e-Informatica Software Engineering Journal, Volume 3, Issue 1, 2009

Transformational Design of Business Processes in BPEL Language

Andrzej Ratkowski*, Andrzej Zalewski*, Bartłomiej Piech**
*Instituite of Control and Computation Engineering, Warsaw University of Technology
**Department of Electronics and Information Technology, Warsaw University of Technology
a.ratkowski@elka.pw.edu.pl, a.zalewski@ia.pw.edu.pl, b.piech@elka.pw.edu.pl

Abstract

A transformational approach to the design of executable processes in Business Process Execution Language (BPEL) is presented. It has been built upon the transformations of business processes accompanied by a formal approach based on process algebras used to verify the behavioral equivalence of business processes. The initial business process can be denoted in BPEL, then a series of transformations is executed upon it. The process resulting from the transformation is verified whether it preserves behaviour denoted by the process being transformed. The transformations improve non-functional properties of the process (performance, modifiability, granularity, maintainability) but do not change its original behaviour. The transformations are steered by Architecture Trade-off Analysis Method (ATAM) that shows the direction of changes and helps an architect to decide which of them to apply. An example of the application of our approach in real-life business process design has also been presented. The paper presents general idea of the design process, theoretical basis of the method as well as experimental verification of the approach and a tool implemented to support the method.

1. Introduction

The following paper presents the concept of design method that is a subject of PhD thesis written by Andrzej Ratkowski under Prof. Krzysztof Sacha's supervision. The article is an extension of a previous paper [26].

The ability to define and execute business processes seems to be one of the most important advances introduced by the research and commercial developments on Service-Oriented Architectures (SOA). The worlds of business modelling and software systems development have never been closer to each other – it is now possible to express software requirements in terms of services and business processes composed of them. BPEL have become a standard for defining executable business processes. This in turn triggered an extensive research on the modeling and verification techniques suitable for those processes.

The approaches presented above, as well as the verification techniques, can indicate absence or existence of certain flows in BPEL processes. However, these are not methods of business processes design – they do not provide any guidance on how to improve the quality attributes of designed systems like maintainability, performance, reusability etc. This is what the approach is aimed at.

In this paper we advocate an idea of transformational design of BPEL business processes in which specified behaviour remains preserved, while quality attributes get improved. There are three basic roots of our approach:

1. software refactoring – the approach introduced by Opdyke in [24], further developed in [20], in which the transformations

of source code are defined so as to improve its quality attributes;
2. business process design – in the realm of SOA informal or semiformal methods dominate the research carried out so far – comp. Service Responsibility and Interaction Design Method (SRI-DM) [21];
3. business process equivalence – there have already been developed several notions of the equivalence between business processes based on Petri Nets [19] and Process Algebras [29].

The transformations of Business Processes are in the core of our approach and represent similar concept as popular software refactorings. Our original notion of business process equivalence has been introduced on a formal Process Algebra model of business processes (explained and discussed in section *Behavioural Equivalence*) and it has been proved that the defined transformations create processes equivalent to the one being transformed. These transformed processes are compliant in terms of their behaviour, however, they have quality attributes changed. These transformations may be steered by the quality scenarios and assessments performed using Architecture Trade-off Analysis Method (ATAM) [18].

This provides a foundation for the transformational design method in which a starting BPEL process is subject to a series of transformations yielding as a result behaviourally compatible model with improved non-functional properties like modifiability, maintainability, performance, reusability etc.

2. State of the Art

Many business processes design and maintenance methods are based on *Business Process Management* (BPM) concepts [31]. According to BPM, process life-cycle consist of five phases:
1. design – existing business processes are analysed and "to-be" processed are designed. The results of this phase are: process flows, main actors, resources and so on;
2. modeling – the purpose of this part is to model and make conclusions on process execution before its practical application;
3. execution – in execution phase processes are put into practice and run in physical environment;
4. monitoring – running processes are monitored, functional and non-functional properties are measured;
5. optimization – this phase is responsible for improvement of processes.

The BPM concept is broadly applied in the processes domain, however, we believe that there is no specialised application in SOA context. Current paper tries to fill this gap.

In the field of general process modeling there are approaches based on *Unified Modeling Language* (UML) like presented in [28]. The authors present suitability of UML activity diagrams for business process.

In the context of Service Oriented Architecture there exist special methods devoted to design business processes like mentioned previously Service Responsibility and Interaction Design Method (SRI-DM) [21]. The SRI-DM method is based on transformation from UML use-cases towards services with proper divided functionality and sequence diagrams that express desired process.

The approach similar to proposed in the current paper is presented in [15]. The authors propose modeling business process as a Petri net and such transformations of the net to reach optimal value of some goal function. The proposed approach is based on optimization techniques.

The research of the current paper is concentrated on converting BPEL processes to one of the formal models that can be subject to model-checking techniques. A survey of such approaches can be found in [3]. It reveals that all of the most important formal models of concurrent systems have been applied: Petri nets (basic model, high-level, coloured) – comp. [14], [32], Process Algebras – comp. [12], [11], Lotos – comp. [9], [30], Promela and LTL – comp. [13], [16], Abstract State Machines – comp. [8], [27], Finite State Automata – comp. [11]. These conversions make it possible to detect deadlock and

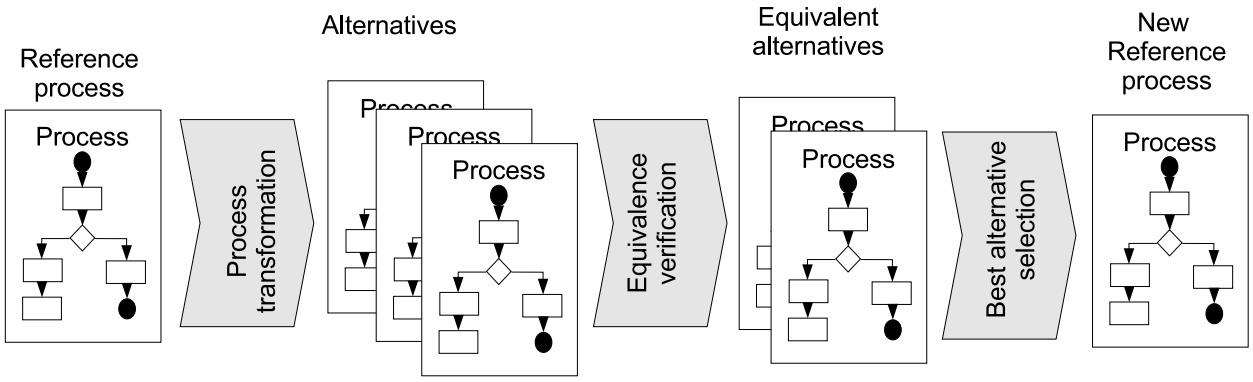

Figure 1. Process transformation algorithm

livelock as well as reachability analysis with automated model checkers.

3. Process Transformation Design Approach

The algorithm of process transformation design is depicted in Figure 1.

1. As it was mentioned in the introduction, the algorithm starts with the original process that is delivered by business oriented staff and the primal process bring up only functional aspects of the process. Functional aspects of the process are: necessary activities, order of activities, relation between them, exchanged data, basic external services invocation and so on. The process is called *reference process*. In following iterations the original process is slightly changed by refactoring transformations [25], [20] like:
   - service split – split one complex services into two or more smaller ones that cover the primary one functionality,
   - service aggregation – opposite to service split – composing two or more services in one larger service,
   - parallelization – making serial activities to run parallel,
   - asynchronization – reconstruction of communication protocol from synchronous to asynchronous.

   The above transformations are called *refactorings* and they are only examples of possible refactorings. Obviously, in a given process only some subset of transformations is possible and a smaller subset is rational.
2. A few independent refactorings on a current process make a few alternative processes which should be equivalent to the original process or at least changes in behaviour should be known.
3. Behaviour preservation is checked by means of behavioural equivalence verification step. In this step formal methods of Process Algebra (PA) [6] are used. The result of verification is either elimination of not-equivalent alternative or accepting changes in behaviour that the transformation makes. The way the transformation changes behaviour is exactly known owing to PA formalism.
4. After eliminating or accepting, all alternatives that are left are evaluated against interesting non-functional properties like:
   - performance,
   - safety,
   - maintainability,
   - availability,
   - or any important property.

   The measure of each property is calculated by using specified metrics, models [7] or simulations.
5. In the following step one alternative is selected amongst others. The selection is based on Architecture Trade-off Analysis Method (ATAM) [18]. In short, the method examines sensitivity of non-functional parameters to design properties and marks out *trade-off points*. Trade-off points are decision variables that affect more than one quality attributes. Changing the value of trade-off points in-

creases some quality attributes and decreases others. In case of SOA, services granulation is an example of such trade-off point. When services are bigger and not numerous then we have good performance and weak maintainability and reusability. If we split the system into more services, performance will decrease but maintainability and reusability will increase.

6. After selecting one alternative process, the selected process becomes new reference process and the algorithm returns to the beginning.

The above steps lead from process that is correct from functional point of view to process that has best or acceptable good non-functional quality attributes.

All steps of the algorithm are guided by a human designer and supported by automatic tools that may:

– suggest possible transformations of a reference process,
– verify behavioural equivalence,
– compute quality metrics of alternatives,
– point out trade-off points.

The conclusion is that the transformational approach does not try to make a totally automatic process design, because, in our opinion, it is impossible without human ability involvement. Instead, the transformational method supports a human designer's creative work in tasks difficult for a human.

4. Behavioural Equivalence

Behavioural equivalence verification is based on Process Algebra transformation and manipulation of BPEL processes [6].

4.1. Process Algebra for Behavioural Equivalence

Process Algebra (PA) [6] is formal semantic that express concurrent and distributed processing. It is specially devoted for parallel, loosely coupled and asynchronous communication so it is tailored to BPEL analysis. During our research we used LOTOS [2] realisation of PA.

Using the LOTOS notation, one can model any process or chain of communicating processes, simulate processes execution and, what is the most important in the context of refactoring, verify equivalence of two different processes. The equivalence is verified by *simulation*, *bisimulation* or *preordering* analysis [6].

To be able to use PA in stated problem it is necessary to use some kind of mapping from BPEL activities to PA terms. There are a few existing BPEL to PA mappings [10, 4], but none of them exactly fit to the needs of transformational process design. Firstly, because they demand full semantic checking in equivalence verification, that is too precise for refactoring. In case of the refactoring equivalence verification, if one process is transformed, its semantic changes but its behaviour does not. Another aspect is that an important property of mapping BPEL to PA for refactoring is that it has to make simple models with possibly the smallest statespace – during the design procedure there are a few changing scenarios and each of them has to be verified – the time spent for one verification is limited. This is the motivation for us to develop new mapping. Mappings of BPEL activities to PA formulas are presented in Table 1.

The mappings do not take into account data values or condition probability. This is motivated by simplification (and better verification performance) of the model. From another point of view, making some assumptions, there is no actual need to examine values of variables in equivalence verification.

There is an artificial mapping of activity which is not explicit part of BPEL but is necessary for equivalence verification. This is *activity dependency* mapping. Let us assume that there are two activities in BPEL process that are not directly attached to each other (by e.g. <sequence> or <switch>) but by shared variable, like in the following example:

```
<receive variable="PurchaseOrder"
         name="ReceivePurchase" />
...
<assign name="assignOrder">
  <copy>
```

Table 1. Sample mappings BPEL activities to PA formulas. Part 1

BPEL	LOTOS Process Algebra
empty <empty name="emptyName" [...] </empty>	process empty_emptyName[dummy] := exit endproc
external service invocation <invoke inputVariable="ivName" outputVariable="ovName" name="invName" [...]> [...] </invoke>	process invoke_invName[ivName,ovName] := ivName;ovName;exit endproc
receive message <receive variable="vName" name="receiveName" [...]> [...] </receive>	process receive_receiveName [vName] := vName;exit endproc
reply <reply variable="vName" name="replyName" [...] > [...] </reply>	process reply_replyName[vName] := vName;exit endproc
assign variable value <assign name="asgName" <copy> <from variable="fromVar"> <from to="toVar"> </copy> </assign>	process assign_asgName[fromVar, toVar] := fromVar;toVar;exit endproc
parallel execution <flow name="flowName"> < ... name="activityA"/> < ... name="activityB"/> [...] </flow>	process flow_flowName[dummy] := activityA || activityB ... endproc
sequential execution <sequence name="seqName"> < ... name="activityA"/> < ... name="activityB"/> [...] </sequence>	process sequence_seqName[linkSyn] := activityA >> linkSyn;activityB >> ... endproc Note: linkSyn should be placed according to potential link synchronization usage.

Table 2. Sample mappings BPEL activities to PA formulas. Part 2

BPEL	LOTOS Process Algebra
conditional execution <switch name="switchName"> <case ...> < ... name="activityA"/> </case> <case ...> < ... name="activityB"/> </case> </switch>	process switch_switchName[dummy] := hide ended in (activityA [ ] activityB ...) endproc
pick <pick name="pickName"> <onMessage partnerLink="ncname" portType="qname" operation="opA" variable="ncname"> activityA </onMessage> <onMessage partnerLink="ncname" portType="qname" operation="opB" variable="ncname"> activityB </onMessage> </pick>	process pick_pickName[dummy] := activityA[ ]activityB endproc
link <flow name="flowName"> <links> <link name="XtoY"/> </links> <sequence name="X"> <source linkName="XtoY"/> <invoke name="A" .../> <invoke name="B" .../> </sequence> <sequence name"Y"> <target linkName="XtoY"/> <invoke name="E" .../> </sequence> </flow	process flow_flowName[dummy] := hide XtoY in (sequence_X[XtoY] |[XtoY]| sequence_Y[XtoY]) endproc

```
    <from variable="PurchaseOrder"/>
    <to variable="ShippingRequest"/>
  </copy>
</assign>
```

Then activity dependency mapping will be:

```
process act_dependency[dummy]
  receive_ReceivePurchase[PurchaseOrder]
      |[PurchaseOrder]|
  assign_assignOrder[PurchaseOrder,
                     ShippingRequest]
endproc
```

The activity dependency expresses indirect dependency of two activities of which, one needs output data from another, no matter what structural dependency (sequence or parallel) in the process are.

4.2. BPEL Behavioural Equivalence

There are a few approaches to determine behavioural equivalence (or in other words behaviour preservation) of refactored processes. In [24] the author proposes such definition, that two systems are equivalent when the response for each request is the same from both systems. According to [22] communication-oriented systems are equivalent if they send messages in the same order.

In case of transformational design we assume that every service fulfills stateless postulate. It means that when BPEL process invokes external service then in every invocation response for some request is always the same, it is independent of history. This assumption leads to a conclusion that state of external services (and all environment) is encapsulated inside the invoking service.

To make this assumption usable and to prove how it can be used we needed some PA theory.

$$B \xrightarrow{x} B' \tag{1}$$

The above formula means that process B reaches state B' after receiving an event (message) x.

Now PA semantics is defined using *inference rules* that has form:

$$\frac{premises}{conclusions}(sidecondition) \tag{2}$$

For example parallel execution (without synchronization) || has 2 symmetric rules:

$$\frac{B1 \xrightarrow{x} B1'}{B1||B2 \xrightarrow{x} B1'||B2} \text{ and } \frac{B2 \xrightarrow{x} B2'}{B1||B2 \xrightarrow{x} B1||B2'} \tag{3}$$

an preceding (sequential composition) >> has 2 rules:

$$\frac{B1 \xrightarrow{x} B1'}{B1 >> B2 \xrightarrow{x} B1' >> B2} \text{ and } \frac{B2 \xrightarrow{\sigma} B2'}{B1 >> B2 \xrightarrow{i} B2} \tag{4}$$

where σ is successful termination and i is unobservable (hidden) event.

If external service S is stateless then:

$$\forall y \in Y S \xrightarrow{y} S \tag{5}$$

where Y is a set of all events. This means that every event, generated externally or from the subjected service, does not change the state and answer from the service.

To analyse a BPEL process using PA terms, the BPEL process has to be translated into PA using mapping mentioned in previous section. The product of translation is a set of PA processes that are sequentially ordered by BPEL steering instructions – sequences, flows, switches and so on. Additionally, a part of mapping is *activity dependency* processes. This artifact symbolizes data dependency between elements.

Let us symbolize it with *dependency operator*:

$$A]x]B \tag{6}$$

which means that state B can be started after A is successfully terminated and event x is emitted (or received).

Below we can see some example, that shows what is our behavioural equivalence based on.

The given process has a set of operations connected with dependency sequence:

$$(A]x]C]z]D) \tag{7}$$

C waits for A result and D for C result.

Beside the above dependency, the process has also structural sequence defined by <sequence> instruction $A \rightarrow B \rightarrow C \rightarrow D$, where B is instruction which is not connected by *activity dependency*. We can relax the structural sequence and consider the process as:

$$(A]x]C]z]D)||B \tag{8}$$

That means that we can treat $(A]x]C]z]D)$ and B as two parallel independent activities.

The proof that (8) is true for stateless services.

1. If there is no external service (8) is true by the definition because there is no interaction between $(A]x]C]z]D)$ and B,
2. If there is stateless external service S, then:

$$\forall y(A]x]C]z]D)||S \xrightarrow{y} ((A]x]C]z]D))'||S \tag{9}$$

and

$$\forall yS||B \xrightarrow{y} S||B' \tag{10}$$

which leads to:

$$\begin{gathered}(A]x]C]z]D) \xrightarrow{y} (A]x]C]z]D)' \\ \Rightarrow (A]x]C]z]D)||B \xrightarrow{y} (A]x]C]z]D)'||B\end{gathered} \tag{11}$$

and

$$\begin{gathered}B \xrightarrow{y} B' \\ \Rightarrow (A]x]C]z]D)||B \xrightarrow{y} (A]x]C]z]D)||B'\end{gathered} \tag{12}$$

The equation (12) is parallel execution inference rules (3) which is proof of (8)

If S was stateful, then

$$\exists y(A]x]C]z]D)||S \xrightarrow{y} (A]x]C]z]D)'||S' \tag{13}$$

then

$$(A]x]C]z]D)||B \xrightarrow{y} (A]x]C]z]D)'||B' \tag{14}$$

this would mean that there are some interactions between $(A]x]C]z]D)$ and B, and that they can not be treated independently.

The above theory makes it possible to divide the whole BPEL process into parts, that are only dependant by *activity dependency* and also makes possible to check if every refactored process is contained in these dependencies. This technique is related to *program slicing* [1] used broadly in source code refactoring. The BPEL service with defined activity dependencies and without structured constraints (sequences, flows, conditional and so on) is called *minimal dependency process* and is used to check the behavioural equivalence. After refactoring, the new (refactored) process has to be translated to PA and its PA image must fulfill preorder relationship with the *minimal dependency process*. Refactored process has to be subgraph of *minimal dependency process states graph*.

5. Transformation Steering

The process of transformations is steered by a method based on the Architecture Trade-off Analysis Methods (ATAM) [18]. The ATAM helps to identify *trade-off points*, that are parameters that have impact on a few quality aspects of the analyzed system. The impact of *trade-off points* is positive on one aspect and negative on another. So to designate proper value of such parameter there a trade-off has to be reach on this parameter.

ATAM helps to decide which alternative should be selected during the process design. In that way ATAM steers transformation in a design algorithm.

6. Process Design Example

In order to illustrate how transformational design works in practice, a simple example is presented below. The example is inspired by BPEL specification [17]. The quality of process is measured in two aspects: performance and reusability. The performance metric is response time under a given load, and reusability is measured by number of interfaces that whole service provides.

6.1. Reference process

The business process is a typical purchase of goods service. The service is composed of three activities: invoicing, order shipping and production scheduling. The activities of the process are organized as follows:

1. the process receives purchase order, receives product type, quantity and desired shipping method,
2. shipping service is requested and the price of shipping is received,
3. an invoice is requested from an invoicing service, the invoice contains product price and shipping price,
4. the production of goods is scheduled by request to a scheduling service.

Each activity is executed in sequence. Next activity starts after the previous is finished. The reference process and surrounding services are depicted in Fig. 2.

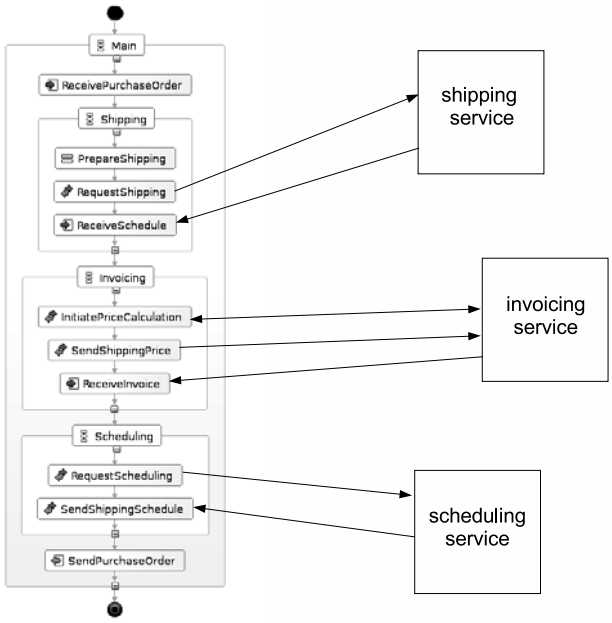

Figure 2. Purchase order reference process

6.2. Process Alternatives

For the current reference process, the designer proposes three alternatives that seem to be equivalent. Alternative (1) is a process that first makes request for shipping service and afterwards, parallelly requests shipping service and invoice service.

Alternative (2) starts all three requests parallelly – invoicing, shipping and scheduling service.

Alternative (3) is a bit more sophisticated – the reference service is split into three services. One of them invokes shipping service, the second one parallelly invokes invoicing and scheduling services, the third service composes two subservices. The alternatives are presented in Fig. 3.

6.3. Equivalence verification

In the current stage of algorithm, alternatives are verified to be behavioural equivalents to reference process. The technique of verification is described in section 5. The result of the verification is as follows:

– alternative 1 is behaviourally equivalent unconditionally,
– alternative 2 is not equivalent, because a request to invoicing service and shipping service depends on data received from shipping service. When all three requests startsat the same time, we can not guarantee, that the data from shipping service is received before a request to scheduling and invoicing services is made.
– alternative 3 is behaviourally equivalent.

Upon the above information, the designer decides to remove alternative 2 from the alternatives set.

6.4. Alternatives Evaluation – Performance

As it was mentioned at the beginning of the section, alternatives are evaluated in performance and reusability aspects. Performance is defined as a mean response time estimation. The web service and connections between services can be modeled, with queueing theory, as M/M/1//inf system. It means that requests arrive to the system independently with exponential interval distribution and response time is also exponentially distributed. Thanks to the above assumptions, average response time of whole system can be estimated as a sum of average responses from its components: services and links between them. To make evaluation simpler, we assume that every network connection has the same average latency R_N. So average response time of the reference process is:

$$R_{RP} = R_{BPEL_RP} + R_{\text{shipping}} + R_{\text{invoicing}} + R_{\text{scheduling}} + 7R_N \tag{15}$$

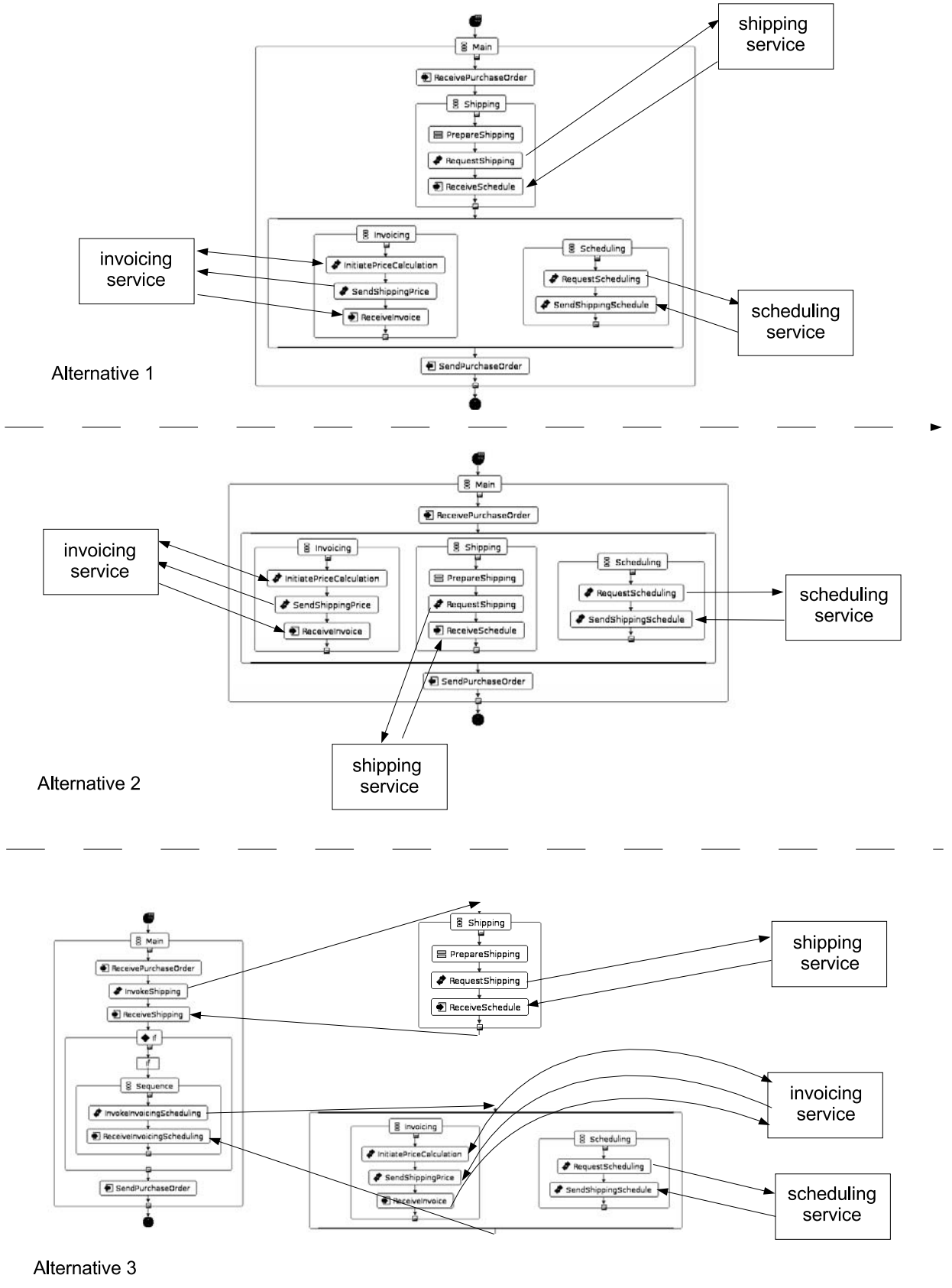

Figure 3. Possible alternatives for reference process

An important fact in the above equation, is that average response times of invoicing, shipping and scheduling are simply added, because requests to services are made consequently, one by one. Let us assume additionally values of each parameter:

- $R_{BPEL_RP} = 2$ ms (average time of processing of main BPEL process),
- $R_{\text{shipping}} = 3$ ms (avg. resp. time from shipping service),
- $R_{\text{invoicing}} = 5$ ms (avg. resp. time from invoicing service),
- $R_{\text{scheduling}} = 4$ ms (avg. resp. time from service),
- $R_N = 1$ ms (avg. network latency).

That gives $R_{RP} = 21$ ms.

For alternative 1 average response time is:

$$R_{A1} = R_{BPELA1} + R_{\text{shipping}} + \max(R_{\text{invoicing}}, R_{\text{scheduling}}) + 7R_N \quad (16)$$

the difference between alternative 1 and reference process is that invoice and scheduling services are requested parallelly, so response time from the parallel part is a maximum of response times from invoicing and scheduling. When we assume that $R_{BPELA1} = R_{BPELRP}$ then: $R_{A1} = 17$ ms.

Finally alternative 3 average response time is:

$$R_{A3} = R_{BPELA3_1} + R_{BPELA3_2} + R_{BPELA3_3} + R_{\text{shipping}} + \max(R_{\text{invoicing}}, R_{\text{scheduling}}) + 11R_N \quad (17)$$

that gives: $R_{A3} = 25$ ms.

6.5. Alternatives Evaluation – Reusability

As a reusability metrics is taken the total number of interfaces that a service delivers. Reference process and alternative 1 delivers four interfaces: one to main composed process and three to elementary services: invoicing, shipping and scheduling. Alternative 3 delivers 6 interfaces: three to basic services, one to composite service and two new interfaces to two sub services – shipping request and invoicing scheduling request.

All the above data are gathered in Table 3.

6.6. Best Alternative Selection

By means of ATAM method it is possible to identify the trade-off point, which is in following example services quantity. If composite service consist of more basic services, then it is more reusable, however, performance suffers.

In the current stage the new reference process has to be designated. Apparently alternative 1 is the best choice. Alternative 1 is better than the current reference process in performance measure and not worse in reusability. Alternative 3 is better in reusability than alternative 1 but much worse in performance, even worse than reference process.

7. Tool Support

As it was mentioned previously, an important goal of the research is to deliver a tool that will support usage of transformational process design. The tool is currently under development. In the current section a current status of tool development is described. The tool is based on open-source NetBeans IDE [23]. It is planned that whole design process will be held in NetBeans. BPEL editor, which is already implemented in the IDE, is used. Beside BPEL editor, a graphical editor is necessary as it will guide

Table 3. Quality metrics for reference process and alternatives

	Reference process	Alternative 1	Alternative 3
Average response time	21 ms	17 ms	25 ms
Reusability	4	4	6
Services quantity	1	1	3

the process design iteration – its layout will be similar to Figure 1. The editor will be the main window of the tool. A designing user will be able to click on every alternative and look inside using native BPEL editor. In the main window there will also be all the important data about quality of alternatives.

7.1. BPEL Refactoring

To automate refactoring process in BPEL language it was necessary to create the tool which provides these features. It was proposed to automate such types of transformation: renaming (variable, partnerLink, and correlationSet), aggregation, asynchronization, parallelization, split. After selecting a part of the code in BPEL file one of the mentioned transformations can be realized (if it is possible). Such tool has not been already implemented – this is why I decided to implement an idea of creating the application as a plug-in to Netbeans IDE which automates refactoring process. There are numerous engineering challenges connected with the detailed design of tool support for BPEL transformations. These have been presented in detail below.

7.1.1. Renaming

It is the simplest type of refactoring – changes the name one of the three elements in BPEL (variable, partnerLink, and correlationSet) and all the occurrences of this element in other language constructions. It seems to be an easy transformation but it is relevant. It would be difficult to do it manually because BPEL contains a lot of constructions with reference to other elements. For instance reference to variable may occur in such elements: receive, reply, invoke, onMessage, throw, copy from, copy to and in XPath expressions: wait, onAlarm, if, else if, while, repeatUntil, forEach. As we can see it is much easier to use an automatic tool which finds all the occurrences of the chosen element in BPEL code. The offered application provides these features. We can change the name one of the mentioned elements and do not have to worry about occurrences in other BPEL constructions – program will do it for us automatically.

7.1.2. Aggregation

Composing one or more services into larger one seems to be easy. It is because somebody may think that it is enough to move logic from one service to another and that is all. It is a wrong approach because there are a lot of other elements which we have to focus on.

First of all, we must find the BPEL file that contains the logic of the invoked process which is automatically done by the proposed tool.

Secondly, it is needed to move elements such as variable, partnerLink, correlationSet and namespaces to the process that is invoking, because all the elements are used in logic which we want to encapsulate.

Last but not least, it may happen that the used variables, partnerLinks or namespaces in invoked process have the same name as in process which is invoking the first one. This situation is considered in the proposed tool – when the situation occurs, application changes the name of the specified element in all constructions where reference to this element occur in order to prevent name collision. A similar situation may happen in namespaces because the one we want to add is already defined. In this case it is also needed to change the name of the added namespace in every place where it occurs. Also a very important thing is to ensure that variable used as input in invoked process (attribute variable in receive element) after the transformation will be the same as input in invoke element before transformation. A similar situation occurs when we have synchronous invocation with output variable it has to be checked whether variable used in reply element will be the same as an output variable in invoke element before refactoring. This situation is also supported by the application.

7.1.3. Asynchronization

In this type of refactoring the offered tool also provides a few conveniences that automate process of transformation. First of them is finding

as many operations as it is possible which are invoked after selected element and they are independent. After that we can change the invocation method from synchronous to asynchronous. If there are no independent operations transformation will be terminated.

To change the invocation from synchronous to asynchronous some changes in WSDL and BPEL file in invoked process are needed. We have to delete (in WSDL file) an output element in operation construction (to make invocation asynchronous) and add a new input element for a reply to the primary process (we can not use the same input element for the reply because the types of used variables may be different). Moreover in BPEL file we must change synchronous element reply to element invoke to make connection asynchronous – we need to define additional partnerLink element to make the connection possible. The application supports all of these transformations.

To finish the transformations it is necessary to provide some modifications in the primary process. This is because of the type of invocation (asynchronous) which we introduce earlier by changing a partner WSDL file. After all independent operations we need to place element receive to collect a response from partner process and delete an attribute outputVariable in the invoke element.

The last thing to remember is to define correlation element to ensure that response will be transferred to the right instance of the primary process. This is why proposed tool makes some modifications in WSDL file of partner process. To be more accurate application defines property element and two propertyAlias elements. Thanks to that it is possible to define correlationSet and correlation elements in primary process file which guarantee that message will be delivered to right process instance. After all mentioned operations, which proposed tool supports, refactoring is finished.

7.1.4. Split

Splitting the service without using the automatic tool may also be difficult. To extract a part of the service and then create another service to be invoked inside the primary service we have to create two new files – a BPEL file and a WSDL file. Moreover, we must fill them with all the necessary information which is indispensable to make a network connection with the new process. As well it is requisite to change the primary process so that the connection with the new process will be possible. All the mentioned operations are supported by our tool.

First of all, the application chooses two variables – one as input variable and second as output variable for synchronous invocation of the new process. Choosing variables is not complicated operation because as input variable is chosen first which occurs in selected code to extract and it is used for one of the operations. In case of the occurrence more than one variable, all of which are not initiated in a chosen logic, the transformation will be terminated. Selection of the output variable is very similar to the selection of the input variable – if exists exactly one variable, which is initiated in the selected logic and used later after the selected code, it will be chosen as output variable.

Next, BPEL and WSDL files are created. In a WSDL file all the necessary constructions are created, such as: message, portType, operation, partnerLinkType and namespaces which defines complex types of the variables. Then using definition created in a WSDL file it is possible to make a new BPEL file and create constructions: variable, partnerLink, namespaces, etc. and place selected logic in new file. All of the operations are supported by the application.

At the end it is necessary to modify the primary process. To make a connection with the new process the application adds invoke activity (with all attributes) instead of the extracted logic and element partnerLink (also with all attributes). After all these transformations splitting the process into parts is possible.

7.2. Tools for Equivalence Verification

The algorithm of equivalence verification consist of three steps:

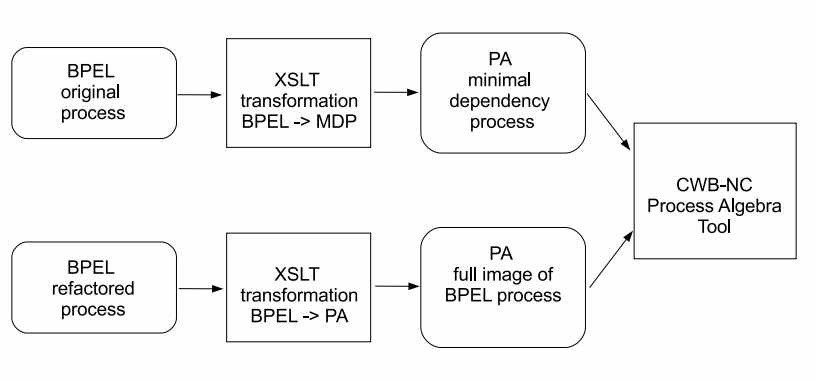

Figure 4. Structure of verification process

1. translating BPEL process to minimal dependency process (MDP) – this step is made only once at the beginning of the refactoring process,
2. translating BPEL process to its PA image,
3. checking preorder relationship of PA image with minimal dependency process.

As a part of the developed tool, translation BPEL to PA was made by means of an XSLT processor, as the PA processor was used Concurrence Workbench for New Century (CWB-NC) [5]. The structure of verification system is in the Figure 4.

The transformation from BPEL to its PA image is a quite trivial action as it was used XSLT preprocessor. The XSLT processor automatically maps BPEL instructions into their PA equivalences as it is listed in Tables 1 and 2.

The second type of mapping – from BPEL to its MDP image is more sophisticated. As it is needed to resolve indirect dependencies between BPEL activities there graph manipulation techniques are applied.

8. Summary and Further Work

A method for transformational design of SOA business processes in BPEL has been presented. It has been founded on the formal framework of process algebras as well as the concept of process equivalence originally developed by the authors. The transformations are aimed at improving certain properties like e.g. modifiability and performance while preserving the behaviour specified by the starting business process model. The whole framework has been accompanied by a prototype tool which has been integrated with NetBeans environment in the form of a plug-in. The challenges resolved during tool development have by no means turned out to be trivial. Therefore, they have also been discussed in the paper which should become a valuable resource of the real implementation experiences in the field of transforming BPEL as well as for the continuation of the work presented here.

The approach has been validated on an exemplary design case. The result of such a case study are promising though some more complicated cases would provide a chance for a more in-depth validation of the whole approach.

References

[1] D. Binkley and K. B. Gallagher. Program slicing. *Advances in Computers*, 43:1–50, 1996.

[2] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Comput. Netw. ISDN Syst.*, 14(1):25–59, 1987.

[3] F. Breugel and M. Koshkina. Models and verification of BPEL. 2006.

[4] J. Cámara, C. Canal, J. Cubo, and A. Vallecillo. Formalizing WSBPEL business processes using process algebra. *Electr. Notes Theor. Comput. Sci.*, 154(1):159–173, 2006.

[5] R. Cleaveland. Concurrency workbench of the new century, 2000. http://www.cs.sunysb.edu/~cwb/.

[6] R. Cleaveland and S. Smolka. Process algebra. 1999.

[7] A. D'Ambrogio and P. Bocciarelli. A model-driven approach to describe and predict the performance of composite services. In *WOSP '07: Proceedings of the 6th international*

workshop on Software and performance, pages 78–89, New York, NY, USA, 2007. ACM.

[8] D. Fahland and W. Reisig. ASM-based semantics for BPEL: The negative control flow. In *Abstract State Machines*, pages 131–152, 2005.

[9] A. Ferrara. Web services: a process algebra approach. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.

[10] A. Ferrara. Web services: a process algebra approach. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 242–251, New York, NY, USA, 2004. ACM Press.

[11] H. Foster, J. Kramer, J. Magee, and S. Uchitel. Model-based verification of web service compositions. In *18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.

[12] H. Foster, S. Uchitel, J. Magee, J. Kramer, and M. Hu. Using a rigorous approach for engineering web service compositions: A case study. In *SCC '05: Proceedings of the 2005 IEEE International Conference on Services Computing*, pages 217–224, Washington, DC, USA, 2005. IEEE Computer Society.

[13] X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL web services. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 621–630, New York, NY, USA, 2004. ACM.

[14] S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In *Proceedings of the BPM 2005*, pages 220–235, Nancy, France, Sept. 2005. Springer-Verlag.

[15] I. Hofacker and R. Vetschera. Algorithmical approaches to business process design. *Computers & OR*, 28(13):1253–1275, 2001.

[16] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley Professional, September 2003.

[17] IBM, BEA, and Microsoft. Business process execution language for web services. http://citeseer.ist.psu.edu/669609.html, 2003.

[18] R. Kazman, M. H. Klein, M. Barbacci, T. A. Longstaff, H. F. Lipson, and S. J. Carrière. The architecture tradeoff analysis method. In *Proceedings of ICECCS*, pages 68–78, 1998.

[19] A. Martens. Simulation and equivalence between BPEL process models. In *Proc. of Intl. Conference DASD'05*, 2005.

[20] F. Martin. *Refactoring: improving the design of existing code*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[21] D. E. Millard, H. C. Davis, Y. Howard, L. Gilbert, R. J. Walters, N. Abbas, and G. B. Wills. The service responsibility and interaction design method: Using an agile approach for web service design. pages 235–244, 2007.

[22] I. Moore. Automatic inheritance hierarchy restructuring and method refactoring. In *OOPSLA '96: Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 235–250, New York, NY, USA, 1996. ACM.

[23] NetBeans IDE. http://www.netbeans.org/.

[24] W. F. Opdyke. *Refactoring Object-Oriented Frameworks*. PhD thesis, Urbana-Champaign, IL, USA, 1992.

[25] A. Ratkowski and A. Zalewski. Performance refactoring for service oriented architecture. *ISAT '2007: Information Systems Architecture And Technology*, 2007.

[26] A. Ratkowski and A. Zalewski. Transformational design of business processes in SOA. In *Proceedings of CEE-SET*, 2008.

[27] W. Reisig. Modeling and Analysis Techniques for Web Services and Business Processes. In *FMOODS 2005, Athens, Greece, June 15–17, 2005. Proceedings*, volume 3535, pages 243–258. Springer Verlag, May 2005.

[28] N. Russell, W. van der Aalst, Arthur, and P. Wohed. On the suitability of UML 2.0 activity diagrams for business process modelling. In *APCCM '06: Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling*, pages 95–104, Darlinghurst, Australia, Australia, 2006. Australian Computer Society, Inc.

[29] G. Salaün, L. Bordeaux, and M. Schaerf. Describing and reasoning on web services using process algebra. In *ICWS '04: Proceedings of the IEEE International Conference on Web Services*, page 43, Washington, DC, USA, 2004. IEEE Computer Society.

[30] G. Salaün, A. Ferrara, and A. Chirichiello. Negotiation among web services using LOTOS/-CADP. In *ECOWS*, pages 198–212, 2004.

[31] W. van der Aalst, A. ter Hofstede, and M. Weske. Business process management: A survey. In *Business Process Management*, *Lecture Notes in Computer Science*, pages 1–12. Springer, Berlin, Heidelberg, 2003.

[32] Y. Yang, T. Tan, J. Yu, and F. Liu. Transformation BPEL to CP-Nets for verifying web services composition. In *Proceedings of NWESP '05*, page 137, Washington, DC, USA, 2005. IEEE Computer Society.