Transforming Internal Activities of Business Process Models to Service Compositions*

Teduh Dirgahayu[1], Dick Quartel[2] and Marten van Sinderen[1]

[1] Centre for Telematics and Information Technology
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands

[2] Telematica Instituut, P.O. Box 589, 7500 AN Enschede, The Netherlands

Abstract. As a service composition language, BPEL imposes as constraint that a business process model should consist only of activities for interacting with other business processes. BPEL provides limited support for implementing internal activities, i.e. activities that are performed by a single business process without involvement of other business processes. BPEL is hence not suitable to implement internal activities that include complex data manipulation. There are a number of options to make BPEL able to implement such internal activities. In this paper we analyse those options based on their feasibility, efficiency, reusability, portability and merging. The analysis indicates that delegating internal activities' functionality to other services is the best option. We therefore present an approach for transforming internal activities to service invocations. The application of this approach on a business process model results in a service composition model that consists only of activities for interaction.

1 Introduction

Web services [1] has become a popular platform on which many enterprises execute their business processes [2, 3]. BPEL [4, 5] is a de facto language for implementing a business process as a composition of Web services. Moreover the approach defined in Model-Driven Architecture [6, 7], especially regarding automatic transformation, is widely used and investigated to speed up the implementation of business processes in BPEL, e.g. in [8, 9, 10, 11, 12].

A business process model of an enterprise may consist of two kinds of activities: (i) activities that are performed to interact with other business processes, e.g. to send and receive messages, and (ii) activities that are performed without involvement of other business processes. We call the latter kind of activities *internal activities* [13]. Internal activities are not exposed to business processes.

As a service composition language, on the one hand, BPEL (version 1.1 and 2.0) provides constructs to implement activities for interacting with other business processes, e.g. *receive*, *reply*, and *invoke*. On the other hand, BPEL provides limited

* This work is part of the Freeband A-MUSE project (http://a-muse.freeband.nl) sponsored by the Dutch government under contract BSIK 03025.

Dirgahayu T., Quartel D. and van Sinderen M. (2008).
Transforming Internal Activities of Business Process Models to Service Compositions.
In *Joint Proceedings of the 5th International Workshop on Ubiquitous Computing (IWUC 2008) 4th International Workshop on Model-Driven Enterprise Information Systems (MDEIS 2008) 3rd International Workshop on Technologies for Context-Aware Business Process Management (TCoB 2008)*, pages 52-63
DOI: 10.5220/0001736800520063

support for implementing internal activities. By default, BPEL uses XPath 1.0 [14] for data manipulation. XPath is a powerful language for querying the contents of XML documents; but it provides limited support for data manipulation, e.g. it supports only simple arithmetic, boolean and string manipulation. Therefore, BPEL and XPath are not suitable to implement internal activities for complex data manipulation. Consequently, a business process model which is targeted to be implemented in BPEL should not contain internal activities for complex data manipulation.

We believe that a business process should not be constrained by such a limitation. A business process model should be allowed to contain internal activities for simple and complex data manipulation. To implement such a business process model, a transformation is required to map the business process model onto a chosen target platform and implementation language. In our case, the transformation has to be able to implement the activity in BPEL.

To overcome the aforementioned limitation, a number of options have been introduced to make BPEL able to implement internal activities with any degree of complexity. An option can be a non-standard or proprietary extension to BPEL [15, 16, 17] or a design method [8, 11, 12, 18].

The objectives of this paper are (i) to analyse available options for implementing internal activities in BPEL and (ii) to present an approach for transforming internal activities to implementation to facilitate the best option. We illustrate this approach using an example business process that is modelled in a language of our convenience (i.e., ISDL [19]). We claim however that the approach is generally applicable to other modelling languages.

The structure of this paper is as follows. Section 2 describes the roles of activities, constraints and functions in a business process model. Section 3 analyses several options for implementing internal activities in BPEL. Section 4 presents an approach for transforming internal activities to facilitate the best option. Finally, section 5 presents our conclusions and indicates future work.

2 Internal Activities, Constraints and Functions

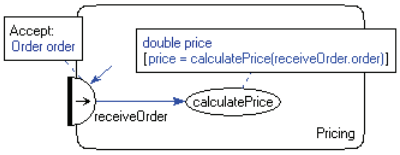
A business process model consists of activities. For each activity, only its result is considered. The business process model abstracts from the way the activities establish their results. Constraints are used to specify the possible results that can be or should be established.

A simple constraint can be easily included in a business process model and leaves the business process model easy to understand. Inclusion of a complex constraint potentially makes the business process model difficult to understand. To avoid that, some details of a complex constraint can be encapsulated in a (parameterised) function. The details of the function are described or specified in other documents associated with the business process model. In this way, a complex constraint can be included in the business process model as a simple constraint calling a function.

Fig. 1(a) shows an example of the behaviour of an (incomplete) business process model *Pricing*. The business process receives an order via an activity *receiveOrder* and then calculates the total price of the received order by performing an internal activity *calculatePrice*. The result of the activity *calculatePrice* is constrained such

that it must be equal to the result of function *calculatePrice()* with the received order as a parameter. Fig. 1(b) shows the specification of the function *calculatePrice()*.

In the figure, a behaviour is represented by a rounded rectangle. An internal activity is represented by an ellipse. An activity for interaction is represented by a segmented ellipse located at the border of the behaviour rectangle. An arrow inside this segmented ellipse indicates the direction in which the message flows. The result of an activity is specified in a box attached to the activity. If this result is constrained, its constraint is specified within brackets in the same box. The keyword *Accept* in the box attached to an activity for interaction indicates that the text following the keyword is the message accepted by the activity.



(a) Function call in the constraint of activity result

$$\sum_{k=1}^{n} quantity_k \times price_k$$

(b) Function *calculatePrice()*

Fig. 1. Example of a business process model.

Functions can also be called in the constraints of other types of model elements. For example, Fig. 2 depicts a business process that first receives an order and then makes a choice between an activity *handleOrderFromNewClient* or *handleOrder FromExistingClient*. The choice is represented by a diamond symbol. This choice is constrained by the result of a function *fromNewClient()* that evaluates whether an order is from a new client. This should not be confused with a "switch" at some implementation language.

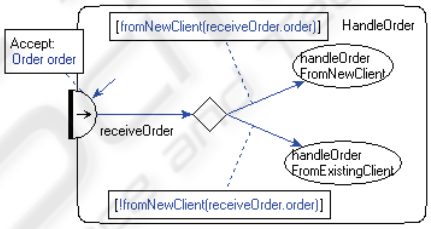


Fig. 2. Function *fromNewClient()* is called in a choice's constraints.

A function call in the constraint of a model element other than an internal activity implicitly defines an internal activity whose result is constrained by the result of that function. This internal activity can be made explicit and the constraint of the model element refers to the result of this internal activity. For example, the behaviour of the business process model in Fig. 3 is equivalent to the model in Fig. 2. This style of modelling is useful, e.g., to allow the result of a function be re-used in multiple model elements.

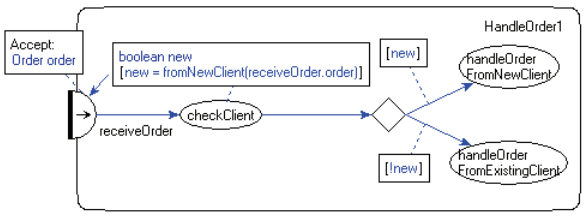


Fig. 3. Function *fromNewClient()* is now called in an internal activity.

From a different point of view, the specification of the function can be seen as the way the internal activity establishes its result. For example, the function specification in Fig. 1(b) can be seen as the way the activity *calculatePrice* in Fig. 1(a) establishes its result. Therefore, the implementation of an internal activity can be done by implementing the specification of the function.

Since a business process model abstracts from the way the results of its internal activities are established, a sole transformation from a business process model to a business process implementation will not result in an executable implementation. To obtain an executable implementation, the description or specifications of the functions called in the constraints of the internal activities should also be transformed into implementations. This transformation results in so-called *function implementations*. During execution, the business process implementation uses the function implementations to establish the results of its internal activities.

3 Options for Implementing Internal Activities

In this section we analyse available options for implementing internal activities based on a number of evaluation criteria.

3.1 Evaluation Criteria

As mentioned previously, a transformation of a business process model to an executable implementation should result in two kinds of implementations: (i) a business process implementation and (ii) function implementations. In our case, the business process implementation is in BPEL and the function implementations are in some implementation language that can also be BPEL and XPath.

While the business process implementation can be obtained through an automatic transformation [9, 10, 11, 12], function implementation is mainly obtained through manual transformation regardless of the options. To our knowledge, there is not yet an effective and efficient way to automatically transform a function description or specification. These kinds of implementations can be in the same or different implementation artefacts. An additional transformation might be required to merge those implementation artefacts into a single implementation artefact.

We analyse the options based on the following criteria.

- *Feasibility*: What support is available for implementing functions?

- *Efficiency*: What is the execution efficiency of a function call?
- *Reusability*: Can function implementations be reused by other business processes?
- *Portability*: Is the business process implementation portable between different BPEL servers?
- *Merging*: Is the merging of the business process implementation and function implementations required?

3.2 Options

We identify the following options for implementing internal activities.

Option 1: BPEL and XPath
A function specification is implemented in BPEL and XPath as part of the business process implementation. BPEL provides a construct *assign* for assigning a value to a variable. This value is created or obtained using XPath expressions. Structured activity constructs, e.g. *while* and *switch*, can also be used in data manipulation. This option uses the standard BPEL.

- *Feasibility*: BPEL structured activity constructs and XPath provide limited support for implementing function specifications, especially the complex ones.
- *Efficiency*: Function implementations are executed in the same execution instance as the business process execution instance. Overhead in calling a function is low; hence the execution efficiency is high.
- *Reusability*: Function implementations can only be used by the business process in which the functions are implemented.
- *Portability*: The business process implementation is in standard languages, i.e. BPEL and XPath, which are supported by all BPEL-compliant servers. Therefore, the implementation is portable between different BPEL servers.
- *Merging*: The business process implementation must provide places into which function implementations can be placed. A transformation is required to merge these two kinds of implementations.

Option 2: Embedded Code
A function specification is implemented in a general-purpose implementation language and embedded in the business process implementation. This option is an extension to the standard BPEL, e.g. BPELJ [15] and Java embedding [16].

- *Feasibility*: A general-purpose implementation language, e.g. Java, typically provides full support for implementing complex function specifications.
- *Efficiency*: Function implementations are executed in the same execution instance as the business process execution instance. Overhead in calling a function is low; hence the execution efficiency is high.
- *Reusability*: Function implementations can only be used by the business process in which the function implementations are embedded.
- *Portability*: The business process implementation can only be executed on a BPEL server that supports the extension.

- *Merging*: The business process implementation must provide places into which function implementations can be embedded. A transformation is required to merge these two kinds of implementations.

Option 3: Server Functions
A function specification is implemented in a general-purpose implementation language. After compilation, the function implementation is deployed in a BPEL server on which the business process implementation is to be executed. The business process calls the function using XPath expressions. This option is a proprietary extension, e.g. custom functions [17].

- *Feasibility*: A general-purpose implementation language, e.g. Java or C#, typically provides full support for implementing complex function specifications.
- *Efficiency*: Function implementations are executed in different execution instances from the business process execution instance. A function call establishes interprocess communication between those execution instances. Overhead in calling a function is higher than the previous options; hence the execution efficiency is lower.
- *Reusability*: Function implementations can be used by other business processes on the same BPEL server. Business processes on different BPEL servers cannot use the function implementations.
- *Portability*: The business process implementation can only be executed on a BPEL server in which the function implementations are deployed. Not every BPEL server supports this extension.
- *Merging*: The business process implementation and the function implementations are deployed separately. No merging is required.

Option 4: Service Delegation
A function specification is implemented by an operation of another Web service. A function call is transformed to an operation invocation. This option transforms a business process model to a service composition model. A service composition model consists only of activities for interaction. This option is used, e.g., in [8, 11, 12, 18].

- *Feasibility*: Function specifications can be implemented in a general-purpose implementation language, e.g. Java or .NET. Such a language typically provides full support for implementing complex function specifications.
- *Efficiency*: A function call establishes interprocess communication between the business process execution instance and the Web service executing the function implementations. Potentially they run on different servers. Overhead in calling a function is higher than all the previous options; hence the execution efficiency is low.
- *Reusability*: As function implementations are presented as Web services operations, they can be used by other business processes on any BPEL servers.
- *Portability*: The business process implementation is in standard languages, i.e. BPEL and XPath, which are supported by all BPEL servers. Therefore, the implementation is portable between different BPEL servers.
- *Merging*: The business process implementation and the function implementations are deployed separately. No merging is required.

3.3 Summary

Fig. 4 shows two different paths taken by the options in transforming a business process model to an implementation. Options 1, 2 and 3 directly implement the business process model in BPEL (with extensions). Option 4 first refines the business process model into a service composition model and then implements the service composition model in BPEL.

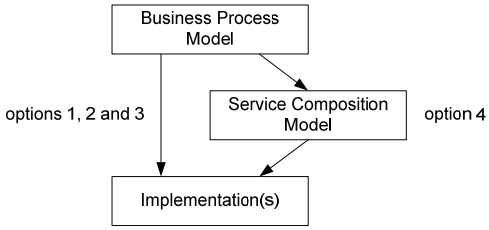


Fig. 4. Different paths taken by the options in implementing business process model.

The analysis is summarised in Table 1. Assuming that all aspects have the same weight, we conclude that option 4 is the best. To improve its efficiency, option 4 can be combined with option 1. Simple arithmetic operations can be implemented in BPEL and XPath, instead of delegating them to some Web services. For example, iteration typically uses addition or subtraction operation to increase or decrease the iteration index. Implementing these arithmetic operations in BPEL and XPath will improve the execution efficiency.

Table 1. Comparison between the options.

Criteria	**Options**			
	1. BPEL and XPath	2. Code embedding	3. Server functions	4. Service delegation
Feasibility	limited	full	full	full
Efficiency	high	high	lower	low
Reusability	no	no	limited	full
Portability	yes	no	no	yes
Merging	yes	yes	no	no

4 Transformation Approach

Our transformation approach is aimed at facilitating the implementation of internal activities using the option of service delegation. It refines a business process model into a service composition model. The approach is defined to be systematic that can be done programmatically in a transformation language. The approach is based on the idea that an internal activity can be refined into an interaction [20].

As illustration, we apply our approach to the transformation of a business process model *Invoicing* as shown in Fig. 5. This business process starts when it receives an order from a customer. The business process then checks whether the order can be

fulfilled. If so, the business process creates an invoice, sets the invoice's payment due date, and then sends the invoice back to the customer. Otherwise, the business process creates and sends back a rejection message to the customer. The keyword *Invoke* in the box attached to an activity for interaction indicates that the text following the keyword is the message sent by the activity. The message is sent by invoking some operation in another business process.

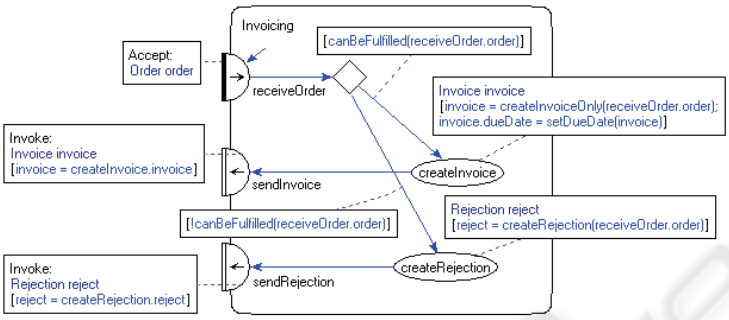


Fig. 5. A business process model to be transformed.

Step 1: Make all the Internal Activities Explicit. To make sure that the resulted service composition model consists only of activities for interaction, any implicit internal activity should be made explicit. Being explicit, all internal activities can be refined into interactions.

In this step, for each constrained model element other than an internal activity, we insert a new internal activity such that the inserted internal activity precedes the model element. We then "shift" function calls in the constraints of the model element to the constraint of the inserted internal activity. The constraints of the model element should now refer to the results of the inserted internal activity.

The application of this step results in the business process model shown in Fig. 6. We insert an internal activity *checkFulfillment* preceding the choice and "shift" the function call *canBeFulfilled* from the choice constraints to the constraint of the inserted internal activity. The choice constraints now refer to the result of activity *checkFulfillment*.

Step 2: Distribute Constraints to a set of Internal Activities. At implementation level, an activity for interaction performs a specific task, e.g. receiving a message or invoking an operation. To obtain a service composition model in which each activity for interaction invokes one Web service operation, each internal activity should be constrained by one function only. If an internal activity is constrained by several functions, these functions should be distributed over multiple internal activities.

In this step, we replace an internal activity whose constraints contain multiple function calls with a set of new internal activities. The number of the new internal activities should be equal to the number of the function calls. We then distribute the function calls such that the constraint of each internal activity contains one function call only.

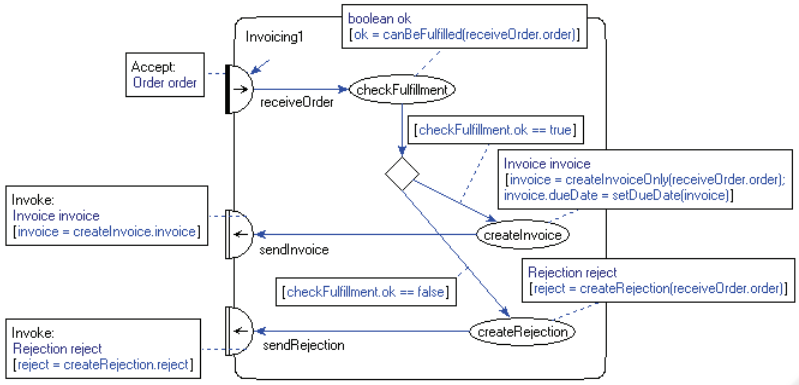


Fig. 6. Function *canBeFulfilled* is shifted to the constraint of an inserted internal activity.

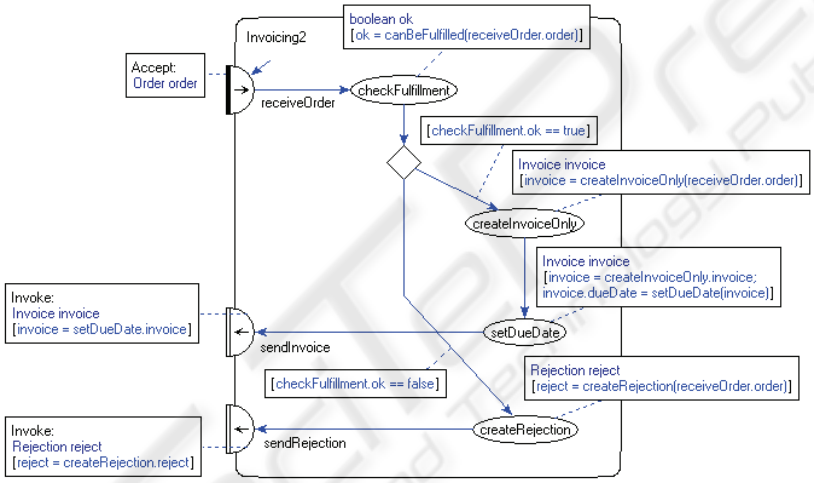


Fig. 7. Function calls are distributed to a set of internal activities.

The new internal activities can be structured in sequence, parallel or combination of both. To determine the correct structure, the dependency between function calls should be considered. For example, if the output of a function *X* becomes the input of another function *Y* in the original activity, the new activities should be structured in sequence such that the activity that calls function *X* precedes the activity that calls function *Y*. If the function calls are independent from each other, the activities can be structured in parallel.

The application of this step results in the business process model shown in Fig. 7. The constraints of activity *createInvoice* of Fig. 6 contains two function calls, i.e. *createInvoiceOnly()* and *setDueDate()*. In Fig. 7, we replace this activity with two activities *createInvoiceOnly* and *setDueDate*; and then distribute the function calls over the constraints of those activities correspondingly. To maintain the dependency between the function calls, we structure those activities in sequence.

Step 3: Refine Internal Activities into Interactions. Finally, we structure the business process model into a service composition model by introducing one or more supporting services and then refining internal activities into interactions between the business process and the supporting services. The supporting services can be provided by the enterprise which owns the business process or by other service providers, e.g. trusted business partners. The supporting services are responsible for implementing the function specifications. In this way, we delegate function implementations to the supporting services.

The application of this step results in the service composition model shown in Fig. 8. We introduce a supporting service *InvoicingSupport* and refine each internal activity into a request/response interaction between the business process and the supporting service. An interaction is represented as two segmented ellipses connected with each other. On the business process' side, the request and response are indicated by the keywords *Invoke* and *Return*, respectively. On the supporting service's side, the request and response are indicated by the keywords *Accept* and *Reply*, respectively. This model can be transformed to a BPEL implementation [21].

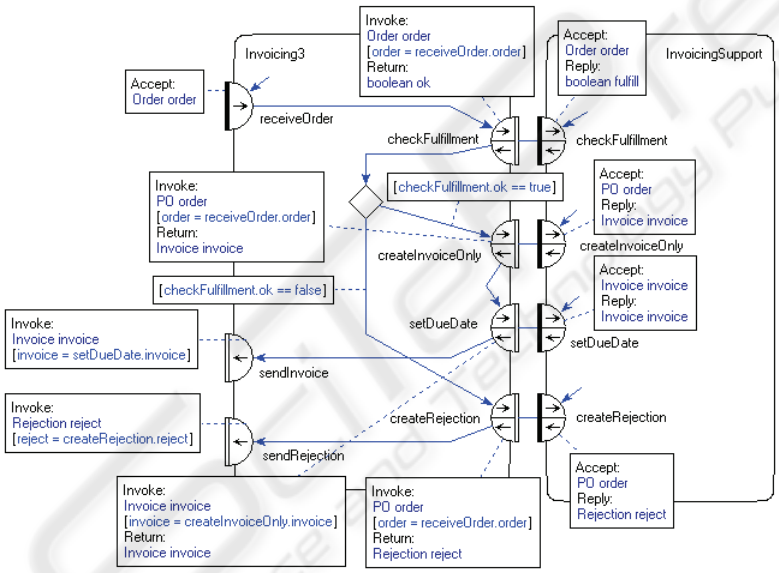


Fig. 8. Service composition model.

5 Conclusions

We have analysed some available options for implementing internal activities of business process models in BPEL. The analysis indicates that service delegation is the best option. To improve its execution efficiency, this option should be combined with the transformation of simple arithmetic operations to BPEL and XPath expressions. We have then presented an approach for transforming internal activities to service delegations. The application of the approach on a business process model results in a service composition model that consists only of activities for interaction. To be a

complete transformation from business process models to implementation, a transformation based on the approach should be complemented with a transformation from service composition models to implementations.

Our approach is originally developed to transform a business process model to a service composition model that is targeted to be implemented in BPEL. Since the resulted service composition model does not contain any BPEL-specific information, the model can be implemented in other service composition languages (not necessarily on Web services platform), e.g. as listed in [22]. For each service composition language, however, a similar analysis as presented in this paper might be necessary to evaluate whether service delegation is the best option among possible options for that language.

Our transformation approach is systematic and can be done programmatically. We have implemented the approach in QVT [23] for models that are developed based on a simple metamodel. Each step is implemented as an individual transformation specification, namely Step1, Step2 and Step3 that respectively correspond to the steps in the transformation approach. The Step1 transformation is applied to a given business process model. The Step2 transformation is applied to the output of the Step1 transformation. The Step3 transformation is applied to the output of the Step2 transformation. The output of the Step3 transformation is a service composition model as the final result of our transformation approach. In the future, we will implement the approach as part of a transformation that we have developed to transform business process models in ISDL to implementations in BPEL.

References

1. W3C. Web Service Architecture. W3C Working Group Note (2004)
2. Erasala, N., Yen, D.C., Rajkumar, T.M.: Enterprise Application Integration in the electronic commerce world. Computer Standards and Interface 25 (2002) 69-82
3. Medjahed, B., Benatallah, B., Bouguettaya, A., Ngu, A.H.H., Elmagarmid, A.K.: Business-to-business interactions: issues and enabling technologies. VLDB Journal 12 (2003) 59-85
4. BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems: Business Process Execution Language for Web Services version 1.1 (2003)
5. OASIS: Web Services Business Process Execution Language Version 2.0 (2007)
6. OMG: Model Driven Architecture (MDA). ormsoc/01-07-01 (2001)
7. OMG: MDA Guide Version 1.0.1. omg/03-06-01 (2003).
8. Koehler, J., Hauser, R., Kapoor, S., Wu, F.Y., Kumaran, S.: A model-driven transformation method. In Proc. of 7th IEEE Intl. Enterprise Distributed Object Computing Conf. (2003) 186-197
9. Kath, O., Blazarenas, A., Born, M., Eckert, K.-P., Funabashi, M., Hirai C.: Towards executable models: transforming EDOC behavior models to CORBA and BPEL. In Proc. of 8th IEEE Intl. Enterprise Distributed Object Computing Conf. (2004) 267-274
10. Dirgahayu, T.: Model-Driven Engineering of Web Service Compositions: A Transformation from ISDL to BPEL. MSc. Thesis. University of Twente, Enschede (2005)
11. Bordbar, B., Staikopoulos, A.: On Behavioural Model Transformation in Web Services. LNCS 3289 (2005) 667-678
12. Korherr, B., List, B.: Extending the UML 2 Activity Diagram with Business Process Goals and Performance Measures and the Mapping to BPEL, LNCS 4231 (2006) 7-18

13. Quartel, D., Dijkman, R., van Sinderen, M.: Methodological Support for Service-oriented Design with ISDL. In Proc. of 2nd Intl. Conf. on Service Oriented Computing (2004), 1-10
14. W3C. XML Path Language (XPath) Version 1.0. W3C Recommendation (1999)
15. BEA Systems, Inc., IBM Corp.: BPELJ: BPEL for Java (2004)
16. Oracle Corp.: Oracle BPEL Process Manager. http://www.oracle.com/technology/products/ias/bpel/index.html
17. Active Endpoints, Inc.: ActiveBPEL Engine 2.0. http://www.active-endpoints.com/active-bpel-engine-overview.htm
18. OMG: Business Process Modeling Notation Specification. dtc/06-02-01 (2006)
19. ASNA. ISDL Home. http://isdl.ctit.utwente.nl
20. Quartel, D., Ferreira Pires, L., van Sinderen, M.: On Architectural Support for Behaviour Refinement in Distributed Systems Design. J. Integrated Design and Process Science 6, 1 (2002) 1-30
21. Dirgahayu, T., Quartel, D., and van Sinderen, M.: Development of Transformations from Business Process Models to Implementations by Reuse. In Proc. of the 3rd Intl. Workshop on Model-Driven Enterprise Information Systems (2007) 41-50
22. van der Aalst, W.M.P., Dumas, M., ter Hofstede, A.H.M.: Web Service Composition Languages: Old Wine in New Bottles? In Proc. of 29th EUROMICRO Conference (2003) 298-305
23. OMG: Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification. ptc/07-07-07