# TRANSFORMING BPEL SERVICE COMPOSITION INTO A SERVICE COMPOSITION DIRECTED GRAPH FOR BETTER COMPOSITION PLAN MANAGEMENT

Lev Belava
Department of Computer Science
AGH University of Science and Technology
Mickiewicza 30, Cracow, Poland
E-mail: Lev.Belava@gmail.com

**KEYWORDS**
SOA, Web Services, BPEL, Service Composition.

**ABSTRACT**

The advantages of web service composition lead to the development of various notations and languages for modeling service composition processes and introduce them to process engines for execution. One of the most popular languages for service compositions is Business Process Execution Language (*BPEL*). BPEL process editing software is widely available and able to perform basic things such as manual visual process edition. But for the purpose of automatic or semi-automatic BPEL process edition there is a need to develop some kind of an easy machine-processable form of BPEL composition plans. This paper presents a Service Composition Directed Graph – a graph-based data structure dedicated to the representation of service compositions. Also, this paper proposes an algorithm that transforms BPEL compositions into Service Composition Directed Graphs.

## 1. INTRODUCTION

The composition of SOA services seems to be an interesting and promising approach to developing new software that can utilize specialized networked functionalities provided by SOA services. The practice of combining different primitive services into a single complex one allows engineers to have the desired level of abstraction and problem granularity and implies greater flexibility than classical out-of-box software systems (Belava 2009).

Web Services are one of the many possible realizations of the SOA paradigm. W3C defines a Web Service as a “software system designed to support interoperable machine-to-machine interaction over a network” (Belava 2010). In addition, this definition explicitly claims that a service must have “an interface described in a machine-processable format (specifically WSDL)” and other systems should interact with a service “in a manner prescribed by its description using SOAP messages” (Belava 2010).

An interface is a machine-readable document that specifies mechanisms of message exchange called WSD (*Web Service Description*), written in the WSDL (*Web Service Description Language*) language. WSD defines message formats, transport protocols, serialization mechanisms etc. It also specifies one or more network addresses where provider agents can be found (Belava 2010).

BPEL is the most common orchestration (not choreography) language for Web Services execution and abstract plans (Cetnarowicz et al. 2010). It defines a model and a grammar for describing the behavior of a business process based on the interactions between the process and its partners (Heravi and Razzazi 2007). All external resources and partners of BPEL process are represented as WSDL services. The semantics of BPEL also known as BPEL4WS depends on WSDL, XML Schema and XPath.

The main contributions of this paper are: defining a new graph-based data structure for representing web services composition plans and proposing an algorithm that automatically transforms BPEL service composition plans into a Service Composition Directed Graph – a special type of directed graph that supports not only basic service calls but also nested compositions, parallel flows and control flow statements. To validate the proposed approach an experimental framework was implemented, and the results of its work are presented in this article.

The paper is structured as follows. Section 1.1 highlights the current related work on the automatic transformation of BPEL service compositions into various data structures. Section 2 introduces a Service Composition Directed Graph. Section 3 describes an algorithm that transforms a BPEL into a Service Composition Directed Graph. Section 4 presents the implemented software and example results. Section 5 concludes the paper and outlines future work.

### 1. 1 Related Work

To date, few BPEL composition transformation algorithms have been proposed. In (Wombacher et al. 2004) BPEL Web Service compositions were transformed into deterministic finite state automata for the purpose of service discovery. In (Lallali et al. 2008) BPEL compositions were transformed into a special composition testing language called IF which enabled better service composition testing. In (Haiqiang et al.

Proceedings 25th European Conference on Modelling and Simulation ©ECMS Tadeusz Burczynski, Joanna Kolodziej Aleksander Byrski, Marco Carvalho (Editors)
ISBN: 978-0-9564944-2-9 / ISBN: 978-0-9564944-3-6 (CD)

2008) BPEL compositions were transformed into a special class of Petri nets called ServiceNet. In (Zheng et al. 2007), the BPEL semantics was modeled by Web Service Automata. In (Schumm et al. 2009) BPEL was transformed to BPMN (a similar, but not identical language) for modeling and visualization purposes. In (Moon et al. 2004) BPEL was transformed to BPMN language for similar purposes as in (Schumm et al. 2009).

The presented paper concentrates on automatic transformation of BPEL service compositions into a directed graph based data structure with the support of nested compositions, parallel flow and control flow statements.

### 1.2 Problem Statement

None of composition algorithms is perfect. For instance, sometimes there is a need to use a pre-defined static composition plan or part of it, or one particular algorithm is preferred to another, or perhaps the operator is inactive or has no time for decisions (Belava 2009). That is why service composition plans have to be both processable by execution engines and at the same time easily machine-processable by automated software that can introduce various changes into the plans.

BPEL is the most common web service composition language. It is XML-based and many well-known parsers are available for it. However even working with XML entities for the purpose of editing service compositions is not a productive decision because such kind of work demands a higher level of abstraction to operate on.

The presented work describes two things: a data structure called a Service Composition Directed Graph that provides a higher level of abstraction for service compositions and a transformation algorithm that can transform BPEL compositions into it.

## 2. SERVICE COMPOSITION DIRECTED GRAPH OVERVIEW

Service Composition Directed Graph is a tuple G=(N, L), and satisfies the following constraints:

1) N is a finite set
2) $L \subseteq (N \times N)$
3) $\forall\, x1=(n1,n2) \wedge x2=(n1,n2) \Rightarrow x1=x2$

N is a node set whose elements can be service nodes or control nodes.

The service node is a representation of a single web service function available on the network.

L is a set of directed connection arcs and represents data and control flow in a Service Composition Directed Graph.

The constraints of the composition directed graph are as follows:

1) the set of nodes is finite, so the directed graph is finite;
2) nodes are connected only by arcs;
3) only one arc could connect two nodes to avoid repeated connections;
4) Service Composition Directed Graph is an acyclic graph.

### 2.1 Service Node Overview

A service node is a primary construction block for every service composition plan. It:

1) is a representation of a single web service function;
2) has a description of its input and output data types;
3) has a unique ID;
4) can be connected with other nodes by arcs.

A service node is shown in Fig. 1.

### 2.2 Control Node Overview

Control nodes are in charge of data and control flow in a service composition plan. Also, they introduce decision making and concurrency into the composition process.

There are a number of properties that control nodes possess:

1) Control node could be one of 3 types: *IF, FLOW, WHILE*. They are shown on Fig. 2, Fig. 3 and Fig. 4.
2) Every control node has its own unique ID.
3) *IF* and *WHILE* nodes have a control condition which should be evaluated during composition process execution.
4) *FLOW* and *WHILE* nodes have nested composition directed graphs that describe service compositions that are executed in this control nodes.
5) Control nodes can be connected by arcs to other nodes in a graph.

### 2.3 Arc Overview

Arcs in a Service Composition Directed Graph satisfy the following conditions:

1) They explicitly define the direction of link between two connected nodes and
2) may contain supplementary information. For example after *IF* node there is a need to determine which outgoing arc is related to which result of evaluation of control condition and that information could be stored in arcs.

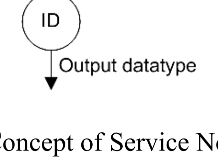


Figure 1: The Concept of Service Node in a Service Composition Directed Graph

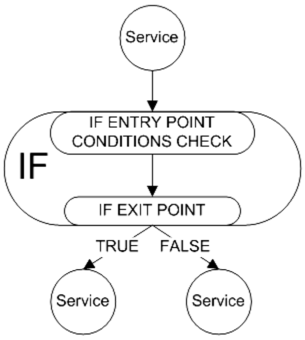


Figure 2: The Concept of IF Control Node in a Service Composition Directed Graph

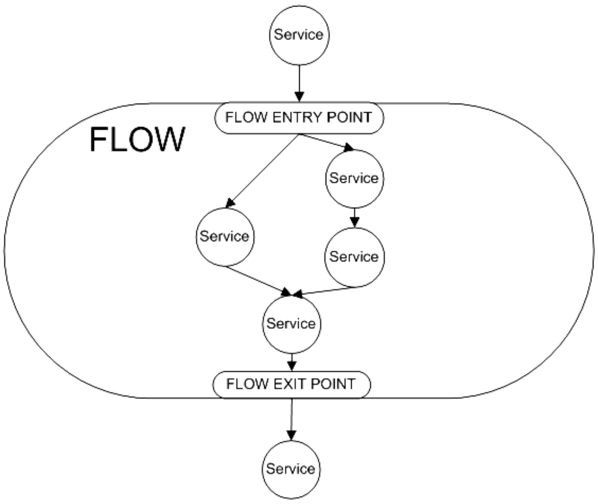


Figure 3: The Concept of FLOW Control Node in a Service Composition Directed Graph

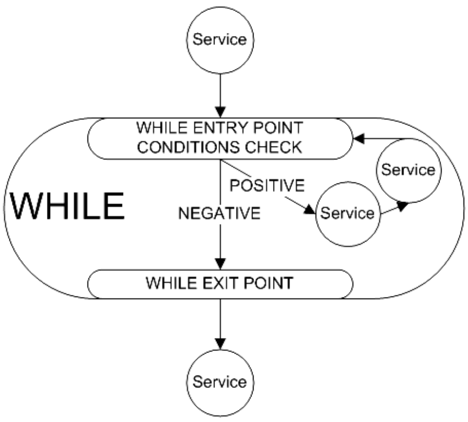


Figure 4: The Concept of WHILE Control Node in a Service Composition Directed Graph

## 3. BPEL SERVICE COMPOSITION TRANSFORMATION ALGORITHM

The transformation algorithm in Fig. 5 is basically a recursive XML parser that takes a BPEL composition and produces a Service Composition Directed Graph. No error-checking or any other form of information correction is provided.

In step 1 of the presented algorithm a root element of the BPEL service composition file is found. This root element is just the first <process> element in the XML file. Also, the fact that XML files can define namespaces should be taken into consideration.

In step 2 a root node is created. In this case it is just an atomic service node with a pre-defined name.

Step 3_1 takes the next element of sequence.

In step 3_2 the element from 3_1 is checked.

Step 3_2_a checks a Boolean result from 3_2.

In step 3_2_a_1 the parsing of the current element is ended and no action on a Service Composition Directed Graph is taken. The flow of control goes to 3_1 that starts processing the next element in the sequence.

Steps 3_3 – 3_7 perform a type check of the current element.

In steps 3_3_a, 3_4_a, 3_5_a and 3_6_a appropriate nodes are added to the Service Composition Directed Graph.

In steps 3_3_b, 3_4_b, 3_5_b and 3_6_b connection arcs are added to the Service Composition Directed Graph. The ID of the parent node is known from the last operation of node adding. Additional information to arcs (if any) is taken from parameters and added appropriately.

In steps 3_3_d and 3_3_e THEN and ELSE sequences of IF element are redirected for parsing. Also information of sequence type is redirected too, so later in steps 3_3_b, 3_4_b, 3_5_b and 3_6_b connection arcs will be enriched with it.

In the 3_4_c plus 3_4_c_1 and 3_5_d plus 3_5_d_1 pairs of steps a nested Service Composition Directed Graph is being created, added to the node by making a recursive parsing call.

```
GET root element from BPEL file
ADD root element to graph (first element(dummy))
GET SEQUENCE for BPEL root element
BEGIN sequence parsing
 3_1: GET next element of sequence
 3_2: CHECK if element is a SEQUENCE or IF or FLOW or WHILE or INVOKE
     3_2_a: IF FALSE - SKIP element
         3_2_a_1: GOTO 3_1
 3_3: IF element is an IF
     3_3_a: ADD IF node to graph
     3_3_b: CONNECT to parent by arc
     3_3_c: SET control statement
     3_3_d: PARSE THEN SEQUENCE (pass "THEN" parameter) - GOTO 3
     3_3_e: PARSE ELSE SEQUENCE (pass "ELSE" parameter) - GOTO 3
 3_4: IF element is a FLOW
     3_4_a: ADD FLOW node to graph
     3_4_b: CONNECT to parent by arc
     3_4_c: CREATE nested service composition directed graph
         3_4_c_1: PARSE FLOW SEQUENCE - GOTO 3
 3_5: IF element is a WHILE
     3_5_a: ADD WHILE node to graph
     3_5_b: CONNECT to parent by arc
     3_5_c: SET control statement
     3_5_d: CREATE nested service composition directed graph
         3_5_d_1: PARSE WHILE SEQUENCE - GOTO 3
 3_6: IF element is an INVOKE
     3_6_a: ADD SERVICE node to graph
     3_6_b: CONNECT to parent by arc
 3_7: IF element is a SEQUENCE
     3_7_a: PARSE SEQUENCE - GOTO 3
END recursive sequence parsing
```

Figure 5: A Pseudocode Notation of the BPEL Service Composition Transformation Algorithm

The BPEL language is very rich and defines many more elements and features than taken into account by the described algorithm. This is done on purpose because a Service Composition Directed Graph is not designed to transfer BPEL compositions into it as closely to source as possible, but to provide a tool and a concept for facilitating automated process composition and edition. Also, there is no barrier for the enrichment of Service Composition Directed Graph and the transformation algorithm with such BPEL features as partner links, ForEach statements, process related variables etc.

## 4. IMPLEMENTED SOFTWARE AND EXAMPLE OUTPUT

The proof of the concept implementation of BPEL service composition transformation algorithm is made in the form of a Java application. The software takes an XML file with a BPEL service composition in it and produces a Service Composition Directed Graph. Also it is possible to make a basic visualization of the produced graph.

The core of transformation is build upon a jdom DOM XML parser that is used to process raw XML data. JgraphT is used as a graph implementation library while JGraph is used for visualization purposes. The structure of Service Composition Directed Graph was defined as a jgrapht directed graph with the abstract class "*AbstractNode*" (Fig. 6) for all nodes and with the use of standard directed arcs. The control nodes were organized into a hierarchy with the abstract class "*AbstractControlNode*" (Fig. 6 and 7). The arcs where implemented in the class "" The manipulation API for the Service Composition Directed Graph was implemented in the class "*PlanGraph*" (Fig. 8).

### 4.1 Example Output

The visualization of the Service Composition Directed Graph is shown on Figs. 9, 10 and 11. This graph was obtained as a result of the transformation of a sophisticated BPEL composition in which *FLOW*, *WHILE*, *SEQUENCE*, *IF* and *INVOKE* BPEL elements were used.

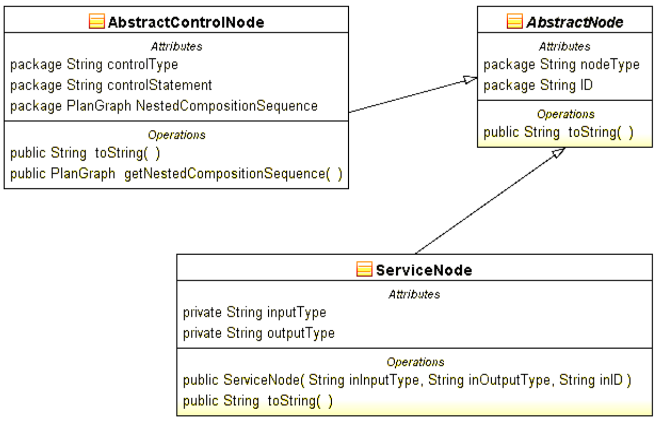


Figure 6: The Implemented Class Hierarchy for the Nodes in a Service Composition Directed Graph

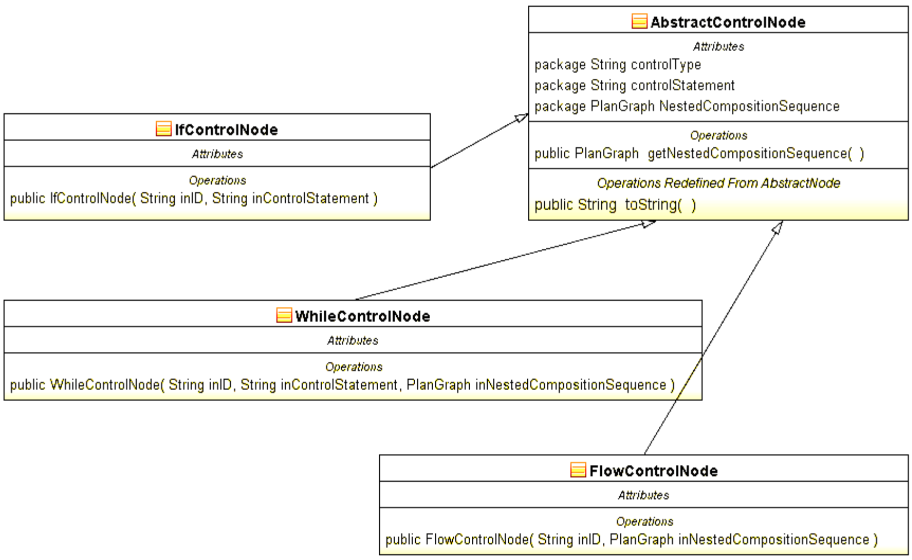


Figure 7: The Implemented Class Hierarchy for the Control Type Nodes in a Service Composition Directed Graph

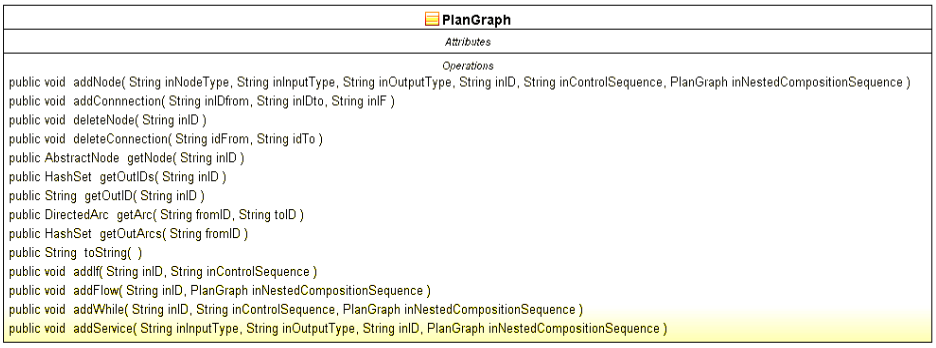


Figure 8: The Implemented API for Service Composition Directed Graph Manipulation

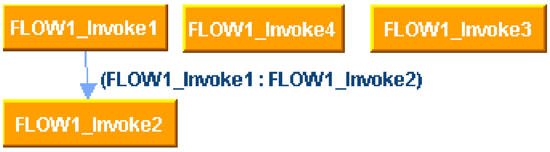


Figure 9: The FLOW1 Nested Service Composition Directed Graph

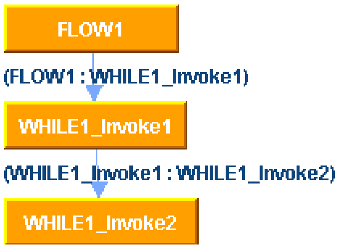


Figure 10: The WHILE1 Nested Service Composition Directed Graph

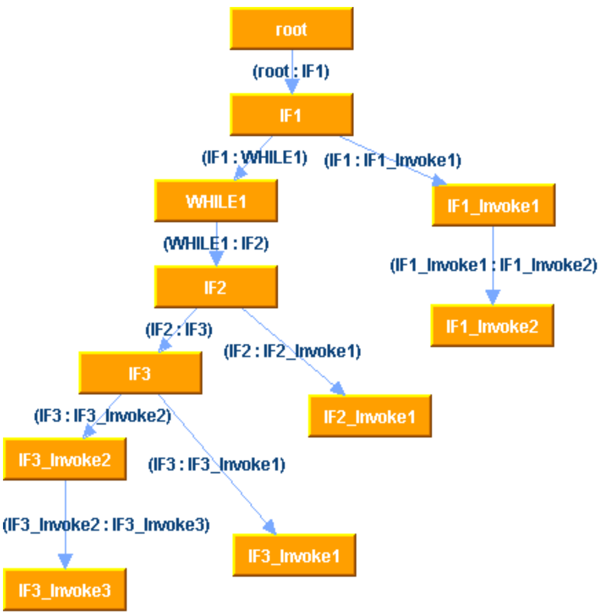


Figure 11: The Visualization of an Example Service Composition Directed Graph

## 5. CONCLUSION AND FUTURE WORK

This work proposes an approach to solving a problem of acquiring a reasonable, more abstract and machine-processable form of service composition representation. This is done by introducing a Service Composition Directed Graph data structure and presenting a transformation algorithm that produces such graphs from service compositions written in BPEL.

Despite the fact that the described algorithm and data structure does not completely cover all the BPEL features, it is applicable as is and can be easily extended with new node types, process variables and other elements.

Future work on the subject may be done in the fields of better BPEL features coverage, transforming other forms of service composition representation into Service Composition Directed Graphs which could in turn be transformed into BPEL executables and abstract plans.

Currently the algorithm is a part of experimental software that will enable complex machine processing of BPEL service compositions and will generate BPEL executables as a result.

## REFERENCES


Belava L. 2009. „Concept of Hybrid Service Composition in SOA Environment”. *Automatyka,* No.13-2, 189-197.

Belava L. 2010. “Concept of Hybrid Service Composition in SOA Environment with Agent Enrichment”. In *Proceedings of the 2010 Second Forum of Young Researchers* (Izhevsk, Russia, Apr.22). Publishing House of ISTU, Izhevsk, 117-122.

Cetnarowicz K.; Dyduch T.; Koźlak J.; Żabińska M.; Błaszczyk P.; Niedźwiecki M.; Rzecki K.; Belava L.; Wąchocki G.; Bech P.; Dziedzic J.; Ptaszek M. 2010. “A Concept of an SOA Agent System and its Application to Support an Integrated Rescue Action”. In *SOA Infrastructure Tools: Concepts and Methods 2010,* Ambroszkiewicz S.; Brzeziński J.; Cellary W.; Grzech A. and Zieliński K (Eds.). Poznań University of Economics Press, Poznań, 471-488.

Haiqiang D.; Haiying X. and Lifu W. 2008. “Transformation of BPEL Processes to Petri Nets”. In *Proceedings of 2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering* (Nanjing, China, Jun.17-19). IEEE Computer Society Press, 166-173.

Heravi B.R. and Razzazi M.R. 2007. “Utilising WS-BPEL Business Processes through ebXML BPSS”. In *Proceedings of the International Conference for Internet Technology and Secured Transactions* (London, UK, May 21-23). e-Centre for Infonomics, UK.

Lallali M.; Zaidi F. and Cavalli A. 2008. “Transforming BPEL into Intermediate Format Language for Web Services Composition Testing”. In *Proceedings of 4th International Conference on Next Generation Web Services Practices* (Seoul, South Korea, Oct.20-22). IEEE Computer Society, Washington, D.C., USA, 191-197.

Moon J.; Lee D.; Park C. and Cho H. 2004. “Transformation Algorithms between BPEL4WS and BPML for the Executable Business Process”. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises* (Taejeon, South Korea, Jun.14-16). IEEE, Piscataway, N.J., 135-140.

Schumm D.; Karastoyanova D.; Leymann F. and Nitzsche J. 2009. “On Visualizing and Modelling BPEL with BPMN”. In *Proceedings of the 2009 Workshops at the Grid and Pervasive Computing Conference* (Geneva, Switzerland, May 4-8).IEEE Computer Society, Washington, D.C., USA, 80-87.

Wombacher A.; Fankhauser P. and Neuhold E. 2004. “Transforming BPEL into annotated Deterministic Finite State Automata for Service Discovery”. In *Proceedings of the IEEE International Conference on Web Services* (San Diego, CA, US, Jul.6-9). IEEE Computer Society, Los Alamitos, C.A., 316-323.

Zheng Y.; Zhou J. and Krause P. 2007. “An Automatic Test Case Generation Framework for Web Services” . *Journal of Software* 2, No.3 (Sep), 64-77.

**LEV BELAVA** was born in Russia and graduated from the AGH University of Science and Technology in Poland, where he studied Computer Science and obtained his M.Sc. degree in 2007. He is now enrolled in a Ph.D. Computer Science program at the same university. His areas of interest are SOA, Service Composition. His email address is Lev.Belava@gmail.com