A Survey on Splitting BPEL Processes to BPEL4Chor Choreography for improving the Business Process

Gowri D Choudhary, Prof. G B Jethava

Abstract---Many Companies applying BPEL to improve their business process by doing process modification. This paper illustrates how to split from a BPEL process to BPEL4Chor choreography. First, the main BPEL process given is split into fragment BPEL processes, in a way that the operational semantic of the main BPEL process is preserved in the collective behavior of the fragmented BPEL processes. The dataflow dependencies of the given BPEL process are analyzed and reflected in the fragmented BPEL processes. Based on the results of the splitting algorithm, BPEL4Chor choreography is generated: The fragmented BPEL processes are converted into participants in the generated BPEL4Chor choreography.

Index Terms---BPEL processes; WSDL; Process Fragmentation; and BPEL4Chor.

1 INTRODUCTION

NOWADAYS globally integrated enterprises are demanding more and more agility in the business. They pursue the ways to reinvent the business process rapidly, such as business process reengineering and continuous improvement process (CIP). The enterprises that embrace CIP improve their business process via modification of the noncompetitive part. If improvement goal in the non-competitive business cannot be reached, the out-sourcing or off-shore of the business process will usually be carried out in order to keep the company's portfolio profitable. To specify business process behavior based on Web services, the Business Process Execution Language (BPEL) has been introduced into industry in recent years. Companies applying BPEL can improve their business process by doing process modification. In the case of out-sourcing, the non-competitive part will be cut-out. The part can be regarded as a cut-out sub-process, which should be run by the third party companies. This cut-out sub-process is called "process fragment"[4]. The number of the process fragments depends on how the original process is "cut". The challenge is how to do the process fragmentation so that the collective behavior of the process fragments preserve the operational semantic of the original process.

An approach is proposed in [8] to decompose the process. In that approach, a BPEL process is firstly transformed into an intermediate form i.e. BPEL-D process in [1] which the data flow is represented by explicit data-links. Then the control link and data link are split in the same way that sending block and receiving block are created and the control (true or false) and data (value) are passed by messaging between the two blocks. At the end, the result is a BPEL process per participant, the corresponding WSDL definition per BPEL process, and a global wiring file. Although the BPEL-D process presents the data flow explicitly and can easily be split, it is not sufficient to split the data dependency in a BPEL process while keeping the operational semantic of that original process, due to the parallelism and Death-Path-Elimination (DPE) in BPEL process.

Therefore, a more BPEL compliant approach for splitting data dependency of a BPEL process is introduced in [4]. The mechanism of splitting data dependency in that approach differs from the explicit data-links in BPEL-D in a way that the data dependencies across the BPEL process fragments are maintained in an implicit manner. BPEL4Chor provides the interconnected interface behavior descriptions by utilizing the Abstract Process Profile for Observable Behavior of BPEL and by adding an interconnection layer on top of the abstract BPEL process.

- *Gowri D Choudhary is currently pursuing masters degree program in Computer Science & Engineering in Parul Institute of Engineering & Technology, India, E-mail: rgk.choudhary@gmail.com.*
- *G.B Jethava is currently professor in Information Technology Department in Parul Institute of Engineering & Technology, India, E-mail: author_name@mail.com*

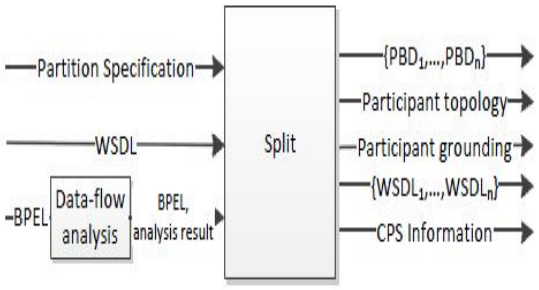

Figure 1.1: In- and Output of Splitting

The work in this paper is based on the above mentioned approach. Nevertheless, we lack in [4] a specific definition or description of the out-coming wiring file after the process fragmentation. Thus, we choose BPEL4Chor as the out-coming wiring specification. BPEL4Chor is a BPEL extension for defining choreographies and is suitable for the global wiring information. This paper aims to demonstrate how one can go from a BPEL process to BPEL4Chor choreography. In other words, we show how to accept a BPEL process, its associated WSDL definition and the partition specification as input, how to split the BPEL process into process fragments, and eventually how to output the corresponding BPEL4Chor artifacts. Figure 1.1 illustrates the input and output of the splitting process that is the main focus in this paper.

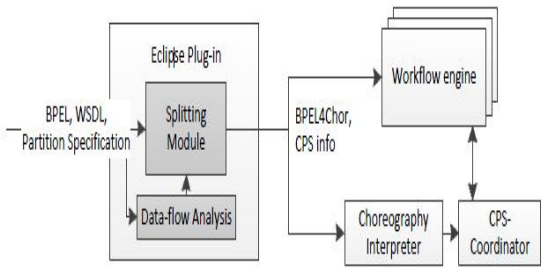

Figure 1.2: Architecture of the Process Fragmentation.

Architecturally, the split module (Figure 1.2) functions as an Eclipse Plug-in in the Eclipse BPEL designer. It takes a BPEL, an associated WSDL, the provided partition specification along with the data-flow analysis on that BPEL process as input and outputs the BPEL4Chor as well as split loop/scope information. The outputs can be consumed by Choreography interpreter and CPS-Coordinator in the workflow engine.

2 PROCESS FRAGMENTATIONS

Process fragmentation is the beginning of the splitting procedure, therefore, it is the part where the input of the split procedure is taken care of. After the input, a main BPEL process is created, and it is about to be fragmented into smaller ones upon the partition specification given.

2.1 Main Process Specification

The Figure 3.1 shows that one of the input for the splitting procedure is a BPEL process, which one can call main BPEL process or original BPEL process, since it will be split into multiple smaller fragment processes. Input that is associated to the BPEL process is the WSDL definition. For the split, it provides the message type referred by variable in BPEL process, the PortType referred by in-bound and out-bound activity, and the PartnerLinkType referred by PartnerLink

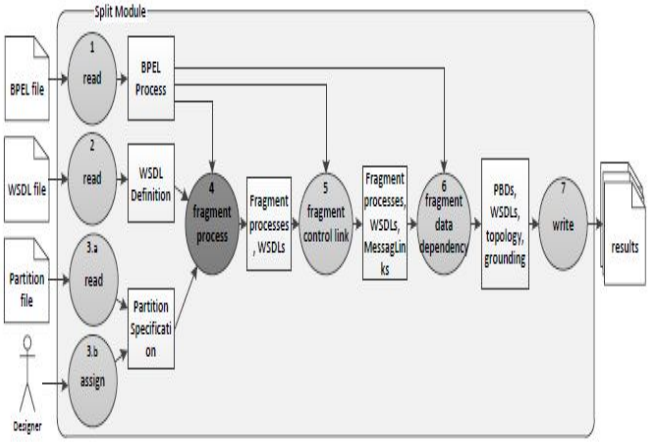

Figure 2.1: The Overview of the Split Module

in the BPEL process. The deployment information of the main process is absent since it is not deployed in this paper, it is split instead. Due to the complexity of supporting all BPEL activities, there is a need to set up a subset so that the task can be achieved in this paper.

2.2 Partition Specification

Partition specification is one part of the input for splitting procedure besides the BPEL and WSDL. It informs the splitting procedure which activity in the main process is assigned to which participant. The participants together constitute the partition of the BPEL process. The term 'participant' indicates a fragment of the main process, and

it has one or more activities in the main process. The main idea is that one divides the activities in different sets, and each set is regarded as a participant.

2.3 Creating WSDL Definitions

A WSDL definition will be created for each fragment process. In this paper, generally, the WSDL definition's name attribute will be the same as the participant's name. Its targetNamespace attribute will come from the original process's WSDL definition. It is explained in [4] ,[6] and [8] that (1) to support the inter-communications between the fragment processes some new artifacts are created in their WSDL definitions, and that (2) some artifacts are copied from the WSDL definition of original process in the proper fragment process to enable the communication between the clients and the fragment processes.

2.4 Creating Fragment Processes

A fragment process will be created for each participant. The newly created process is an empty process. In the run-time, necessary artifacts will be copied into the fragment process. What is copied into the fragment process depends on the participant associated. The new fragment process will be named after the participant. In this thesis, the algorithms in [2] are adapted to handle the creation of the process fragment. The function PROCESS_CHILD in that chapter is the main algorithm to add activity in the fragment process. The main idea is that given the main process, the participant, and the fragment process, we iterate through all activities in the main process in a top-down manner, in each iteration an activity will be handled.

2.5 Collecting Information for BPEL4Chor

The ultimate aim of the splitting process in this paper is to output the BPEL4Chor Choreography on the process upon the partition specification given. The idea is to prepare the output so that we collect the information pieces available in each stage of the splitting procedure, store the them in intermediate data, at the end assemble all pieces together for output.

After Process Fragmentation, the information useful for BPEL4Chor in the run-time is:

1. Topology
2. Grounding
3. Control link fragmentation
4. Data Dependency fragmentation

3 CONTROL LINK FRAGMENTATIONS

Fragmenting Control Link is to split the control link on the base of whether or not the control link crosses over processes given a specific partition specification. In the former case the control flow is transmitted by exchanging message between the processes. In the step 5 of the Figure 2.1, we can see that in this stage the control link is to be fragmented. The results of step 4 (Process Fragmentation) are taken as input, and "step 5 - fragmenting control link" will be run, output of the procedure will be the modified fragment processes, WSDL definitions, and BPEL4Chor artifacts i.e. message links. In this paper, the details of fragmenting control link will be explained.

3.1 Concept to Fragment Control Link

The concept is introduced as follows. Figure 3.1 illustrates the concept of fragmenting control link. The non-split process (left) contains two activities, *a* and *b*, with is connected by a link *l(a, b, q)*. The notion *l(a, b, q)* is used to indicate the link from activity *a* to activity *b* with *q*, which is a Boolean expression and can be omitted if it is not specified. The activity *a* is placed in Participant 1 (right) after the fragmentation, and the activity *b* in Participant 2 (right). Besides the placement of *a* and *b*, some constructs are created in Participant 1 and 2.

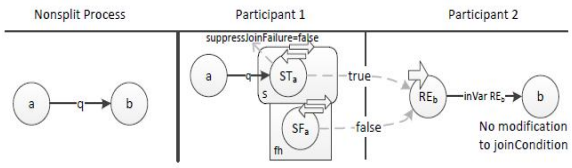

Figure 3.1: Concept for Splitting Control Link across Processes [2].

The original link *l(a, b, q)* is split across the participants. We regard the newly created constructs in Participant 1 as *sending block*, and the one in Participant 2 as *receiving block*. The control is propagated from activity *a* to activity *b*, via message exchange between sending block and receiving block. The message encodes whether the control is in valid- or faulty status. That way, the behavior of the original link in the non-split process is reproduced in the fragment processes.

3.2 Fragmenting Control Link in BPEL

In the last section, the conceptual fragmenting control link is introduced. A simplified scenario where a control link across processes is given, and the solution was that one reproduces the control flow of the original link by creating the *sending block* and *receiving block* each in the

corresponding fragment process. In this section we focus on how one fragments the control link in BPEL. Consider a more general scenario, where we get fragment processes by splitting the main process with the help of partition specification. The control links in the fragment processes are still not handled. The question is how to apply the concept for fragmenting control link on those control links that are split, and how to handle the links that are not split. We can handle the link by proposing the SPLIT-CONTROL-LINK algorithm. It goes through all the fragment processes and checks for each basic activity of the current fragment process whether each of its outgoing control links cross over processes. The procedure SPLIT-CONTROL-LINK terminates after each fragment process has been processed. It iterates in the following sequence: (1) process (2) activity (3) source (4) link. Each iteration consumes one outgoing link of an activity and the corresponding incoming link that associated to another activity. As a result, when the procedure is ended, all the outgoing links and their associated incoming links are processed.

4. DATA DEPENDENCY FRAGMENTATION

In Figure 2.1, data dependency fragmentation (step 6) will be executed after the control link fragmentation. Unlike the BPEL-D method, the data dependency fragmentation in this paper splits the data dependency in an implicit way. We analyze the data-flow of the main process first. Then against a (data) variable and its reader (activity), the analysis result is able to tell us which writers the reader is dependent on. Based on that knowledge we create the construct in the fragment process that contains the writers of the data and in the fragment process in which the reader of the data presents. The local resolver is responsible for summarizing the data from the various writers and sending it to the receiving flow per message (in the case of the writers and the reader being in different fragment processes), while the receiving flow is responsible for collecting the data sent by the local resolver, assembling that data in its previous order, and eventually rerouting the data to the reader.

The message, which is exchanged between the two sides, contains not only the data but also the information of whether the writers succeed. A Writer Dependency Graph (WDG) is created in order to re-generate the control dependency of the writers in the receiving flow and therefore realize the '*last writer wins*' policy. Then a Partitioned Writer Dependency Graph (PWDG) based on the WDG is created to reduce the quantity of those exchanged messages.

4.1 Data-Flow Analysis of BPEL Process

One input of the algorithm for fragmenting data dependency in this paper is the data-flow analysis of the main BPEL process. It serves the purpose of determining the data dependencies between activities. Such a concept for data-flow analysis has been presented by Kopp et al. [5], [3]. It has been extended by Breier [10] and implemented by Gao [7]. To encode the data dependencies determined by the data-flow analysis, we use the function that describes a set of tuples.

5. RESULT OF BPEL4CHOR CHOREOGRAPHY

After the fragmentation of the data dependency, one has split the main process completely. The task at the end is to transform the executable fragment BPEL processes into Participant Behavior Descriptions (PBDs), and then output them together with the participant topology and grounding that have been prepared in the previous steps. This step is emphasized in Figure 2.1.

5.1 Participant Behavior Description (PBD)

In this stage the fragment processes are executable BPEL processes. a PBD is an abstract process profile. Therefore, we need to transform each of the executable fragment processes into an abstract process that meets the constraints as follows:

1. Each communication activity contains a namespace wide unique identifier.
2. The partnerLink, portType, and operation attributes in communication activity are excluded.
3. If there is a pair of combined <receive> and <reply>, an enforced *messageExchange* is created.

5.2 Participant Topology

The participant topology is the structure aspect of the BPEL4Chor choreography. It consists of three main notions: *participantType*, *participant*, and *messageLink*.

5.3 Participant Grounding

The participant grounding provides the web service specific configuration for the choreography. The two main notions are the *messageLink* and *participantRef*.

6 CONCLUSION AND FUTURE WORK

The main aim is to describe the concepts of splitting BPEL to BPEL4Chor Choreography processes out of a plain BPEL process. This paper provides a detailed description about the concept to BPEL4Chor choreography instead of a set of plain BPEL processes. Although the BPEL-D process presents the data flow explicitly and can easily be split, it is not sufficient to split the data dependency in a BPEL process while keeping the operational semantic of that original process, due to the parallelism and Death-Path-Elimination (DPE) in BPEL process. Therefore, BPEL4Chor splitting data dependency of a BPEL process is introduced. We lack in a specific definition or description of the out-coming wiring file after the process fragmentation in BPEL-D. Thus, we choose BPEL4Chor as the out-coming wiring specification. BPEL4Chor is a BPEL extension for defining choreographies and is suitable for the global wiring information. BPEL4Chor provides the interconnected interface behavior descriptions by utilizing the Abstract Process Profile for Observable Behavior of BPEL and by adding an interconnection layer on top of the abstract BPEL process.

In future, we are going to work on the implementing the BPEL4Chor splitting algorithm and other related algorithms. We are going to extend this work by implementing all the steps that are described in this paper.

REFERENCES

1. R. Khalaf "Note on Syntactic Details of Split BPEL-D Business Processes". Technical Report Computer Science 2007.
2. R. Khalaf "Supproting business process fragmentation while maintaining operational semantics : a BPEL perspective".2008.
3. O. Kopp, R. Khalaf, F. Leymann, "Reaching Definitions Analysis Respecting Dead Path Elimination Semantics in BPEL Processes". 2007
4. R. Khalaf, O. Kopp, F. Leymann. "Maintaining Data Dependencies across BPEL Process Fragments". International Journal of Cooperative Information Systems (IJCIS), 17(3):259–282, 2008.
5. O. Kopp, R. Khalaf, F. Leymann. "Deriving Explicit Data Links in WS-BPEL Processes", In IEEE International Conference on Services Computing. IEEE, 2008.
6. W3C, "Web Services Description Language (WSDL)", 2001. URL http://www.w3.org/TR/wsdl.
7. Y. Gao, "Implementing the Dataflow Analyse for WS-BPEL 2.0". (2246):54, 2010.
8. R. Khalaf, F. Leymann "Role-based Decomposition of Business Processes using BPEL", in International Conference on Web Services (ICWS 2006), pp. 770–780. IEEE Computer Society, 2006.
9. S. Breier. "Extended Data-flow Analysis on BPEL process " 2008.