Transformational Design of Business Processes for SOA

Andrzej Ratkowski and Andrzej Zalewski

Warsaw University of Technology
Department of Electronics and Information Technology
{a.ratkowski@elka.pw.edu.pl,
a.zalewski@ia.pw.edu.pl}

Abstract. By describing business processes in BPEL (Business Process Execution Language) one can make them executable. Then a problem arises how to assure that some non-functional requirements concerning e.g. performance of these processes, are met. In the paper a transformational approach to design of business processes is presented. To check equivalence of business processes resulting from the transformations, a BPEL description is converted to Process Algebra (Lotos version) and model-checking techniques are applied. The paper contains also an example of applying the proposed approach in a real-life situation.

Keywords: SOA, BPEL, business process design.

1 Introduction

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

The recent research is concentrated on converting BPEL processes to one of the formal models that can be analysed with model-checking techniques. A survey of such approaches can be found in [3]. It reveals that all the most important formal modeling techniques developed for concurrent systems are applicable here: Petri nets (basic model, high-level, coloured) – (see e.g. [23], [21], [28]), Process Algebras – (see e.g. [12], [11]), Lotos – (see e.g. [9], [25]), Promela and LTL – (see e.g. [13], [15]), Abstract State Machines – (see e.g. [7], [26]), Finite State Automata – (see e.g. [11]). These conversions make possible deadlock and livelock detection as well as reachability analysis with automated model checkers.

The approaches presented above accompanied by appropriate verification techniques can detect certain flows in BPEL processes. However, they are not methods of business processes design – they do not provide any guidance on how to

Z. Huzar et al. (Eds.): CEE-SET 2008, LNCS 4980, pp. 76–90, 2011.

improve the quality attributes of designed systems like maintainability, performance, reusability etc. This is what our approach is aimed at.

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

1. Software refactoring – the approach introduced by Opdyke in [20], further developed in [17], 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 – e.g. Service Responsibility and Interaction Design Method (SRI-DM) [14];
3. Business process equivalence – there have already been developed several notions of the equivalence between business processes based on Petri Nets [16] and Process Algebras [24].

The transformations of Business Processes are in the core of our approach and represent similar concept as popular software refactorings. In a formal Process Algebra model of business processes there have been introduced our original notion of business process equivalence (explained and discussed in section *Behavioral 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 behavior, however, quality attributes have changed.

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 behaviorally compatible model with improved non-functional properties like modifiability, maintainability, performance, reusability etc.

The rest of the paper has been organized as follows: the section *Transformational Approach* describes generally concept of our transformational approach, sections *Behavioral Equivalence* drills down formal aspects of our method, finally section *Process Design Example* shows practical example of the application of the proposed approach.

2 Transformational Approach

The process of transformational design of business processes has been depicted in figure 1.

1. The transformational desing starts from establishing reference process representing only functionality of the process, order of the activities, relations between them as well as exchanged data and external services invocations. In following iterations the original process is gradually changed by refactoring transformations [22], [17] like:
   – Service split – dividing one complex services into two or more more simple ones,

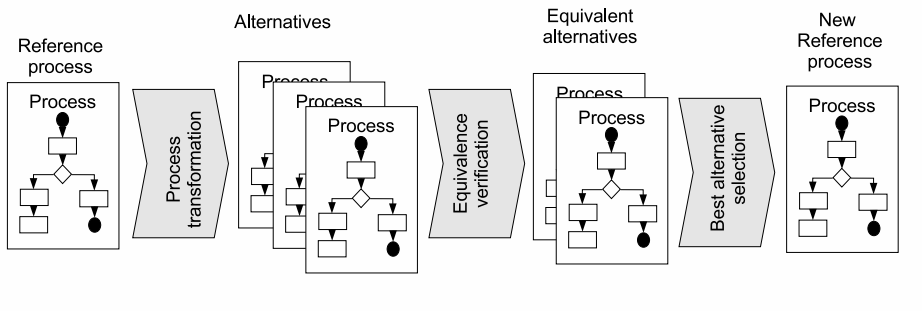

Fig. 1. Process transformation algorithm

- Service aggregation – opposite to service split: composing two or more services into one larger,
- Parallelization – making serial activities to run in parallel,
- Asynchronization – replacing synchronous communication protocol with asynchronous one.

The above transformations are referred to as *refactorings* or transformations and are only examples of possible refactorings.

2. A few independent refactorings or transformations executed on a given process create a few alternative processes, which should be equivalent to the original one or at least changes in behaviour should be known.
3. Behavioral equivalence verification step is aimed at checking behaviour preservation. In this step formal methods of Process Algebra (PA) [5] are used. The result of verification is either elimination of not-equivalent alternative or acceptance of changes in behaviour that transformations caused. The transformation changes behaviour is exactly known thanks to PA formalism.
4. After eliminating some of the alternatives or accepting all of them, alternatives are evaluated against non-functional properties like:
   - performance,
   - safety,
   - maintainability,
   - availability,
   - or any important property.

   The measure of each property is calculated e.g. with the metrics [18], models [8] or simulation.
5. A single alternative is selected from the set of acceptable processes. This is based on the evaluation performed in the previous phase as well as the predefined desing preferences.

The above steps should lead from the process, which is correct from functional point of view, to the process that has desired non-functional quality attributes.

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

– Suggest possible transformations of reference process,
– Verify behavioral equivalence,
– Compute quality metrics of alternatives,
– Point out quality attribute trade-off points.

3 Behavioral Equivalence

Behavioral equivalence verification is based on the transformation of BPEL processes to Process Algebras [5], which is a formal model, particularly suitable for the modeling of concurrent, loosely coupled and asynchronously communicating systems such as BPEL business processes.

3.1 Process Algebra for Behavioral Equivalence

We use LOTOS [2] implementation of PA as a model of BPEL process. To achieve this we have devised a mapping from BPEL to PA terms. There have already been proposed a number of BPEL to PA mappings – e.g. [10] or [4], however, they do not meet the needs of transformational design as they define full semantic equivalence preserving every detail of the internal structure of the process (i.e. the order of activities and their relation with other activities). This strictness narrows the possibility of changing the process structure or even makes it impossible. Therefore, we need a looser equivalence definition to assure that enough freedom is given to the trasformation of business processes.

A separate issue is that transformations should produce simple models with possibly smallest statespace (the mappings referred above produce rather complicated models). During design procedure a few alternative process structures are considered and the equivalence of each of them has to be verified – this in turn should not be too time-consuming if the method is expected to be of a practical importance.

Most important mappings of BPEL activities to PA formulas have been presented in table 1.

The mappings neither take into account data values nor condition fulfillment. This is motivated by simplicity of the model and its more efficient verification with a model-checker.

There has also been added 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>
     <from variable="PurchaseOrder"/>
     <to variable="ShippingRequest"/>
```

Table 1. Sample mappings BPEL activities to PA formulas

BPEL	LOTOS Process Algebra

external service invocation
 	
`<invoke inputVariable="inputName"`
`  outputVariable="outputName"`
`  name="invokeName" [...]`
`</invoke>`
	`process invoke_invokeName`
	`  [inputName,outputName] :=`
	`hide tau in (`
	` inputName;tau;outputName;ended;exit`
	`  )`
	`endproc`

receive message
 	
`<receive variable="variableName"`
`  name="receiveName" [...]`
`</receive>`
	`process receive_receiveName`
	`  [variableName] :=`
	`hide tau in (`
	`  variableName;tau;ended;exit`
	`  )`
	`endproc`

assign variable value
 	
`<assign name="assignName"`
`  <copy>`
`  <from variable="fromVar">`
`  <from to="toVar">`
`  </copy>`
`</assign>`
	`process assign_assignName`
	`  [fromVar, toVar] :=`
	`hide tau in (`
	`  fromVar;tau;toVar;ended;exit`
	`  )`
	`endproc`

parallel execution
 	
`<flow name="flowName">`
`  < ... name="activityA"/>`
`  < ... name="activityB"/> [...]`
`</flow>`
	`process flow_flowName[dmmy] :=`
	`hide ended in (`
	`  activityA |[ended]| activityB ...`
	`  )`
	`endproc`

sequential execution
 	
`<sequence name="seqName">`
`  < ... name="activityA"/>`
`  < ... name="activityB"/>[...]`
`</sequence>`
	`process sequence_seqName[dmmy] :=`
	`  activityA >> activityB >> ...`
	`endproc`

conditional execution
 	
`<switch name="switchName">`
`  <case ...>`
`  < ... name="activityA"/>`
`  </case>`
`  <case ...>`
`  < ... name="activityB"/>`
`  </case>`
`</switch>`
	`process switch_switchName[dmmy] :=`
	`hide ended in (`
	`  activityA [] activityB ...`
	`  )`
	`endproc`

```
  </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 that one needs output data from another, no matter what structural dependencies (sequence or parallel) in the process are.

3.2 BPEL Behavioral Equivalence

The key structure of our definition of behavioral equivalence is *minimal dependency process* (MDP). MDP is the process that is as simple as possible but still gives the same response for given stimulation as original process. MDP is constructed as set of activities executed in parallel that do not interact with each other. It is achieved by relaxing unnecessary structural activities that are in the original process. Current section supplies theoretical basis for the construction of MDP and the evidence that it can be used for process equivalence definition.

There are few approaches to determine behavioral equivalence (or other words behaviour preservation) of refactored processes. In [20] author proposes definition that two systems are equivalent when response for each request is the same from both systems. According to [19] 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 assumption. It means that when BPEL process invokes external service then every response for some request is the same and do not depend on history. This assumption leads to the 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 need basic PA theory.

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

The above formula means that process B reaches state B' after receiving 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 :

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

and precedes (sequential composition) >> has 2 rules :

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

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

If external services 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 by the subject service, does not change the state and answer of the service.

To analyse BPEL process using PA terms, the BPEL process has to be translated into PA using the mapping mentioned above. The result of the translation is a set of PA processes that are sequentially ordered by BPEL steering instructions – sequences, flows, switches and so on. Additionally, part of the mapping is *activity dependency* process. This artifact symbolizes data dependency between the activities (one needs data generated by the other one).

Let us denote 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 illustrate the foundations of the behavioral equivalence concept. Let us consider a process that 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.

Apart from the above dependency, the process has also structural sequence defined by <sequence> instruction A -> B -> C -> D, where B is an 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, we can treat (A]x]C]z]D) and B as two parallel, independent activities.

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 :

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

and

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

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

If S was statefull, 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 possible to break the whole BPEL process into a set of subprocesses, which depend on each other only by data dependecies represented as *activity dependencies*. This technique can be related to *program slicing* [1] used broadly in source code refactoring. BPEL service with defined activity dependencies and without structured constraints (sequences, flows, conditional and so on) is called *minimal dependency process* (MDP). Such a MDP becomes a basis for verification of equivalence of transformed (refactored) process with the requirements specified by MDP. After refactoring, new (refactored) process has to be translated to PA and its PA image must fulfill preorder relationship specified by MDP, therefore, refactored process has to be a subgraph of *MDP's states graph*.

3.3 Application of PA in Refactoring

As it was mentioned in the transformational approach introduction, PA formalism can help to decide if examined process is equivalent to reference one or points changes between processes. This is done by comparing execution state graphs of reference and refactored process. There are two possible results of this examination:

- execution graph of refactored process is subgraph of MDP reference process – this means that refactored process is equivalent to reference process or

– execution graph is not subgraph of MDP reference process – processes are not equivalent, but the information that can be obtained from the comparision is: what edges or nodes of execution graph are in refactored process but does not exist in MDP graph. Those extra edges and/or nodes are related to instructions or parts of code that makes importand diference between original and refactored process. The human designer can then decide, if theese differences are acceptable or not, in context of refactored process.

3.4 Algorithm and Tools for Equivalence Verification

Algorithm of equivalence verification consist of three steps:

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

For purposes of the test, translation BPEL to PA was made by using XSLT [27] processor, as the PA processor Concurrency Workbench for New Century (CWB-NC) [6] has been used. The structure of the verification system is presented in figure 2.

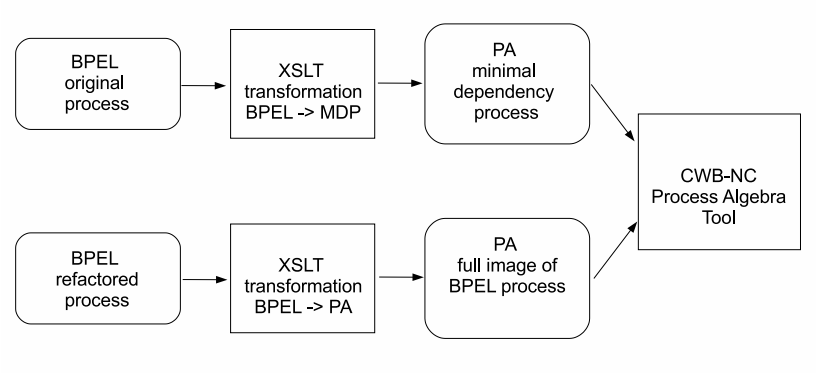

Fig. 2. Structure of verification process

4 Process Design Example

Entire approach can be illustrated on a practical example of an order handling process. During the transformations two quality attributes are taken into account: performance and reusability. First of them is measured by a response time under given load, the latter one by a number of interfaces the whole considered system provides.

4.1 Reference Process

The order handling process service is composed of three basic activities: invoicing, order shipping and production scheduling, which operate as follows:

1. A purchase order is received – it defines purchased product type, quantity and desired shipping method,
2. Shipping service is requested, which returns the shipping costs,
3. Invoice service provides the process with an invoice,
4. Production of ordered goods is scheduled with the request to scheduling service.

All the activities are performed consecutively. The reference process with accompanying services is depicted in fig. 3.

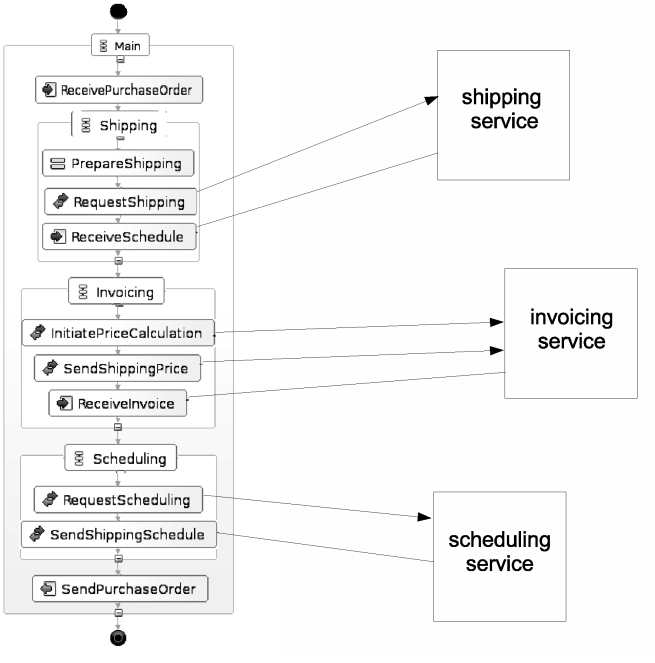

Fig. 3. Purchase order handling reference process

4.2 Process Alternatives

The designer consideres three alternatives to the reference process shown in fig. 4:

1. Alternative (1): the purchase process first makes request to the shipping service then in parallel to scheduling and invoice services.
2. Alternative (2): the process runs parallely all three requests – to invoice, shipping and scheduling services.

3. Alternative (3): a bit more sophisticated one – the reference service is splited into three separate services. First of them invokes shipping service, second one invokes invoice and scheduling services, the third one composes the other two subservices.

4.3 Equivalence Verification

Each of the three alternatives are verified whether they are behaviorally equivalent to the reference process. Technique of the verification has been described in section 4. The result of the verification is as follows:

- Alternative (1) is behaviorally equivalent unconditionally,
- Alternative (2) is not equivalent, because the request to invoicing and shipping services depends on the data received from shipping service. When all three requests start at the same time, we can not guarantee that the data from shipping service is received before the request to scheduling and invoicing services. This alternative cannot be accepted.
- Alternative (3) is behaviorally equivalent.

4.4 Alternatives Evaluation – Performance

As it was mentioned at the beginning of the section, alternatives performance and reusability is assessed so as to chose a preferred process' structure. Performance is measured as a mean response time under certain load level. The web service and connections between services can be modeled with queueing theory like M/M/1//inf system [8]. 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 the whole system can be estimated as a sum of average response times of 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:

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

Please note that average response times of invoicing, shipping and scheduling are simply added, thanks to the fact that services are invoked consecutively. Let us assume additionally that the values of appropriate parameters are as follows:

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

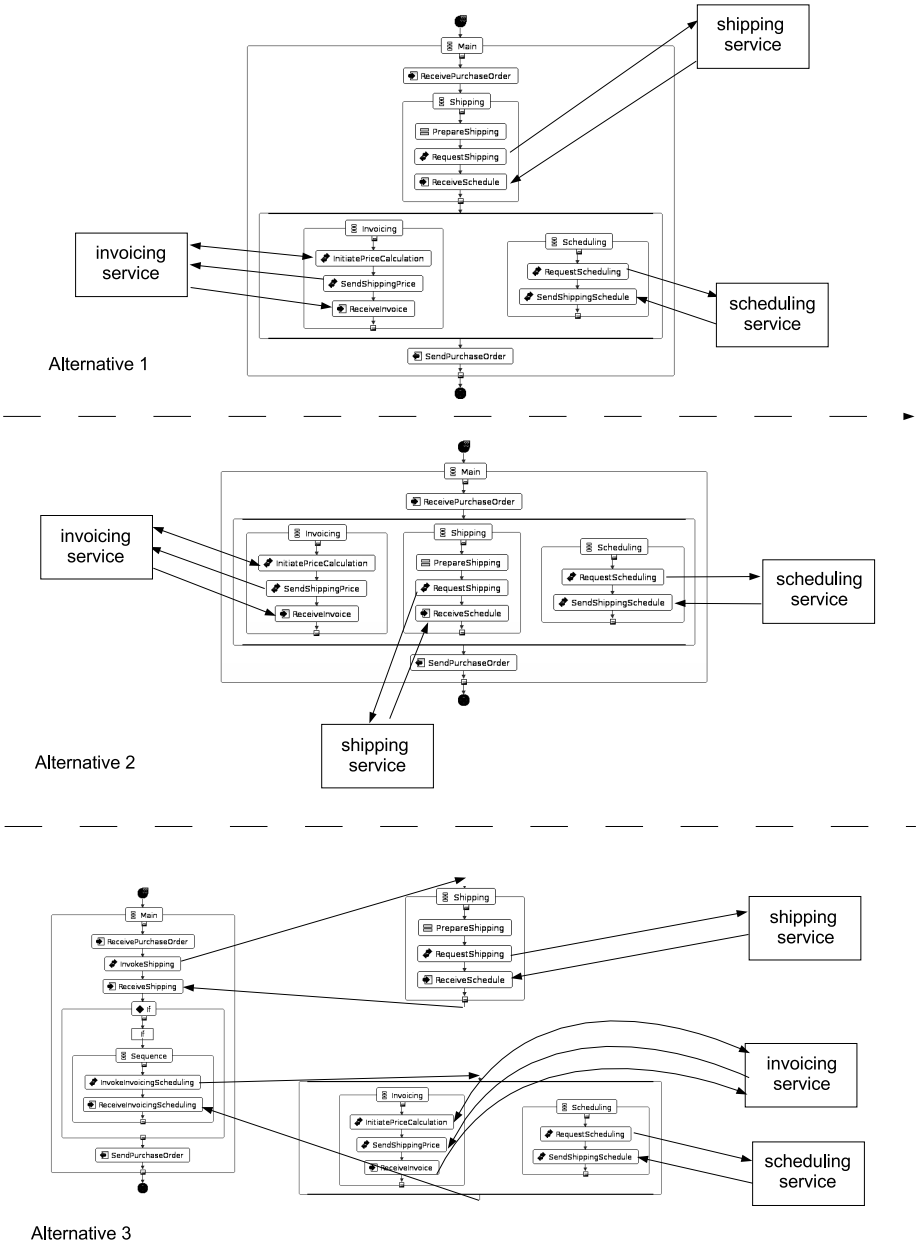

Fig. 4. Discussed alternatives for the reference process

That gives $R_{RP} = 21ms$.
For alternative (1) average response time is:

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

the difference between alternative (1) and reference process is that invoice and scheduling services are requested parallelly, so response time of parallel part is the maximum of response times of invoicing and scheduling processes. When we assume that $R_{BPELA1} = R_{BPELRP}$ then:
$R_{A1} = 17ms$.
Finally alternative (3) average response time is given by:

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

that yields: $R_{A3} = 25ms$

4.5 Alternatives Evaluation – Reusability

Total number of interfaces provided by the considered system (i.e. process and acompanying services) has been used as a reusability metric (the more interfaces the higher reusability). Reference process and alternative (1) deliver four interfaces: one to the purchase process and three to the elementary services: invoicing, shipping and scheduling. Alternative (3) delivers six interfaces: three to the elementary services, one to the composite service (process) and two new interfaces to two subservices.
All the above data is gathered in table 2.

Table 2. Quality metrics for the reference process and its 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

4.6 Alternatives Selection

There is a trade-off between the two considered quality attributes: systems consisting of more basic services are more reusable at the expense of performance and vice versa. The designer has to decide according to assumed design preferences: when reusability is prefered – alternative (3) should be chosen, otherwise when performance is the most important attribute – alternative (1) should be preferred.

5 Summary

The paper presents a transformational approach to the design of electronic business processes denoted in BPEL. The approach has been founded on a novel concept of business processes equivalence, which makes possible to construct business processes by their gradual transformations. Processes resulting from those transformations can be formally verified against their specification as well as against typical properties of concurrent systems like livenes or reachability. The designer, who steeres the transformations by evaluating non-functional attributes, is informed either that transformed process meets predefined requirements or which parts of process' behaviour has changed. He can accept or reject such a non-equivalent transformation, beeing also assisted by supporting tools. The usefulness of our approach has been presented on a not-trivial example. The directions for tool support development have also been provided, some of these tools have already been implemented: e.g. BPEL to LOTOS transformation tool.

Further research can include development of predefined transformations that have been proved to preserve *a priori* properties like process equivalence as well as the development of a business process design environemnt (software tools) integrating all the tools needed to support the presented approach. The challenge in building such tools is to make one consistent process out of all the separated method steps and to build an integrated development environment. Integrated tool can be built as an extension of one of the open-source software development environments like Eclipse or NetBeans.

References

1. Binkley, D., Gallagher, K.B.: Program slicing. Advances in Computers 43, 1–50 (1996)
2. Bolognesi, T., Brinksma, E.: Introduction to the iso specification language lotos. Comput. Netw. ISDN Syst. 14(1), 25–59 (1987)
3. Koshkina, M., Breugel, F.: Models and verification of bpel (2006)
4. Cámara, J., Canal, C., Cubo, J., Vallecillo, A.: Formalizing wsbpel business processes using process algebra. Electr. Notes Theor. Comput. Sci. 154(1), 159–173 (2006)
5. Cleaveland, R., Smolka, S.: Process algebra (1999)
6. Cleaveland, R.: Concurrency workbench of the new century (2000), `http://www.cs.sunysb.edu/~cwb/`
7. Fahland, D., Reisig, W.: Asm-based semantics for bpel: The negative control flow. In: Abstract State Machines, pp. 131–152 (2005)
8. D'Ambrogio, A., Bocciarelli, P.: A model-driven approach to describe and predict the performance of composite services. pp. 78–89 (2007)
9. Ferrara, A.: Web services: a process algebra approach. In: ICSOC 2004: Proceedings of the 2nd International Conference on Service Oriented Computing, pp. 242–251. ACM Press, New York (2004)
10. Ferrara, A.: Web services: a process algebra approach, pp. 242–251 (2004)
11. Foster, H., Kramer, J., Magee, J., Uchitel, S.: Model-based verification of web service compositions. In: 18th IEEE International Conference on Automated Software Engineering, ASE (2003)

12. Foster, H., Uchitel, S., Magee, J., Kramer, J., Hu, M.: Using a rigorous approach for engineering web service compositions: A case study. In: SCC 2005: Proceedings of the 2005 IEEE International Conference on Services Computing, pp. 217–224. IEEE Computer Society, Washington, DC (2005)
13. Fu, X., Bultan, T., Su, J.: Analysis of interacting bpel web services. In: WWW 2004: Proceedings of the 13th International Conference on World Wide Web, pp. 621–630. ACM, New York (2004)
14. Hofacker, I., Vetschera, R.: Algorithmical approaches to business process design. Computers & OR 28(13), 1253–1275 (2001)
15. Holzmann, G.J.: The SPIN Model Checker: Primer and Reference Manual. Addison-Wesley Professional, Reading (2003)
16. Martens, A.: Simulation and equivalence between bpel process models (2005)
17. Martin, F.: Refactoring: improving the design of existing code. Addison-Wesley Longman Publishing Co., Inc., Boston (1999)
18. Seattle University Everald E. Mills. Software metrics, SEI-CM-12-1.1 (1988)
19. Moore, I.: Automatic inheritance hierarchy restructuring and method refactoring, pp. 235–250 (1996)
20. Opdyke, W.F.: Refactoring Object-Oriented Frameworks. PhD thesis, Urbana-Champaign, IL, USA (1992)
21. Ouyang, C., Verbeek, E., van der Aalst, W.M.P., Breutel, S., Dumas, M., ter Hofstede, A.H.M.: Formal semantics and analysis of control flow in ws-bpel. Sci. Comput. Program. 67(2-3), 162–198 (2007)
22. Ratkowski, A., Zalewski, A.: Performance refactoring for service oriented architecture. In: ISAT 2007: Information Systems Architecture And Technology (2007)
23. Hinz, S., Schmidt, K., Stahl, C.: Transforming BPEL to Petri Nets. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) BPM 2005. LNCS, vol. 3649, pp. 220–235. Springer, Heidelberg (2005)
24. Salaün, G., Bordeaux, L., Schaerf, M.: Describing and reasoning on web services using process algebra, p. 43 (2004)
25. Salaün, G., Ferrara, A., Chirichiello, A.: Negotiation among web services using LOTOS/CADP. In (LJ) Zhang, L.-J., Jeckle, M. (eds.) ECOWS 2004. LNCS, vol. 3250, pp. 198–212. Springer, Heidelberg (2004)
26. Reisig, W.: Modeling- and Analysis Techniques for Web Services and Business Processes. In: Steffen, M., Tennenholtz, M. (eds.) FMOODS 2005. LNCS, vol. 3535, pp. 243–258. Springer, Heidelberg (2005)
27. W3C. Xsl transformations (xslt) version 1.0 (1999), `http://www.w3.org/tr/xslt`
28. Yang, Y., Tan, T., Yu, J., Liu, F.: Transformation bpel to cp-nets for verifying web services composition. In: Proceedings of NWESP 2005, p. 137. IEEE Computer Society, Washington, DC (2005)