SERVICES COMPOSITION CHEACKING APPROACH BASED ON GRAPH TRANSFORMATION

Redouane NOUARA and Allaoua CHAOUI
Misc lab. Mentouri University Constantine Algeria
Email :nr25100@hotmail.com; a_chaoui2001@yahoo.com

Abstract

The Dynamic services composition is a major challenge whose realisation will determine the success of the service-oriented architecture (SOA) approach. In this trend several approaches have been proposed, they are based either on semantic aspects or planning tools that include a formal mathematical basis. In this paper, we propose a new approach to service composition checking based on the uses of graph transformation (GT). Services are modelled by their Business Protocols, a special deterministic finite-state automata (DFA) that describe service’s external behaviour. The cheacking of service composition approach use the Double Pushout (DPO) an algebraic approach of graph transformation. The idea is to try to generate dynamically the transformation rule \( p \) whose application generates the composite service graph. We propose an algorithm for computing de rule \( p \) in a constant time.

Keywords

Web service, graph transformation, automatic service composition.

1. INTRODUCTION

Service Oriented Architecture (SOA) is an ideal solution to the problems of distributed applications development characterized by heterogeneity and lowcoupling of these components. Despite the great step made in this field by standardizing the protocols for service description (WSDL), service communication (SOAP) and service management (WS-*), all researchers and manufacturers are convinced that the success of the SOA service-oriented approach inevitably successful automation of dynamic composition of services. In which the selection of composed services is made on the fly. As alternatives, several approaches have been proposed based generally 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. The service behaviour is associated with its static interface description defined by standard Web service languages like WSDL. The specification of external and observable behaviour of service 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[6][2]. The crucial problem that this posed is whether a service selected based on certain criteria (functional or not functional) may be successfully composed with the desired service. Even if they are compatible in terms of interaction interfaces. Verifying composability of services plays an important role in the operation of dynamic composition. If the non-formal approaches to composition (AI planning) have proven their limits at the expense of purely formal approaches. With their mathematical basis, these approaches are ideal solutions for the automation and verification of services composition.

The article is structured as follows: In section 2 we introduce the concepts and definitions of the graph transformation formalism based on an algebraic approach. A state of the art on the use of graph transformation as a tool in the service composition literature is presented in Section 3. Section 4 introduces the Business Process (BP) used as a formalism for modelling behavioural services in our approach. In section 5 we details the proposed services cheacking composition approach based on GT and presented our algorithm for generating automatically the transformation rule \( p \) in section 6 and 7. We compute the complexity of the algorithm in section 8 and finally we closed this paper by a conclusion and future works in Section 6.

2. GRAPH TRANSFORMATION

Graphs offer a very rich mathematical formalism for modelling because they are a natural means for expressing complex systems situations on an intuitive level[8]. What justifies their wide uses in the specification data, diagrams, flow control, for the entities and relationships for UML diagrams[7]. The basic idea of graph transformation is the change of a source graph \( G \) in another target graph \( H \) based on a rule’s application, similar to use
of Chomsky grammars in words generation. It is used in several areas of computing for model transformation such as modelling and specification of visual processing model according to the MDA (Model Driven Architecture) approach[8]. In what follows we present some basic definitions of the graph transformation used in our approach.

Fig 1: Double Push Out Transformation

2.1. Graphs and Graphs morphisms

A labelled graph \( G = (V, E, s, t, l_s, l_t) \) 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 respectively define the sources and target of edges. \( l_s : V \to L_V \) and \( l_t : L_E \to L \) are labelling functions that attribute respectively a node’s label from the set \( L_V \) (resp edges label from \( L_E \)). Let \( G_1 \) and \( G_2 \) be two graphs \( G_i \) defined by \( G_i = (V_i, E_i, s_i, t_i) \) with \( i = 1, 2 \). A graph morphism \( f : G_1 \to G_2 \), with \( f = (f_V, f_E) \) consists in two functions \( f_V : V_1 \to V_2 \) and \( f_E : E_1 \to E_2 \) that preserves source and target functions defined by[8]: 
\[
f_s \circ s_1 = s_2 \circ f_t \quad \text{and} \quad f_t \circ t_1 = t_2 \circ f_s.
\]

A graph transformation is the operation of transforming a source graph \( G \) into a target graph \( H \) by application of a production rule \( p \):
\[
p = L \xrightarrow{l} K \xrightarrow{r} R
\]

\( L \) and \( R \) called respectively the left and right hands of the production and \( K \) the gluing graph with \( K = L \cap R \). \( l \) and \( r \) are respectively morphisms \( l : K \to \overline{L} \) and \( r : K \to \overline{R} \). The rule \( p \) is applicable only if there is a morphism \( m : L \to G \) and the resulting graph \( H \) is constructed by adding \( L/K \) to the graph \( G \) and removing \( R/K \).[8]

3. RELATED WORKS

The use of graph transformation in the formalization of distributed architectures and the verification of services composition has been very little attention in the literature, in our known. Foster[17] propose a service composition approach verification 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, and compiled into the Finite State Process notation (FSP) to reason about the concurrent programs. A major works done on service verification are based on model checking approaches, WSAT[18] is a framework for analyzing the interactions among composite Web services modelled as conversations between services. Hamadi[15] propose an approach that use Petri nets for modelling services, the service composition is done by a merging Petri nets procedure and to model the control flow of the composite service, it is based on a merging of petri nets based on purely syntactic aspects. This approach does not use it explicitly graph transformation formalism of composition but it use an algebra that specifies different concurrent execution forms.

4. SERVICE MODELLING FORMALISM

We adopt in this work the model proposed by enatallah and all[1], [2] and Berardi and all[9] for modelling services. This model captures the conversations that a service supports. Conversation is defined as the ordered set of messages exchanged between the service and its user. The model uses deterministic finite state automata (DFA). In which the states represent the different phases through which the service passes during its lifecycle, transitions models the events and/or actions that occurred during the service interaction. These transitions are triggered by messages exchanged by the service which corresponds to an invocation or a response to the latter or by internal events serving as the expiration of a waiting period. The set of messages exchanged is called conversation. The model has a single initial state and several final states, the labels of transitions are associated with polarities (+,-) that specifies the origin of messages. Polarity + (respectively -) indicates that the message is received (respectively sent) by the service.

To use the graph transformation approach, we formalize the external behaviour of services, in this article, with a language graph notation as mentioned in Ehrig[8] instead of the automaton notation used in [1],[2]. In order to integrate the BP specificities in a graph model notation, we extended the graph definition by the initial and final states of the automaton. Let \( A \) be a BP of a service, formally we use the following definition of \( A \):
\[
A = (V, E, s, t, l_v, l_e, v_0, F)
\]

With \( V \) and \( E \) respectively describe sets of states and edges, \( s \) and \( t \) the functions start and target of edges defined by \( s, t : E \to V \). The sets \( l_v \) and \( l_e \) represent respectively the states and edges labels (with their respective polarities). The initial state is designed by \( v_0 \) with \( v_0 \in V \) and \( F \) the set of
final states (with \( F \subseteq V \)). Example: Figure 2 describes the modelling of an e-commerce protocol service that manage the order of some gods, with the start as initially state and as the set \{\text{Cancel, delivery}\} as final states. Labels \text{login}(\text{+}), \text{select}(\text{+}), \text{confirm}(-), \text{payment}(\text{+})\) are messages exchanged between the service and client while the message depending on the polarity sign, \text{Cancel}(-)\) is an event generated by the service and sent to the client after the timeout of payment by the customer.

![Sample Business Process](image)

**Fig 2 : Sample Business Process**

### 5. SERVICES COMPOSITION

**VERIFICATION APPROACH**

The verification of service composition approach we propose is based on the use of graph transformation in which either two services composing \(S_1\) and \(S_2\) are modelled by their respective Business protocols and the result operation is the service composite \(Sc\). The idea is to use graph transformation approach with \text{DPO} as formal tools to merge the graphs \(S_1\) and \(S_2\) to give the graph \(Sc\) by generating automatically a transformation rule \(p\). The first service \(S_1\) represent the source graph (i.e, the graph \(G\) in definition of \text{DPO} transformation) and the service \(S_2\) constitute the right part of the transformation rule (i.e the graph \(R\)). Hence the generation of the rule \(p\) is equivalent to the generation of the left hand graph \(L\) and the gluing graph \(K\) as depicted in Figure 3. The transformation is characterized by the fact that it makes no deletion of vertices or edges on the source graph \(S_1\), it only add the graph \(S_2\). Therefore the difference between the left side of the rule (the graph \(L\)) and the gluing graph \(K\) must be empty, i.e \(L - K = \emptyset\) which means that \(L\) is identical to \(K\) and therefore to have the rule \(p\) we have only to generate the gluing graph \(K\) with two morphisms: the first \(m : K \rightarrow S_1\) and the second \(r : K \rightarrow S_2\).

![Approach Of checking services Composition](image)

**Fig 3 : Approach Of checking services Composition**

As the transformation rule that controls the composition of the two services is in the form 
\[ p = L \xrightarrow{id} K \xrightarrow{r} R \]
With the \(id : K \rightarrow K\) identity isomorphism. Therefore the rule can be formulated by:
\[ p = K \xrightarrow{r} S_2 \]

### 6. GLUING GRAPH GENERATION

In our case, we have (1) to automatically generate the rule \(p\), specially the gluing graph \(K\) with the morphism \(r : K \rightarrow S_2\) and (2) to identify a morphism \(m : K \rightarrow S_1\) which represents the matching of the left side of the rule in the graph \(S_1\).

To our knowledge, the problem of automatic generation of rules in graph transformation approach have not been treated and much work has dealt with more precisely the problem of identifying the set of graph morphisms between two graphs. Most of these methods are based on two approaches (1) coloring graphs[10] and (2) calculating morphism by iterative algorithms[11]. In the first method different colors are assigned to adjacent vertices and generate the morphism from the color classes, where the second method, calculates a set of morphisms for special type of graphs defined by rooted graphs, they have a special vertex called root which is unique and from which all other vertices can be reached.

As solution we propose an approach based on the finding of maximum common subgraph (MCS) between two graphs. Several solutions have been proposed for this problem, Bunk[12] presented a comparative study of solutions proposed for calculating the MCS, from which appear two major approaches (algorithms) first those based on the algorithm McGregor[13] who uses a back tracking search approach on a tree, the seconds are based on Durand[14] who propose an iterative approach for building the MCS by adding iteratively nodes and edges, this approach is well suited to rooted graphs like BP because Dodds[11] proved that the calculation of the set of morphisms between two graphs is not NP-complete and can be performed in constant time. The algorithm requires time in the worst case is bound by:

\[ \sum_{i=0}^{n} b^i \]

Where \(n\) is the number of nodes and \(b\) the bound outdegree of each node. Based on this observation we propose an algorithm that generates a common subgraphs, which is not necessarily the largest. In the event that generated the graph is empty, this
meant that both services have no state in common with each other and therefore cannot be dialed.

7. FINDING THE GLUING GRAPH
ALGORITHM

The algorithm uses a weight function poid that return for a given node the number of nodes that are reachable from this node. In the beginning all states of the service that has the greatest number are sorted according to their weight. The first step consist in finding a state v, that it’s into both graphs (v₁ ∈ V₁ ∩ V₂) in this case only we can conclude the existence of the gluing graph K. The next step is to enlarge the graph K to it’s maximum by adding at each iteration:

- The nodes that are in both graphs and whose edges have their terminations already in the graph K (f/v ∈ V₁ ∩ V₂ and tᵣ(v) = tᵣ(v) ∈ K)
- The edges which are in the two graphs and where the sources nodes already in the graph K (e/e ∈ E₁ ∩ E₂ and sᵣ(v) = sᵣ(v) ∈ K)

Algorithm 1 Function Poid return the number of nodes reachable from a node

Require: v a graph node.
Ensure: the liste reachable nodes from v
1: if Terminal(v) then
2: \( \text{Poid} \leftarrow \{v\} \)
3: else
4: \( k \leftarrow \text{nodecount}(v) \)
5: for \( j = 1 \) to \( k \) do
6: \( \text{Poid} \leftarrow \text{Poid} ∪ \text{poid}(\text{nextnode}(v,j)) \)
7: end for
8: end if
9: return Poid

8. Algorithm complexity

The algorithm consists of three major parts:
- The first calculates the weight vector of all nodes in the largest graph (assumed equal to \( n \)), this is achieved by calling the function Poid \( n \) times. In the worst case this is done in \( n^2 \) time.
- The second part search \( v₀ \), the nearest node to the start node (root) in the biggest graph, where \( v₀ \) is in both graphs (\( v₀ \in V₁ ∩ V₂ \)). This is done done in the worst case \( n \) times.
- If there is a node found in second part, the third part build the gluing graph K. the

worst case is where the graph is complete (every pair of nodes is connected by an edge). In this case we have the size \( |Z₀| = 1 \) and \( |Z| = n \) with \( 1 < i < n \). By execute the loop \( n \) iterations we have:

\[ 2 + 2n + \cdots + 2n = 2n^2 - 2n + 2 \]

So the time required for the execution of the algorithm is limited by:

\[ n^2 + n + 2n^2 - 2n + 2 = 3n^2 - n + 2 \]

Algorithm 2 Gluing Graph generation Algorithm

Require: \( S = \{V₁, E₁, s₁, t₁\} \) Services.
Ensure: \( K_{col} \) The gluing Graph.
1: \( V₁ = \{s₁, \ldots, sᵣ\} \) and \( E₁ = \{e₁, \ldots, eᵣ\} \)
2: \( V₂ = \{s₁, \ldots, sᵣ\} \) and \( E₂ = \{e₁, \ldots, eᵣ\} \)
3: \( \text{frdos} \leftarrow \text{frdos} \)
4: if \( n ≥ k \) then
5: for \( i = 0 \) to \( n \) do
6: \( \text{Pd}(i) \leftarrow \text{Poid}(vᵢ) \)
7: end for
8: else
9: for \( i = 0 \) to \( k \) do
10: \( \text{Pd}(i) \leftarrow \text{Poid}(vᵢ) \)
11: end for
12: end if
13: \( i \leftarrow 0 \)
14: while (not trouve) and (\( i ≤ k \)) do
15: \( x \leftarrow \text{Pd}(i) \)
16: if \( (vᵢ \in V₁ ∩ V₂) \) then
17: trouve \( \leftarrow \text{true} \)
18: else
19: \( i \leftarrow i + 1 \)
20: end if
21: end while
22: if trouve then
23: \( Z₀ \leftarrow \{vᵢ\} \)
24: \( Y₀ \leftarrow \{e/e = ss(vᵢ) = st(vᵢ)\} \)
25: \( i \leftarrow 0 \)
26: repeat
27: Find Nodes that have \( Zᵢ-1 \) as a terminate nodes
28: \( eᵢ \leftarrow \{e ∈ E₁ ∩ E₂ \text{ and } t_e(v) = t_e(v) \in Zᵢ-1\} \)
29: Find edges that have start in \( Zᵢ-1 \)
30: \( Yᵢ \leftarrow \{e ∈ E₁ ∩ E₂ \text{ and } s_e(v) = s_e(v) \in Zᵢ-1\} \)
31: \( i \leftarrow i + 1 \)
32: until (\( Zᵢ \leftarrow Zᵢ \) and \( Yᵢ \leftarrow Yᵢ \))
33: \( K_{col} = (Zᵢ, Yᵢ, sᵣ, tᵣ) \)
34: else
35: \( K_{col} = \emptyset \)
36: the two services can’t be composed.
37: end if
38: END Algorithm
9. CONCLUSION

The need for formal methods for specifying the service composition is justified by the need to have a mathematical basis, only that can guarantee the success of the operation. In this context, we have in this article explored the possibility of using the graph transformation as a formalism for the service composition, especially the DPO approach. Services are modelled by their BP (AFD) a formalism that offers the possibility of specifying the external behaviour of services that is vital in the process of composition. This approach realizes the checking of the composition operation by an automatic generation of the production rule that control the generation of composite service graph. Considering the particularity of BP we identified a set of patterns for composite service.

10. REFERENCES


