Support of Parallel BPEL Activities for the TeamCom Service Creation Platform for Next Generation Networks

T.Eichelmann[1, 2], W.Fuhrmann[3], U.Trick[1] and B.Ghita[2]

[1] Research Group for Telecommunication Networks, University of Applied Sciences Frankfurt/M., Frankfurt/M., Germany
[2] Centre for Security, Communications and Network Research, University of Plymouth, Plymouth, United Kingdom
[3] University of Applied Sciences Darmstadt, Darmstadt, Germany
e-mail: eichelmann@e-technik.org

Abstract

The development of value added services is currently still very cost and time consuming. The need for specific user generated services demands for efficient service development methods. The Service Creation Environment of the TeamCom project represents a promising approach. It supports the developer with the development of services on the basis of re-usable service components called Communication Building Blocks and describes the service by a control logic. For the description of a value added service, a language, which was optimized for business processes, is suggested: the Businesses Process Execution Language (BPEL). The services described in BPEL are translated into Java code and compiled. For the deployment of the service, a Service Execution Environment, based on JSLEE is supported. This publication extends the TeamCom approach with the support of parallel processes. It is shown, how forked BPEL activities can be translated into JSLEE services.

Keywords

SCE (Service Creation Environment); NGN (Next Generation Network); BPEL (Business Process Execution Language); JAIN SLEE (JAIN Service Logic Execution Environment)

1. Introduction

The development of new services is currently still very time consuming and requires an extensive knowledge of the used technology, whereby the development of special services for small user groups or also individual users is hardly improved. So a high need of methods exists, which allow a simplification in the development of new value added services. Therefore the design of the services must be independent from the concrete implementation. This has the advantage that the developer can concentrate on the actual service design, without the requirement of detailed knowledge of the used technologies. Here the TeamCom project (TeamCom, 2009) comes into play. It allows a fast, economical and efficient creation of value added services by providing a Service Creation Environment for an abstract design of the service process and an automatic creation and deployment of value added services.

For the design of the services, the Business Process Execution Language (BPEL) (OASIS, 2004) is used, because it is an established business specification language. However a BPEL engine is appropriate for Web Services but it is not suitable for the control of real time, communication services. Therefore another approach is suggested, which converts the description of a business process into executable code. The generated code should be based on the JSLEE (Sun and Open Cloud, 2008) architecture, because this architecture is adapted on the requirements of communication services. This paper is an extension of the TeamCom approach. In the past, the TeamCom approach has only supported sequential services. This paper shows how forked BPEL activities can be translated into parallel running service components in JSLEE services.

The Content of this paper is structured as follows: In chapter 2 the architecture of the service platform is described. The 3rd chapter is concerned with the concept of the Service Creation Environment from the TeamCom project. Chapter 4 describes the lifecycle of the services from the idea, up to the realization. The conversion of forked BPEL processes to JSLEE services is described in the 5th chapter.

2. Architecture

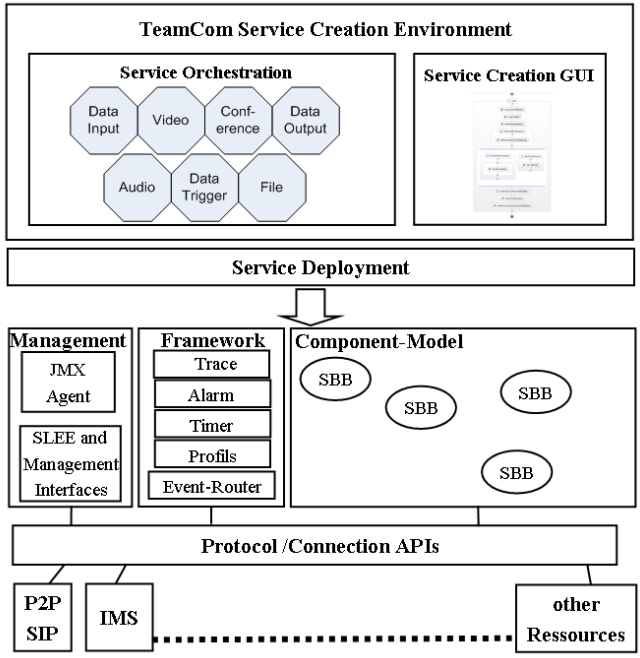

Figure 1: Architecture

The TeamCom Project proposes an integrated architecture (Lehmann et al., 2009) (Lasch et al., 2009) (Lasch2 et al., 2009) for service creation, deployment and

execution. This architecture (Figure 1) can be divided into four layers: The Service Creation Environment (SCE), which includes the design and composition of new services, the Service Deployment (SD), the Service Execution Engine (SEE) containing one or more Application Servers (AS) based on JAIN SLEE (JAIN Service Logic Execution Environment) and finally the Service Transport Layer (STL). The Service Transport Layer abstracts different protocols in order to enable upper service layers to be independent of a specific communication protocol. Therefore it supports several communication networks, e.g. IP Multimedia Subsystem (IMS) (TS 23.228, 2006) or Peer-to-Peer SIP (P2PSIP, 2009).

The SCE contains a GUI, permitting a developer to orchestrate service components and to integrate external services easily without having to develop low-level software code. The Service Execution Environment establishes a layer, where created Service Components are deployed and activated. In order to support NGNs that are based on IP the following requirements for a Service Execution Environment have been defined: independence from the operating system, service orchestration based on components, comfortable component deployment, support for SIP (Rosenberg et al., 2002), extensibility for other protocols and possibility for co-operation between numerous application servers. To fulfil these requirements the architecture makes use of the JAIN SLEE standard.

The JAIN SLEE standard defines a component, event and transaction based architecture. The architecture is written in the Java language and standardised by a Java Community Process. It is part of the JAIN (Java API for Integrated Networks) Initiative consisting of several telecommunication companies. JAIN SLEE is designed for ensuring low latency and providing high throughput to accomplish the requirements for communication services. It uses a distributed component model similar to Enterprise Java Beans (EJB) (Sun and Oracle, 2006). In the standard so called Resource Adaptors (RA) are defined abstracting the underlying infrastructure. These Resource Adaptors provide a common Java API which hides the communication protocol underneath. In detail when a communication protocol message is received the corresponding RA translates this message into a Java event class. Afterwards the event and an activity class both together are passed to the JAIN SLEE event router. A JAIN SLEE activity represents a session of a communication protocol e.g. a SIP dialog. The event router utilizes these objects to look up the services which requested to receive the specific event. Accordingly the service itself is able to react on the event and to create an answer by using defined Java Interfaces. The answer is translated to a communication protocol response by a RA.

The service itself is composed of one or more Service Building Blocks (SBB). These SBBs contain the application/service execution logic and are deployed on a JAIN SLEE Application Server such as Mobicents (Mobicents, 2009). The SBB component model includes a lifecycle, registration and security management. In addition, SBBs are able to access timer, trace, alarm and profile facilities which are also provided by a JAIN SLEE server.

3. Service Creation Environment

As depicted in the last chapter the Service Creation Environment includes service orchestration on the basis of reusable Service Components and existing services. In addition to the components a service logic is required for describing the workflow properly. The following two sections explain both.

3.1. Communication Building Blocks

Eight elementary Service Components called Communication Building Blocks are derived from the requirements for telecommunication services. By combining these elementary components it should be possible to describe and generate other value added services. These Communication Building Blocks comprise audio, video, text, file, conference, data input, data output and data trigger components. This chapter discusses the abstract Service Components in detail.

Audio: The Audio Component handles all kind of audio communication including the establishment of a call, answering calls, manipulation of audio streams (e.g. mixing, transcoding) and sending and receiving DTMF tones.

Video: The Video Component is responsible for playing and recording of video streams. It enables to create and close video calls and to combine different video signals for merging a new video stream.

Text: This component exchanges messages between two partners and has the capabilities of handling strings, e.g. search for a specific word in a text, replace alphabetic characters or change the encoding of a text.

File: The File Component handles creation, deletion, sending and receiving of binary files. Another task of File is to write and read any kind of data from and to any position in a file. Finally this component is able to rename files or directories.

Data Input: All kind of data queries are processed by the Data Input component. This includes database queries as well as reading data from a sensor.

Data Output: The counterpart of Data Input is Data Output being concerned with writing data to a destination e.g. to a database or to control an actuator.

Conference: This is a special kind of communication component because it re-uses internal functions of the previously described components, e.g. audio stream mixing.

On top of this, the Conference Component provides functionalities for creating and deleting conference “rooms”, adding and removing users to/from a room.

Data Trigger: The Data Trigger is closely related to an event generator. If a specific data trespasses a value an event is triggered. This data can be a sensor value, a timestamp or periodical dates.

3.2. Service Logic

Business workflows within and in between enterprises usually follow defined processes, the so called business processes. This resulted in the definition of the BPEL (Business Process Execution Language) standard, an XML (W3C, 2008) based process description language, building completely on web services. Most often BPEL is used to orchestrate web services but the language is protocol independent. It is possible to create so called bindings which could specify the usage of BPEL for other protocols than SOAP (Simple Object Access Protocol) (SOAP, 2007). In BPEL it is possible to define synchronous and asynchronous processes. Thus it fits very well to be used in combination with JAIN SLEE. For a structured process different BPEL tags are defined, e.g. sequence, if, while. Moreover forked processes can be created, allowing for execution in parallel and synchronisation afterwards. A BPEL process has the ability to invoke other services asynchronously and itself can be invoked from other services.

4. Service Creation and Deployment

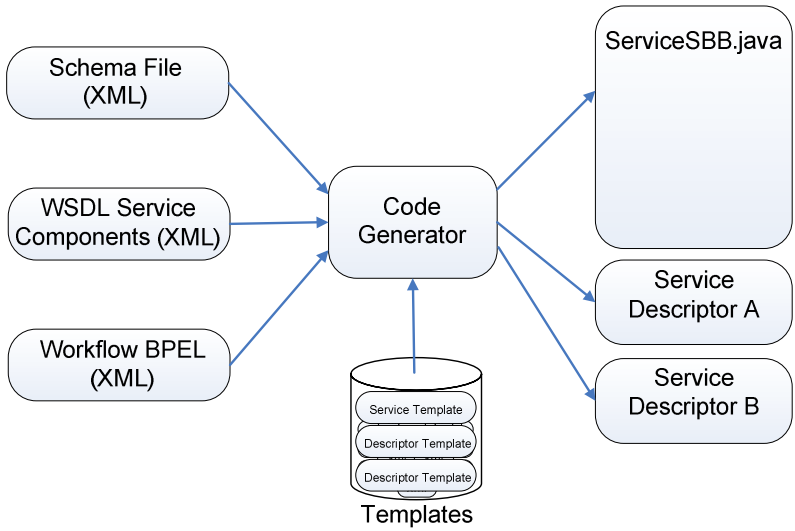

Figure 2: CodeGenerator

Service creation and deployment (Eichelmann et al., 2008) can be realised in five steps: writing a non-technical description of the service, converting this description to a service description language, analysing the description language, generating Service Building Blocks from the description language and deploying the service. First a service description has to be verbalised. The description could be e.g. a text, depending on the process and the involved people. Afterwards a service developer is able to create a BPEL based description graphically via the service creation Graphical User Interface (GUI). In this step the service developer utilizes the Communication Building Blocks which are included in BPEL as partner links and configures their input. After finishing the BPEL process description the generation of

the service will start (see Figure 2). The service designed by the service developer shall not be executed on a BPEL process engine. Instead the service shall run on a JAIN SLEE application server.

So the BPEL process has to be converted in a form to be executable as a JAIN SLEE service. The code generator analyses the BPEL process and parses the workflow step by step. The result from each individual step is saved in template files. Finally the goal of the code generator is the creation of the Java classes and the necessary descriptor files needed for a JAIN SLEE service. While the code generator parses the BPEL process, it analyses the BPEL activities. Pending on the BPEL activity the code generator adds pre-defined Java fragments to the template files. Process activities which initiate events for the partners or waiting for events from them must be examined to figure out which Communication Building Block is affected by this event. For each method which can be invoked on a partner a pre defined Java method exists. If the Communication Building Block is identified, the appropriate Java method which represents the used method from the BPEL process can be inserted into the template file. Other workflow activities perhaps need variables or data structures which are defined in BPEL. These structures are represented in XML schemata and have to be transformed into Java code also.

5. The conversion from BPEL processes to JSLEE Service Building Blocks

The BPEL process has to be converted to a JAIN SLEE service. This conversion is done by the Code Generator, which parses the process step by step. Every BPEL activity within the process has to be analysed by the Code Generator. It has to decide, whether the actual activity can be handled sequentially or if this activity requires parallel processing.

5.1. Conversion of sequential BPEL activities to SBBs

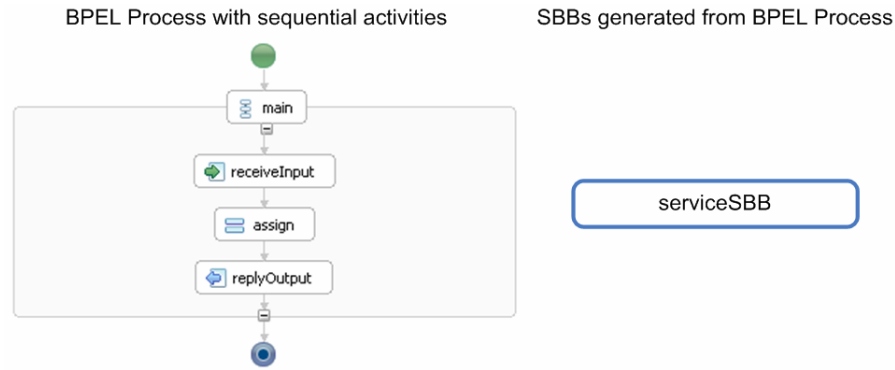

Figure 3: BPEL process with sequential activities

Sequential BPEL processes can be translated in JSLEE into one single SBB. In (Figure 3), a simple BPEL process is shown, which contains only three activities

within its main sequence, a receive activity called receiveInput, an assign activity called assign and a invoke activity called invokeCallback.

This process waits with his receive activity for an arriving event. After the event has arrived, the content of the variables can be manipulated with the assign activity. If this process for example represents an echo service, which sends an arriving text back to the sender, the arriving text is copied with the assign from the input string variable into the output string variable. If the text should get send back to the transmitter, the destination address of the message receiver can also be set with the assign activity. If the message is ready, it can be send with the invoke activity. This BPEL process only contains sequential BPEL activities, which means, that only one SBB is necessary for the JSLEE service.

5.2. Conversion of parallel BPEL activities to SBBs

If the Code Generator is reaching a flow activity while parsing a BPEL process, it is generating a new BPEL process from each branch he found within the flow sequence. (Figure 4) shows newly generated BPEL processes from the flow branches of a flow activity within the main sequence of a simple process.

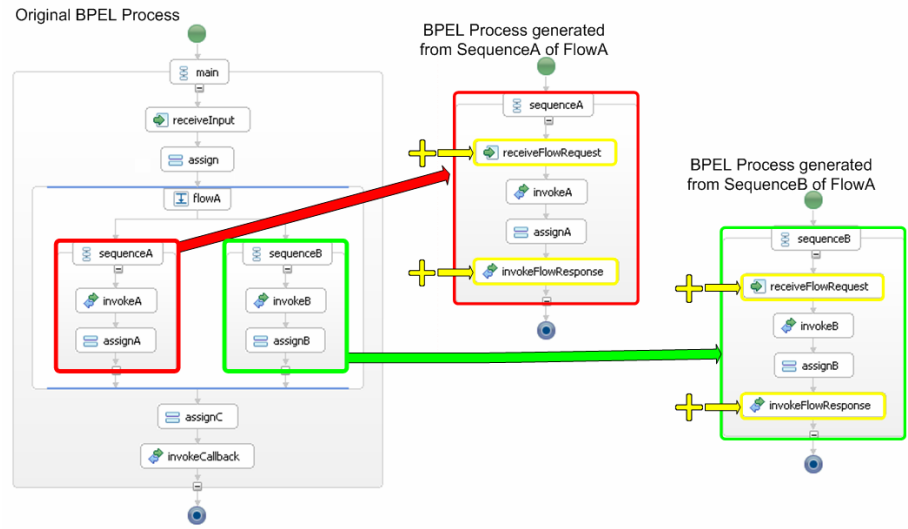

Figure 4: BPEL processes generated from a flow activity

This flow activity contains two flow branches. From every branch a new BPEL process will be generated. Every new process additionally gets a receive activity as first activity of the BPEL process and an invoke activity as last activity of the process. Later, from these two activities the SBB Java methods will be generated which sends and receives events. The newly generated BPEL processes are handed over to the Code Generator again. Now the Code Generator can generate new SBBs from the BPEL processes. If there are multiple flow activities nested within a BPEL process, this is repeated until all flow branches are transformed into new BPEL processes and are finally translated into SBBs.

A forked JSLEE service component is required for the transformation of the BPEL flow activity. In order to use parallel running service components in JSLEE, the SLEE standard offers the possibility to use several SBBs in one service. These SBBs run in parallel from each other. Each SBB can either form its own service or several SBBs can form a single service together. In this work, several SBBs are used to represent the flow activity. Each branch of the flow activity is represented in its own SBB. If a service consists of several flow activities, it is differentiated if they are contained in the same sequence (Section 5.3.1) or if the flow activities are contained within another flow activity (Section 5.3.2). Events are used to communicate between the SBBs. With these events, the necessary variables are transferred between the SBBs. (Figure 5) shows a BPEL process, which was extended by a flow activity and an assign activity compared to the process in (figure 3). The additional assign, which is called assignC is used to copy the returned variables from the flow SBB into the variables of the answer message.

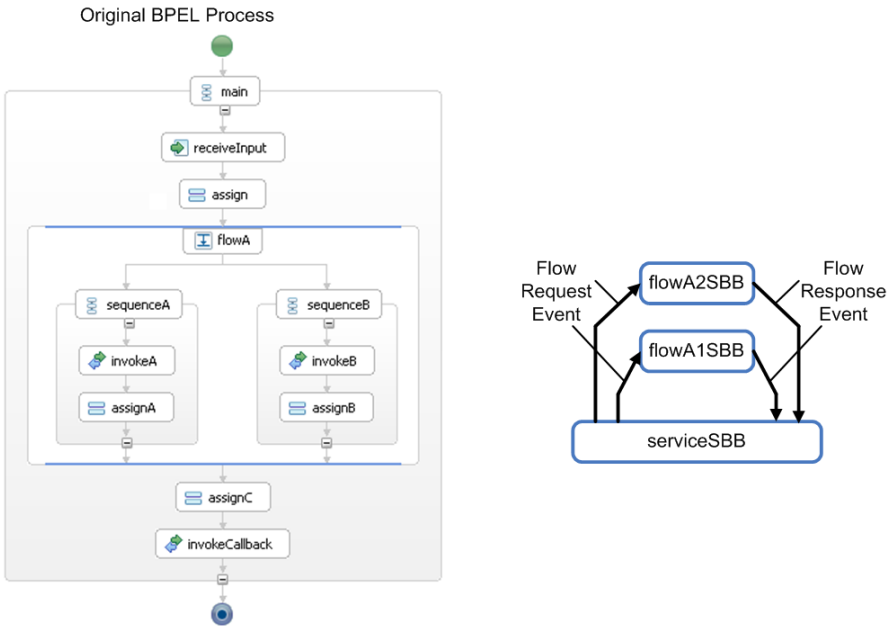

Figure 5: BPEL process with a single flow activity and the resulting SBBs

All activities which are not within the flow activity like receiveInput, assign, assignC and invokeCallback can further be run sequentially. The sequences within the branches of flow activity must be processed in parallel. Every branch, which exists in a flow, is converted into its own SBB. If a sequence in a flow branch contains further sequential activities, these are sequentially processed again within the SBB of the flow branch like the main sequence in the general SBB. The BPEL process in (figure 5) contains two branches in its flow activity. The flow activity is called flowA and the sequences are called sequenceA and sequenceB. In this example both sequences contain two activities each, an invoke activity and an assign activity. In each sequence a partnerlink is called and afterwards variables are set. This variables could

e.g. contain the status of the invoke activity. The processing of the activities behind the flow is continued after all branches of the flow activity are finished with processing. (Figure 5) shows the resulting SBBs from this BPEL process on the right side. The general SBB of this service called serviceSBB contains the sequential activities of the main sequence. In the SBB called flowA1SBB, the activities of the first flow branch (sequenceA with invokeA and assignA) are contained and in the SBB flowA2SBB, those of the second (sequenceB with invokeB and assignB). The name of an SBB which is generated from a flow branch consists of the name of the flow and an index and the word SBB. Events are used for the communication between SBBs. If an SBB reaches a position during its program code processing where an SBB, generated from a flow activity, should be called, it activates this SBB by sending an event. Request and response events have been defined for the call and for the answer of SBBs. Each SBB which was generated from a flow branch is called with its own request event and answers with a response event after its processing. Both, the request and the response event contain the necessary variables which will become available in both SBBs in this way. The generated SBBs and the direction of the request and response events are shown on the right side of (figure 5). A BPEL process is not only limited to one flow activity, it can contain as much flow activities as desired in different nesting levels. Two different nesting possibilities must be considered on the generation of the SBBs. On the one hand several flows can be contained in the same sequence; on the other hand further flow activities can also be contained in the branches of a flow.

5.3.1. Conversion of several flow activities within a sequence

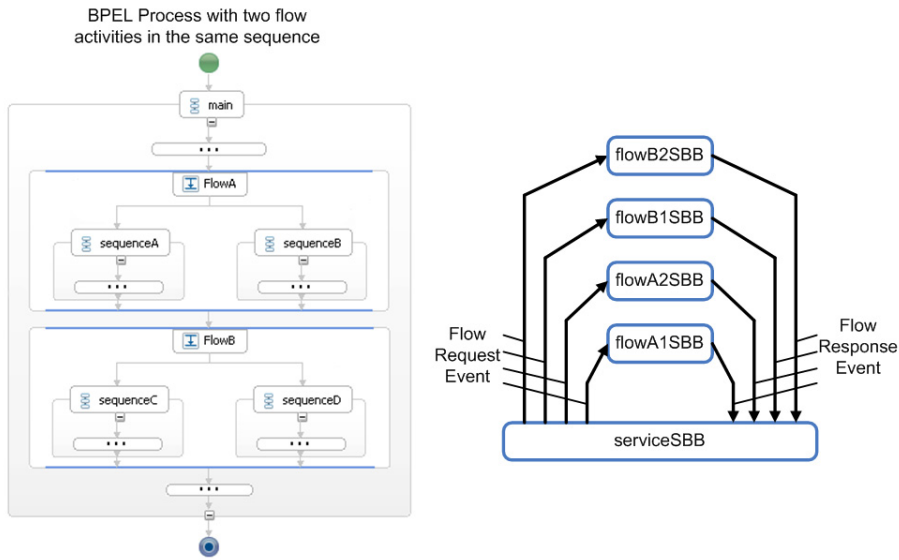

Figure 6: BPEL process with two flow activities within a sequence and the resulting SBBs

Figure 6 shows a simplified BPEL process which contains two flow activities in the same sequence. The first flow activity is called flowA and the second is called flowB.

Each flow activity consists of two flow branches and in each branch a sequence with further BPEL activities is contained. For each branch in each flow an SBB is generated. Each SBB represents the activities of the appropriate branch in Java code. So for example the flowA activity with the sequence sequenceA from the BPEL process, shown in Figure 6, is translated into an SBB with the name flowA1SBB and the right branch of flowB is translated into an SBB called flowB2SBB. All flow SBBs are again called with request events and they answer after their processing also again with response events. From this BPEL process the general SBB and four SBBs from the flow branches are generated, two SBB for flowA and two SBB for flowB. During the processing of the Java code, which represents the main sequence from the BPEL process, the flowA activity is reached first and request events are send to activate the SBBs FlowA1SBB and FlowA2SBB. Both SBBs process now their tasks in parallel from each other. The general SBB serviceSBB is activated again if both SBBs answer with a response event. Only then the program code for the second flow flowB can be reached. Here the SBBs FlowB1SBB and FlowB2SBB are called with request events and they answer with response events. Afterwards the java code that represents the remaining activities of the main sequence can be processed.

5.3.2. Conversion of flow activities nested within a flow activity

As already mentioned above, flow activities can be nested within flow activities in BPEL. Figure 7 shows such a BPEL process.

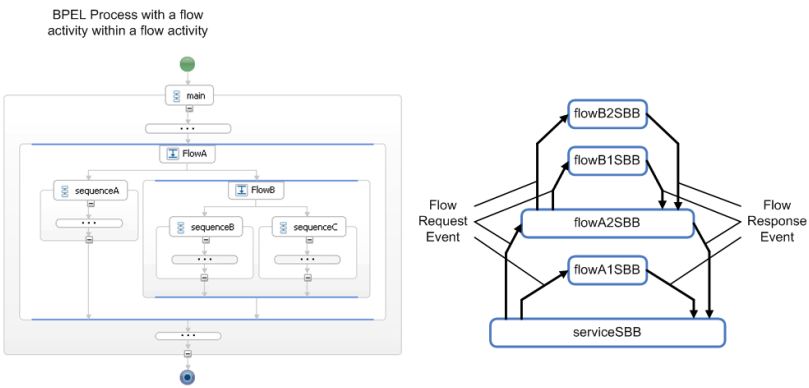

Figure 7: BPEL process with a flow activity nested within another flow activity and the resulting SBBs

FlowB is embedded in the right flow branch of flowA. Each flow branch is converted again into one SBB. An SBB is also generated for the flow branch of flowA that contains flowB. In this example the SBB is named flowA2SBB. The difference to flow activities within the same sequence is that the request events are

send by the SBB, in which the called flow is contained. In the example from (figure 7) the SBBs flowB1SBB and flowB2SBB are called by the SBB flowA2SBB, so the flowA2SBB sends the request events and also expects the response events. Therefore flowA2SBB can finish its processing only if flowB1SBB and flowB2SBB sends their response events and the serviceSBB can continue with the processing if flowA1SBB and flowA2SBB are finished.

6. Conclusion

The automated service development has a high potential, it also empowers small and medium sized enterprises to develop their own value added services in a time and cost efficient way. The service development should be just as simple as the production of web pages and therefore be applicable for more people. Nevertheless more services are leading to a better utilization and economy of the communication networks. The combination of the BPEL and JSLEE technologies simplify the development of these value added services. In addition the abstraction from specific communication protocols, the re-usable Communication Building Blocks and the use of existing partial services facilitates the service development. The TeamCom architecture shows a continuous solution from the service development to the deployment and remains independent from specific protocols. The presented approach for the conversion of parallel BPEL activities into JSLEE SBBs further extends the TeamCom project by various possibilities of value added services. It would also be possible to offer one ore more SBBs for every BPEL activity. This extended approach could represent the basis for a BPEL engine on a JSLEE application server.

7. Acknowledgment

The research project providing the basis for this publication was partially funded by the Federal Ministry of Education and Research (BMBF) of the Federal Republic of Germany under grant number 1704B07. The authors of this publication are in charge of its content.

8. References

Eichelmann, T., Fuhrmann, W., Trick, U. and Ghita, B. (2009), “Creation of value added services in NGN with BPEL”, *SEIN*, Wrexham, 2008.

Gudgin M. (2007), "SOAP Version 1.2 Part 1. Messaging Framework (Second Edition)", W3C, April 2007.

Lasch, R., Ricks, B. and Tönjes, R. (2009), “Service Creation Environment for Business-to-business Services”, *SOCNE*, 2009.

Lasch, R., Ricks, B. and Tönjes, R. (2009), “Konzept eines BPEL zu JSLEE Compilers auf Basis wieder-verwendbarer Kommunikationsbausteine”, *Mobilkommtagung*, 2009.

Lehmann, A., Eichelmann, T., Trick, U., Lasch, R., Ricks, B. and Tönjes, R. (2009), “TeamCom: A Service Creation Platform for Next Generation Networks”, *ICIW*, Venice, 2009.

Mobicents Open Source JAIN SLEE Server Project Web Site (2009): http://www.mobicents.org. (Accessed 18 August 2009)

OASIS (2004), OASIS Standard, “Web Services Business Process Execution Language Version 2.0”, OASIS.

P2PSIP IETF Project Web Site (2009), http://www.p2psip.org/. (Accessed 18 August 2009)

Rosenberg J. (2002), RFC 3261 “SIP: Session Initiation Protocol” IETF, June 2002.

Sun Microsystems, Open Cloud (2008), JSR-000240 Specification, Final Release, „JAIN SLEE (JSLEE) 1.1”, Sun.

Sun Microsystems, Oracle Corporation (2006), JSR-000220 Specification, Final Release, "Enterprise JavaBeans, Version 3.0", SUN, May 2006.

TeamCom Project Web Site (2009): http://www.ecs.fh-osnabrueck.de/teamcom.html. (Accessed 18 August 2009)

TS 23.228 (2006), "IP Multimedia Subsystem (IMS); Stage 2 (Release 5)", 3GPP, June 2006.

W3C (2008), "Extensible Markup Language (XML) 1.0 (Fith Edition)", *W3C Recommendation*, November 2008.