage of having a formal process for handling graphs (either simple or typed attributed graphs) [4]. This allows formalizing the necessary conditions that must be met to conclude the success or failure of service composition. The purpose of this approach is to check whether two elementary services S_1 and S_2 modelled by their respective BPs (1) can be syntactically composed by generating a valid composite service S_c i.e. the set of invocation edges I is not null and (2) check behavioural compatibility specially the deadlock free in the composite service. The GT is used as a formal tool to merge the two graphs for giving the composite service Sc (if it exists) by automatically generating a Graph Grammar G = (P, G_0) where P is a set of transformation rules called GTS (Graph Transformation System) and G0 the start graph.

P = {p_i, $1 \leq i \leq n$} where: pi is a rule that represents the interaction between the two services that can be either:

A service invocation: In which one of the two services invoke a method of the second service or inversely a response to a previous invocation of another service.

An internal event generated by a service like timeout expiration.

The P rules have an identical structure which consists of creating an invocation edge between two states, one belonging to each service. The start graph G_0 is represented by the two graphs (S_1 and S_2).

The two services are syntactically composable if the graph grammar G exists i.e. the set P is not empty, in this case the execution of its rules on G_0 generate the composite services Sc.

The existence of Sc does not imply that the two services can be composed because some conditions must be checked before concluding the composability of the two services. In what follows, we define and formalize these necessary conditions.

5.1 Conditions of Services Composability

As mentioned in the beginning of this paper, managing services in a fully open and totally dynamic environment requires, before a service be involved in a composition process, to operate some checks that confirm a priori the success of their composability. These verifications must be done at the same time at syntactic and behavioural levels as detailed in [15] and [16]. Namely, the syntactic consist of checking the mismatches occurring in service interfaces and behavioural aspects (called mismatch in service Business Protocol). The first aspect was already discussed in literature and is not considered in this paper. The second one is to check the behavioral compatibility which can be either (1) a deadlock free of the conversations between the two services or (2) an unspecified reception of a message from the other service. In this paper, only the conversation deadlock free is covered because the unspecified reception of messages cannot be checked (1) before the runtime of service composition and (2) the BPs alone cannot guarantee that the message may be intended for another service.

5.1.1 Existence of Invocation's Message(s)

Checking the existence of exchanged messages between the two services is to verify the composability in a purely syntactic point of view, i.e. that there is at least one message issued by one service (with polarity (-) in its BP) and at the same time expected by the other service (with polarity (+)) as described in Figure 4. In this article, we do not take into account the compatibility of exchanged messages in terms of structure, i.e. the number and parameter types, nor the semantic, i.e. the interpretation of a data element's meaning or an operation's function that can be easily checked. The existence of exchanged messages can be formalized as follows, let be S1 resp S2 two services defined by $S_i = (V_i, E_i, s_i, t_i, l_{V_i}, l_{E_i}, v_{0_i}, F_i)$ with $i \in \{1, 2\}$ and BPS1 (resp BPS2) the Business Process of S1 (resp S2). There is an exchanged message(s) between S1 and S2 if and only if:

$$\exists\, e \in (E_1 \cap E_2), \exists\, s_i \in V_1, s_k \in V_2 \ where \begin{pmatrix} s_i.(-)e \in BP_{S_1} and\ s_k.(+)e \in BP_{S_2} \\ or \\ s_i.(+)e \in BP_{S_1} and\ s_k.(-)e \in BP_{S_2} \end{pmatrix}$$

5.1.2 Deadlock-Free Conversations

As mentioned at the beginning of this paper, syntactic compatibility does not conclude the composability of two services, a second condition must also be checked, it relates to the behavioural compatibility between the two services. This compatibility consists of verifying the deadlock-free between the two services conversations. This situation is characterized by the case where each service is in a waiting situation for reception of a message sent by the other service.

As shown in figure 5, service S1 (stay in state A1) is awaiting receipt of message m5 from service S2 and at the same time S2 (In state SartB) expects the message m7 coming from S1.

To formalize the deadlock-free we define the function *Poid(x)*, with x a BP state, it returns the set of states reachable from x. The set I of invocation messages between S1 and S2 defined by I = $\{m_i(a,b),\ 1 \leq i \leq n\}$, with a $\in$ V1 and b $\in$ V2 or inversely where $m_i(a, b)$ is a message exchanged between two services (sent from State a to State b). We conclude to deadlock free between $S_1$and S_2 if and only if:

$$\exists\, s_i, s_j \in V_1, \exists\, x_k, x_p \in V_2 \; where \begin{pmatrix} s_j \in Poid(s_i) \wedge x_p \in Poid(x_k) \\ and \\ \exists m_i(x_p, s_i) \in I \wedge \exists m_j(s_j, x_k) \in I \end{pmatrix}$$

5.2 Rules generation

The generated transformation rule set p_i, if it exists, has the same structure, as depicted in Figure 6, and characterized by the facts (1) the left side (LHS) is constituted by two states a_1 and a_2 one belonging to each service (see Figure 6(b)), (2) the right side (RHS) is constituted by the states (a_1 and a_2) connected by an edge (Figure 6(c)) and libelled with the exchanged invocation message. The application of p_i results in creating the edge between a_1 and a_2. In order to avoid an indefinite execution of the grammar rules, and impose a single execution, we add for each rule a NAC which is the RHS of the rule as depicted in Figure 6(a). (3) Grammar rules are not subject to any execution order and therefore can be executed in a random order. In the next section, we present our proposed algorithm for the automatic generation of the grammar whose application creates the composite service if it exists.

(a) NAC (b) LHS (c) RHS

Figure 6. Structure of Generated rules.

6. ALGORITHM FOR CHECKING APPROACH

The BP model used for formalizing the external service behaviour as automata-based graph presents an interesting feature that be an oriented and rooted graph i.e. it has (1) a special single state called root (in our case the BP initial state), from which all other graph states are reachable, and (2) the output edges of each state are bounded by a constant number because BPs are deterministic finite automata. This feature has a major advantage that allows the development of algorithms for processing BPs whose complexity is not exponential i.e. the execution time is limited as proved by [17] and [18] [19]. Based on this, in our approach we propose an algorithm for automatically generating the composite service grammar. The algorithm calls a weight function *Poid* that returns, for a given node, the set of nodes reachable from this node (see Algorithm 1). Essentially based on recursive functions, the algorithm operating principle consist, in the first step of browsing the states of the first graph, starting from the start state, and at each time it searches the existence of an invocation message between this state and another one belonging to the second service, which satisfies the existence of invocation message condition (cited above in Section 5.1). If an invocation message exists, the two identified states are converted to a transformation rule as explained in the previous section and added to Set I . In the case where the invocation message list is empty we conclude to a syntactic incompatibility between services. Otherwise in the second step it executes the generated rules to create invocation

edges between services. Finally, it checks the deadlock-free between the two conversations by checking Equation 2, if this condition is meet; it concludes to a behavioural incompatibility and therefore the two services are composable otherwise the two services can be composed.

***Algorithm** 1:Function Poid: compute the set of nodes reachable from a given node.*
***Require**: v a graph node.*
***Ensure**: the list of reachable nodes from v*
1: ***if*** *Terminal(v)* ***then***
2: *P oid ← {v}*
3: ***else***
4: *k ← nodecount(v)*
5: ***for*** *j = 1 → k* ***do***
6: *P oid ← P oid ∪P oid(nextnode(v; j))*
7: ***end for***
8: ***end if***
9: ***return*** *P oid*
10: ***END***.

Algorithm 2: Check the composability of two services based on their BPs.
Require: $BP_1 = (V_s, E_s, s_s, t_s)$
Require: $BP2 = (V_t, E_t, s_t, t_t)$ Services.
Ensure: *R* Set of rules.
1: $V_s = \{v_s^0, .., v_s^n\}$ and $E_s = \{e_s^0, .., e_s^m\}$
2: $V_t = \{v_t^0, .., v_t^k\}$ and $E_t = \{e_t^0, .., e_t^l\}$
3: sta ← *start_state*(BP 1)
4: stb ← *start_state*(BP 2)
5: *compute*();
6: *search_event*(sta; stb);
7: *run_rules*();
8: ***if*** Nbrule = 0 ***then***
9: print("SY NTACIC INCOMPAT IBILIY BETWEEN SERVICES")
10: ***else***
11: DEADLOCK ← false
12: *list_inv_arcs ← get_liste_invocation;*
13: ***for*** k = 1 ***to*** size(*liste_inv_arcs*) ***do***
14: arc1 ← *liste_inv_arcs*(k)
15: source_arc1 ← get_source(arc1);
16: dest_arc1 ← get_destination(arc1);
17: ***for*** j = k + 1 ***to*** size(liste_inv_arcs) ***do***
18: arc2 ← liste_inv_arcs(j);
19: source_arc2 ← get_source(arc2);
20: dest_arc2 ← get_destination(arc2);
21: ***if*** source_arc1 ∈ Poids(dest_arc2) and
source_arc2 ∈ Poids(dest_arc1) ***then***
22: DEADLOCK ← true;
23: ***end if***
24: ***end for***
25: ***end for***
26: ***if*** DEADLOCK ***then***
27: PRINT("deadlock between the two bps —> behavioral incompatibility")
28: ***else***
29: PRINT("behavioral compatibility —> the two services are compatible")
30: ***end if***
31: ***end if***
32: ***END.***

6.1 Algorithm Complexity

The algorithm has as input the two BPs A and B that are rooted graph, suppose that A have n nodes, and the number of edges connected to each node is bounded by an integer k. The graph B has m nodes each one bounded by e edges. The algorithm browse the first BP from the initial sate, and for each edge it check the existence of an invocation edge with a node belonging to the second BP. This operation is done in m^e instructions. With n node in the first graph, the algorithm need: $n^{k(m^e)}$ instructions, in worst case, to achieve the execution.

7. IMPLEMENTATION

The above-mentioned algorithm has been implemented using the Eclispe java IDE and the API AGG (see Homepage http://user.cs.tu-berlin.de/˜gragra/agg/) for graph transformation. This choice is guided by the AGG features that provide a set of necessary functions for dynamically manipulating components of GT. AGG is a general development environment for algebraic graph transformation systems which follows the interpretative approach. It allows the dynamic creation of all GT components (typed graphs, graphs, rules, NAC, nodes, edges), and to dynamically manipulate them by adding or removing operations. Its special power comes from a very flexible attribution concept and graphs are allowed to be attributed by any kind of Java objects [20]. These features of dynamically managing the GT and automatic execution of grammar have guided our choice to using the AGG API. As an example, the two services shown in Figure 4 whose corresponding graph represented with the AGG framework (a GUI environment) shown by the screenshot in Figure 7. This graph introduced to our application as input generates an output on the console (see Figure 8) that describes the different steps done during the execution. In which three invocation messages are founded as detailed in Section 4.2 and depicted in dotted line in the resulting generated composite service graph shown in Figure 9. After processing, two invocation edges have a deadlock situation which proves behavioural incompatibility as a result.

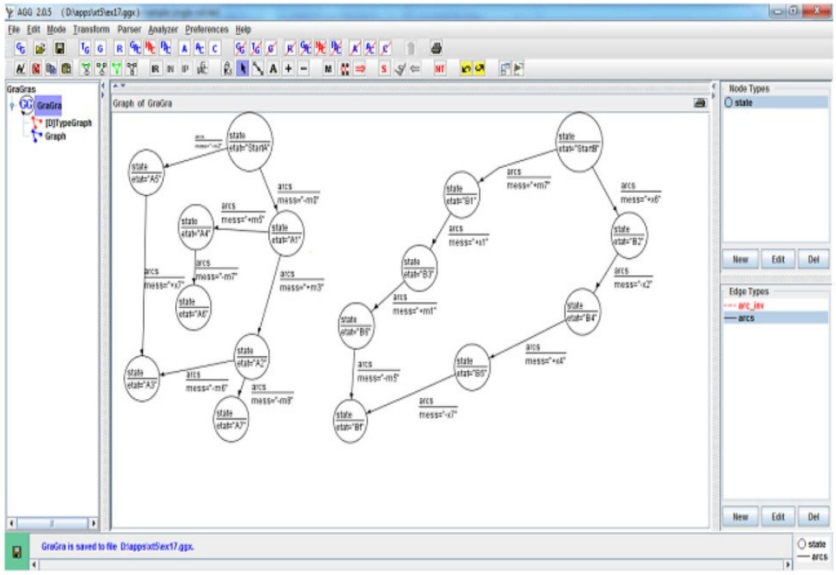


Figure 7. Screen Capture of Input BPs.

```
Load the two BPs...
BPs Loaded.................
Geting arcs types
Event arcs Type :arcs
Invocation Arcs type :arc_inv
Geting node type and Start nodes STA, STB
Start Node of BP1  libeled by   :"StartA"
Start Node of BP2  libeled by   :"StartB"
found exchanged message-->Rule Regle:1 Added between nodes : "B5"<--->"A5"
found exchanged message-->Rule Regle:2 Added between nodes : "B6"<--->"A1"
found exchanged message-->Rule Regle:3 Added between nodes : "A4"<--->"StartB"
---->Run rule Regle0
---->Run rule Regle1
---->Run rule Regle2
The two BPs have    3   invocation messages exchanged
===>    CHECKING DEADLOCK BETWEEN MESSAGES  :"x7"  <--->  "m5"
           DEAD LOCK FREE BETWEEN MESSAGES:"x7"  <--->  "m5"
===>    CHECKING DEADLOCK BETWEEN MESSAGES  :"x7"  <--->  "m7"
           DEAD LOCK FREE BETWEEN MESSAGES:"x7"  <--->  "m7"
===>    CHECKING DEADLOCK BETWEEN MESSAGES  :"m5"  <--->  "m7"
           DEAD LOCK FREE BETWEEN MESSAGES:"m5"  <--->  "m7"
         THERE IS DEADLOCK BETWEEN THE TWO BPs   ==> BEHAVIRAL INCOMPATIBILITY
SAVING THE RESULT GRAPH IN FILE:       D:/apps/Xt5/ex30.ggx
```

Figure 8. Output Execution.

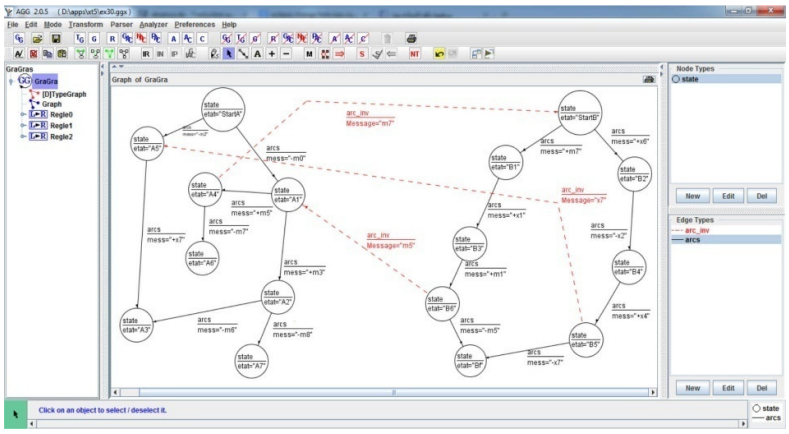


Figure 9. Screen Capture of Resulting Composite Service BP.

8. CONCLUSION AND FUTURE WORK DIRECTIONS

Dynamic services composition is a big challenge facing the success of SOA approach for which several tools have been proposed in literature. Among these tools, we find that formal based methods are the most promising. The choice of formal methods for specifying and dynamically checking service composition is justified by the need to have mathematical based tools, which guarantees the success of these operations. In this context, this paper, explored the possibility of using graph transformation as a tool for service composition checking. Services are modelled by their BPs; a formalism that specifies the external and observable behaviour of services, which is vital in the process of composition. The approach realises the checking composition by an automatic generation of production rules that controls the generation of composite service BP. We have proposed (1) an extension of BP for modelling composite service behaviour (2) a formalisation of necessary and sufficient conditions to check the composability of services (3) and an algorithm for checking services composition that we have implemented with the AGG API. As future work we expect (1) experiment the algorithm on real cases to optimise its complexity (2) extend the BP model to support the specification of service interfaces in order to describe service composition in a more realistic way (3) the use of model transformation tools to translate service BP model to a textual formalism specification such as Lotos.

REFERENCES

[1] Bentahar, J.; Yahyaoui, H.; Kova, M. ; Maamar, Z. 'Symbolic model checking composite Web services using operational and control behaviors' Expert Systems with Applications. Vol. 40 (2013) pp 508-522.

[2] Papazoglou, M.P., and Georgakopoulos, D. 'Service-Oriented Computing. Communications of the ACM', 46(10):25-28

[3] Benatallah, B., Casati, F., and Toumani, F. Web 'Service Conversation Modeling ACornerstone for E-Business Automation' IEEE Internet Computing 2004.

[4] Ehrig, H., Ehrig, K., Prange, U., and Taentzer, G. 'Fundamentals of Algebraic Graph Transformation' (Monographs in Theoretical Computer Science. An EATCS Series) Springer Verlag.

[5] Elboussasidi, G. 'Développement logiciel par transformation de modèles', Thèse de doctorat Université de Montréal 2009.

[6] Habel, A., Heckel, R., and Taentzer, G. 'Graph grammars with negative application conditions', Fundamenta Informaticae vol 26 (1995) pp 287-313. 1996.

[7] Barkaoui, K., Eslamichalandar, M., Kaabachi, M. 'A Structural Verification of Web Services Composition Compatibility.' In J. Barjis, M.M. Narasipuram, G. Rabadi, J. Ralyté, and P. Plebani (Eds.) CAiSE 2010 Workshop EOMAS'10, Hammamet, Tunisia, pp. 30-41.

[8] Foster, H., Uchitel, S., Magee, J. and Kramer, J. 'Model-based Verification of Web Service Compositions' Proceedings 18th IEEE International Conference on Automated Software Engineering.

[9] Bultan, F. and Su, T. 'Analysis of interacting BPEL web services' the 13th international conference onWorld Wide Web. New York, NY,USA:ACM Press 2004.

[10] Papapanagiotou, P., Fleuriot, J. 'Formal verification of web services composition using linear logic and pi-calculus'. In ninth IEEE European Conference on Web services (ECOWS), pp 31-38. IEEE (Septembre 2011).

[11] Hamadi, R. and Benatallah, B 'A Petri Net-Based Model for Web Service Composition' fourfteenth Australian Database Conference.

[12] Ding, W., Tian, Z., Wang, J., Zhu, J., Liang, H., Zhang, L., 'Conflicts Analysis for InterEnterprise Business Process Model' Systemics, Cybernetics and informatics Volume 1, Number 3.

[13] Benatallah, B., Casati, F., Toumani, F., and Hamadi, R. 'Conceptual Modelling of Web Service Conversation' 15th International Conference Advanced Information Systems Engineering (Caise'03) Klagenfurt Austria.

[14] Berardi, B., Calvanese, D., De Giacomo, C., Lenzerini, M., and Mecella ,M. 'Automatic composition of e-services that export their behavior'. In International Conference ServiceOriented Computing 2003.

[15] Benatallah, B., Casati, F., Grigori, G., Nezhad, H. R. M., and Toumani, F. 'Developing adapters for web services integration' in International Conference Advanced Information Systems Engineering (CAiSE05), 2005, pp. 415-429. 2005.

[16] Nezhad, H. R. M., Benatallah, B., Casati, F., and Toumani, F. 'Web services interoperability specifications, IEEE Internet Computing, vol. 39, no. 5, pp. 24-32, 2006.

[17] Dörr, H.: 'Efficient Graph Rewriting and its Implementation', volume 922 of Lecture Notes in Computer Science Springer-Verlag, 1995.

[18] Dodds, M., Plump, D.: Extending C for checking shape safety. In Proceedings Graph Transformation for Verification and Concurrency, Electronic Notes in Theoretical Computer Science Elsevier, 2005.

[19] Dodds, M., and Plump, D. (2006) 'Graph Transformation in Constant Time' Third International Conference Graph Tranformation Natal, Rio Grande do Norte, Brazil.

[20] Taentzer, G. 'AGG: A Graph Transformation Environment for Modeling and Validation of Software.' In J. Pfaltz, M. Nagl, and B. Boehlen, editors, Application of Graph Transformations with Industrial Relevance (AGTIVE'03), volume 3062 of LNCS, pages 446 - 456. Springer, 2004.

AUTHORS

Redouane Nouara

Bachelor of Science (B.Sc.), Computer Science, University of Constantine, Algeria, 1993.
Magister, Computer Science,University of Tebessae, Algeria, 2008.
Associated professor 2010.
Research:
- Formal Tool.
- Graph Transformation
- MDA Approach
- Model Transformation.
- System Checking.

Allaoua Chaoui

Bachelor of Science (B.Sc.), Computer Science, University of Constantine, Algeria, June 1986
Master of Science (M.Sc.), Computer Science,University of Constantine, Algeria, 1992.
Doctor of Philosophy (Ph.D.), Computer
Science, University of Constantine, Algeria, 1998. Research:
- Distributed Computing,
- Software Engineering,
- Theory of Computation
- Formal Tool.
- MDA Approach
- Model Transformation.