BPEL Fragments for Modularized Reuse in Modeling BPEL Processes

Zhilei Ma and Frank Leymann
Institute of Architecture of Application Systems
Universität Stuttgart
Universitaetsstr. 38, 70569 Stuttgart, Germany
zhilei.ma@iaas.uni-stuttgart.de
frank.leymann@iaas.uni-stuttgart.de

Abstract

BPEL has been established as the standard for modeling business processes by orchestrating Web services. When modeling a BPEL process, users end up in basically three approaches: from scratch, by using a process template or a reference process, and by ad hoc modifying a existing process model to meet the current requirements. There is strong demand on a flexible and modularized approach for reusing BPEL process models. As neither the BPEL standard nor the current BPEL extensions support such a feature for reuse, we present in this paper a formal definition for BPEL fragments. In this paper we first study the related work on process fragments and distinguish BPEL fragments from existing reuse approaches in service-oriented process modeling. Based on the requirement analysis we present a definition of BPEL fragments in XML schema. By adopting our definition of BPEL fragments, the flexibility of modularized reuse of BPEL process models can be significantly increased.

Keywords: *Reuse, service orchestration, process fragment, BPEL, BPEL fragment*

1 Introduction

With the advent of service-oriented[4][7] business process management (BPM)[26][27] technologies, business process modeling gains more and more attentions and applications in different businesses. Web Services Business Process Execution Language (WS-BPEL or BPEL for short) [2] has been established as an OASIS standard for modeling executable and abstract business processes by orchestrating Web services[29]. Modeling business processes in general and modeling BPEL processes in specific could be enormous time-consuming and error-prone. Reuse has been proved as a valuable concept to avoid reinvent the wheel, take the burden of repeated work off users and improve the quality and efficiency of process modeling by allowing to exploit best-practice process models [17][20]. Based on our observation and our intensive investigation the reuse support in business process modeling is quite limited. When building a new business process, a process modeler ends up in basically three approaches: from scratch, by using a process template [15][10] or a reference process [8][9], and by ad hoc modifying a existing process model to meet the current requirements.

Process template, reference processes and existing process models provide process modelers only the possibility for reuse process models as a whole. Enabling parts of business processes that can be reused in many different places or business contexts is a desired practice based on the authors' experience and case studies reported in the literatures [3][21][23]. Modularized reuse is especially desired for complex or large business processes. To bring more dynamic and flexibility into reuse in business process modeling, we need a more modular and granular way to define and describe reusable parts of business process models, which can be identified, extracted from different process templates, reference processes and enterprise specific process models. We call such reusable parts *process fragments* and such reusable parts of BPEL processes *BPEL fragments*.

Both in academia and industry there is a lack of a formal definition for what is a BPEL fragment in terms of reuse. A formal definition on BPEL fragments is essential for different stakeholders to have an common understanding for a better communication and to understand how they could exploit BPEL fragments in process modeling. As BPEL and its current extensions do not provide such a feature for reuse in a modularized and interoperable way [25], we present in this paper a formal definition in XML schema for BPEL fragments, which enables defining reusable process logic of BPEL processes in a more flexible and modularized manner. This paper lays the cornerstone for the research work we are carrying on to realize the reuse-aware process modeling approach that we presented in our previous paper [19].

This paper is organized as follows: In section 2 we provide an overview of the related work on process fragments and distinguish BPEL fragment from other concepts for reuse in service-oriented process modeling. Then we analyze in section 3 the requirements on the definition of BPEL fragments in terms of reuse. The definition of BPEL fragments and the necessary extensions of BPEL standard are described in section 4. We close the paper with a short conclusion and an overview on our work-in-progress and future work.

2 Related Work

Decompose complex process models into smaller building blocks is an essential reuse technique. Process fragments provide a modularized view on big and complex process models. In [28] the authors follow the Divide and Conquer strategy by decomposing a process model into Single-Entry-Single-Exit fragments for analysis and verification.

Especially when modeling collaborative or decentralized processes, a single process designer does not have necessarily the complete knowledge on the whole process, but each organizational units involved has its know-how locally. In this case process fragments can be modeled in a decentralized way and combined later vertically or hierarchically to build up the whole process model [18][13].

Process fragments can not only be used for decentralized process modeling but also for decentralized process execution. [11][12] presented a static approach for distributed process execution with three major steps, i.e. identify, create, and execute BPEL process fragments, by keeping the original execution semantics of the parent BPEL process model. Another approach for distributed process execution follows an dynamic paradigm [24]. In this approach, a process is fragmented step by step during the execution of it. A process instance executes firstly the immediate tasks and then forward the reminder to the next execution engine and so on. Both approaches are extremely interesting for organizations who want dynamically outsource parts of their business processes or distribute their business process over the organizational boundaries. However, the focus of these work have not been set on reuse and none of them addresses the requirements on the reusable BPEL fragments and how it should be defined.

Generally speaking, the two major goals of Web services are reuse and interoperability [7]. Web services provide two reuse styles. For the black box reuse style, users require little or no knowledge about the internal logic and the implementation of a Web services [29]. Some Web services provide users the possibilities to customize or parameterize the services to suit the current usage context [5] [6]. Such adaptive Web services belong to the gray box reuse style. While Web services is programming in the small, BPEL fragments can be considered as programming in the large [16][30]. In contrast to Web services BPEL fragments act as a container for reusable process logic of orchestration of Web services at the process level. One may argue that a process can be decomposed into coherent parts which can be exposed as Web services that can be invoked by an activity in a BPEL process. However, a BPEL fragment may not be semantically complete or syntactical compliant with the BPEL standard. Therefore, a BPEL fragment may not necessarily be executable and be able to be exposed as a Web service. For example, a user may only want to define a collection of `<catch>` constructs as a reusable BPEL fragment, which describes the fault handling logic for a set of faults that should be collectively handled in a certain business context.

In [25] the author analyzed three possible approaches for modularized ruse of BPEL processes and came to the conclusion that sub-process seems to be the more general approach to solve this issue. BPEL-SPE [14] is a BPEL extension that allows users to define a subset of BPEL process as a sub-process, which can be invoked by the same or another BPEL process. Sub-processes in general are executable and may carry different grade of autonomy against their parent processes. For design reusable BPEL process logic we need a more flexible approach, which should allow the defined BPEL fragments to be incomplete. With incompleteness we mean that a BPEL fragment may be syntactically incomplete or be designed as a template or pattern with placeholders that must be refined later.

Generally speaking, a pattern provides a means for abstracting and tackling recurring problems in a give domain [1]. The workflow patterns [22] aim to lay the groundwork and establish an common understanding on the requirements on process modeling languages and process-aware information systems. In addition they provide an means for evaluating the expressiveness of various business process modeling languages and the capabilities of diverse workflow management systems. They can be considered as programming constructs in the conventional programming languages such as Java. In contrast to workflow patterns, process fragments provide a means for users to capture domain-specific or individual business patterns for recurred business problems or to save repeated process modeling work. When defining such process fragments, users may use the established workflow patterns to model the control flow, data, resource and exception handling logic involved in the process fragments. A process fragment can be considered as a subroutine that has been developed by using the programming constructs in a programming language.

3 Requirements on BPEL Fragments

A BPEL fragment can essentially be considered as a container for logically related process activities that bear high reusability regards to future process modeling. The term process activities do not only refer to the activities used for describing normal process behavior but also the activities and BPEL constructs used for specifying process behavior for exception, compensation, and event handling. Whether a certain part of existing BPEL process are reusable depends strongly on the application domains of the BPEL processes as well as results yield after the planning and analysis activities [19]. To give the business analysts the flexibility for modeling reusable BPEL fragments, we need a new construct for encapsulating reusable process logic. In this section we analyze which requirements a BPEL fragment should meet in terms of reuse.

Basic Activities as Atomic Units (R1)
Activities are the basic elements in BPEL for describing process logic. BPEL activities can be divided into two classes: basic and structured activities. Basic activities specifies elemental steps of the process behavior, e.g. interacting with Web services, mapping data, throw exception, delay process execution, terminate process execution, and specify domain-specific elemental behavior by defining extension activities. Basic activities can be orchestrated together by using structured activities, which defined the possible control logic of the enclosed activities. Structured activities can be nested recursively to encode more complex and sophisticated control logic.
Basic activities are atomic units in BPEL for describing process logic. As a BPEL fragment acts as a container for reusable process logic, a BPEL fragment must contain at least one BPEL basic activity, regardless how deep the basic activity has been nested in other BPEL constructs, such as structured activities. Without a BPEL basic activity a structured activity is just a structure template without any process logic in it, regardless whether it contains further structured activities or scopes. Definitions on partner links, variables, correlation sets and so on define the context of a BPEL process. They may also represent reusable elements, but they do not encode the operative process logic directly. Therefore, they should not be considered as a valid BPEL fragment.

Flexible Granularity (R2)
In addition to structured activities, users can use BPEL scope to enclose activities, which influences the execution behavior of the activities in it. Within the scope one can define handling logic for compensating executed process logic, catching and processing errors, consuming events , and terminating the parent process instance.
When breaking BPEL process apart, it is obvious that all kinds of BPEL artifacts, range from basic activities including extension activities, simple or nested structured activities, to scope with or without special handling logic attached, could represent reusable process logic in a certain domain. In addition, users may want to define special handling logic or a subset of it, such as fault handlers for processing standard faults, compensation handlers for well-defined compensation logic for all sales processes, as reusable BPEL fragments. Therefore, a BPEL fragment should provide the flexibility for users to define reusable BPEL fragments in any granularity. The reasonable granularities we have identified are: basic activity, structured activity, scope, fault handler, compensation handler, event handler, termination handler, and any combination of the four types of BPEL handlers. The last case allows users just to define the complex fault, compensation, event and termination logic for a certain business context and to reuse them just by attaching these handlers to a scope in process modeling.

Process Context (R3)
Different types of BPEL constructs mentioned above are used in BPEL to specify stateful interactions with different partners. The semantic and the state of the activities depend on the process context that can be defined at the process or scope level in a BPEL process. The process context includes variables, partner links, message exchanges, correlation sets, extension definitions, and imported documents. A BPEL fragment should contain also information on process context for the process logic defined in it. On one side, process context makes a BPEL fragment semantically more complete. Including behavioral context help users in discovering and reusing BPEL fragments by getting a better understanding what the BPEL fragment encodes. On the other side, when reusing a BPEL fragment for modeling new BPEL processes definitions on process context can also be reused without having to define them again, thus makes reuse of BPEL fragments more comfortable for users.

Variability Points (R4)
A BPEL fragment may be used as a template or a business pattern that provide users the possibility to customize at certain points. For example, a full specified BPEL fragment may contains specific process logic that varies from business context to business context. Such non-agonistic process logic reduces the reusability of a BPEL fragment. To increase the reusability of the BPEL fragment such specific activities can be replace by placeholders, which indicate that the non-agnostic process logic has been removed. To increase the insight in such placeholders and provide users guidelines on how to fill out such placeholders, the eliminated process logic may be kept as annotations or documentations, which can be displayed to users in a graphical modeling tool to show the original content. Placeholders can not only be applied for BPEL activities but also for attributes. For example, users may want to leave out the at-

tribute `operation` of an `<invoke>` activity and specify it later by reuse. The definition of BPEL fragments should allow users to define such variability points in a BPEL fragment.

Unbroken Control Flow (R5)
BPEL is about orchestration of Web services. Control flow defines the logical relations and execution orders of the involved Web services in a BPEL process. When designing BPEL fragments, activities should be kept connected in the target BPEL fragments, which eases the discovery and reuse of BPEL fragments. When discovering BPEL fragments, users may want to query on structural information on BPEL fragments stored in a reuse repository. An unbroken control flow makes the discovery process more easier and reduces the runtime computing complexity, as the query processor does not have to consider all possible combinations of the discrete elements in a BPEL fragment. When reusing a BPEL fragment with an unbroken control flow users do not have to figure out how to stitch the discrete elements in a BPEL fragments together, which could be a time-consuming and error-prone task and could reduce the benefits should be gained by reuse.

4 BPEL Fragments

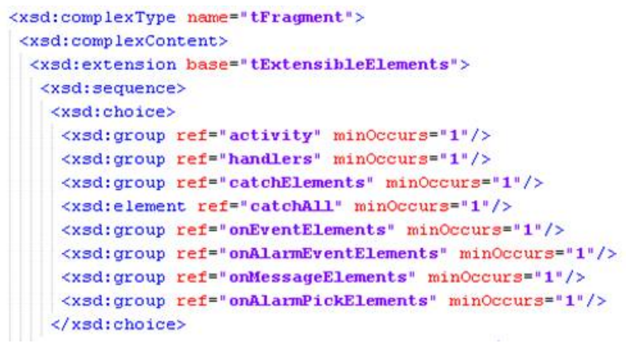
Based on the identified requirements, we present in this section a formal definition on BPEL fragments in XML schema. Our definition extends the XML schema for abstract BPEL processes, as it provides all BPEL constructs as well as opaque tokens as placeholders for BPEL activities and attributes(R4). Except changes that have been mentioned in this section, all the definitions of other constructs are as defined in the XML schema for abstract BPEL processes. To design the content model for BPEL fragments, we have made some essential changes on the XML schema for abstract BPEL processes and they are:

- Change the target namespace to http://www.iaas.uni-stuttgart.de/wsbpel/2.0/process/fragment

- Change the default namespace to http://www.iaas.uni-stuttgart.de/wsbpel/2.0/process/fragment

- Change ”xsd-derived” from XSD NS to the BPEL Fragment NS to http://www.iaas.uni-stuttgart.de/wsbpel/2.0/process/fragment

We define a new construct named `<fragment>`, which acts as the root element of a BPEL fragment and inherits the standard attributes of the `<process>` construct including *name*, *targetNamespace*, *queryLanguage*, *expressionLanguage*, *suppressJoinFailure*, and *exitOnStandardFault*. The attribute *abstractFragmentProfile* points to the usage profile, which defines the guidelines on how to resolve the opaque tokens in the BPEL fragment. The `<fragment>` construct also enables defining behavioral context (R3) on variables, partner links, message exchanges, correlation sets, extension definitions, and imported documents. The `<choice>` element in Figure 1 defines the syntax of defining the process logic of a BPEL fragment. It allows users to define BPEL fragments in a flexible granularity (R2) and results in different types of BPEL fragments.



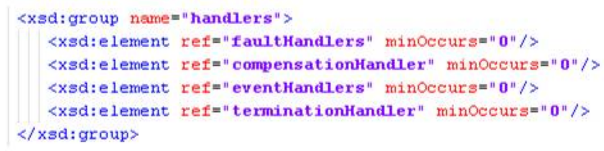
```
<xsd:complexType name="tFragment">
 <xsd:complexContent>
  <xsd:extension base="tExtensibleElements">
   <xsd:sequence>
    <xsd:choice>
     <xsd:group ref="activity" minOccurs="1"/>
     <xsd:group ref="handlers" minOccurs="1"/>
     <xsd:group ref="catchElements" minOccurs="1"/>
     <xsd:element ref="catchAll" minOccurs="1"/>
     <xsd:group ref="onEventElements" minOccurs="1"/>
     <xsd:group ref="onAlarmEventElements" minOccurs="1"/>
     <xsd:group ref="onMessageElements" minOccurs="1"/>
     <xsd:group ref="onAlarmPickElements" minOccurs="1"/>
    </xsd:choice>
```

Figure 1. A snippet of XML schema for BPEL fragment

The `<activity>` construct allows define atomic and composite BPEL fragments. An *atomic BPEL fragment* or *simple BPEL fragment* contains one single basic activity. As the name tells, an atomic BPEL fragment is the most granular BPEL fragment and can not be decomposed further into more granular BPEL fragments. A *composite BPEL fragment* comprises of a structured activity as its direct child, which specifies the control flow of a collection of activities.

A fault handler does not only catch process exceptions, in case of transactional processing they also invoke a compensation handler to rollback the transactions. The behavior of the fault handler of a scope begins by In another case, an event handler may need a fault handler to clear possible exceptions that may occur during the event processing. In such cases, users may want to group logically dependent handlers together so that they can be coherently reused in another process context. For that reason, we introduce a new construct `<handlers>` into the definition of BPEL fragments. As showed in Figure 2 the construct `<handlers>` is defined as a container for `<faultHandlers>`, `<compensationHander>`, `<eventHandlers>`, and `<terminationHandler>`. The cardinality of each handlers ensures a flexible combination of the handlers and enforces that each handler, as for a *scope* activity, can maximal appear once.

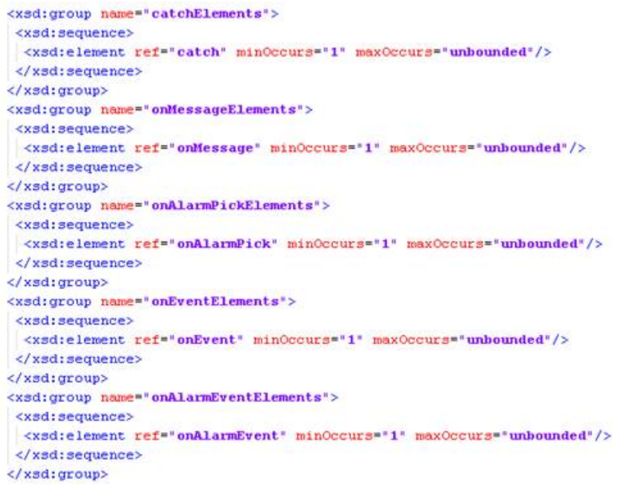
So far we have decomposed BPEL into scope, structured activities, basic activities, and handlers. To achieve more flexible granularity three BPEL constructs can be fur-



```
<xsd:group name="handlers">
    <xsd:element ref="faultHandlers" minOccurs="0"/>
    <xsd:element ref="compensationHandler" minOccurs="0"/>
    <xsd:element ref="eventHandlers" minOccurs="0"/>
    <xsd:element ref="terminationHandler" minOccurs="0"/>
</xsd:group>
```

Figure 2. XML schema for handler BPEL fragments

ther decomposed. A `<pick>` activity consists of one single `<onMesssage>` event and one or more `<onAlarm>` events. An `<onMessage>` event defines the processing logic for when a certain type of message has been received by the process. The processing logic can be eventually reused whereas this type of message needs to be processed in a process. Analogously we can group a logically coherent `<catch>` constructs to build a reusable BPEL fragment. To provider users this kind of flexibility, a `<fragment>` could be designed as a group of `<catch>` constructs, a group of `<onMessage>` events, a group of `<onAlarmPick>` events, a group of `<onEvent>` events, and a group of `<onAlarmEvent>` events. As the `<onAlarm>` event in the `<pick>` activity and the `<onAlarm>` in the `<eventHandlers>` are both defined locally in BPEL, we lift them to first class elements in the definition of BPEL fragments, so that we can use them directly as child elements of a BPEL fragment. These two elements are respectively renamed to `<onAlarmPick>` as well as `<onAlarmEvent>`, in order to distinguish them from each other. Figure 3 illustrates the definitions.



```
<xsd:group name="catchElements">
 <xsd:sequence>
  <xsd:element ref="catch" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:group>
<xsd:group name="onMessageElements">
 <xsd:sequence>
  <xsd:element ref="onMessage" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:group>
<xsd:group name="onAlarmPickElements">
 <xsd:sequence>
  <xsd:element ref="onAlarmPick" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:group>
<xsd:group name="onEventElements">
 <xsd:sequence>
  <xsd:element ref="onEvent" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:group>
<xsd:group name="onAlarmEventElements">
 <xsd:sequence>
  <xsd:element ref="onAlarmEvent" minOccurs="1" maxOccurs="unbounded"/>
 </xsd:sequence>
</xsd:group>
```

Figure 3. A snippet of XML Schema for BPEL fragments

In the XML schema for abstract BPEL processes, the minimal cardinality of process elements has been declared as "0". In order to meet (R1) we change globally the lower limit of the cardinality of basic activities in other BPEL constructs from "0" to "1" in the XML schema for BPEL fragments. This constraint would lead to the result that each structured activity, scope, and other activity containers in a BPEL fragment, such as `<catch>`, would have to contain at least one basic activity to become a sound BPEL fragment, regardless whether this basic activity is a direct child of the `<catch>` construct or nested in structured activities and whether there exist already one basic activity in the BPEL fragment. However, a user may want to define a `<scope>` with a fault handler, in which the user does not want to define the concrete fault handling logic, but only the fault types that should be catch for this scope. In this case the `<catch>` construct does not contain any basic activity, which violets the cardinality defined in the schema. In such cases users can exploit the `<opaque>` activity to meet the cardinality constraint. It is easy to derive that our definition on BPEL fragments ensures that all BPEL fragments have unbroken control flow, because the `<fragment>` does not allow more than one BPEL activities act as direct children of the root element of a BPEL fragment and the control flow definition of the process logic in the direct children elements of the root element is complaint with BPEL standard.

5 Conclusion and Future Work

With the advent of Web services technologies, it is possible to modeling business processes by orchestrating reusable Web services in Web Services Business Process Modeling Language (WS-BPEL or BPEL for short). When modeling BPEL processes, there is a strong demand on flexible manner for modularized reuse of BPEL process models. However, the BPEL standard and the current BPEL extensions do not support such a feature. In this paper we studied the related work on process fragments in general and distinguished our definition on BPEL fragments for reuse from existing reuse approaches in a service-oriented process modeling environment. After that we presented a formal definition in XML schema for BPEL fragments, which meets the requirements we identified.

Design BPEL fragments from scratch is one possibility in process modeling. More interesting for users could be extracting of a reusable BPEL fragment from an existing BPEL processes, as most of organizations who has adopted BPEL for process modeling may have already a collection of BPEL processes. We are working on the algorithm for extracting BPEL fragments and the integration of the implementation of the algorithm into the open-source Eclipse BPEL Designer, whereas the resulted BPEL fragments are based on the formal definition presented in this paper.

References

[1] C. Alexander, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiksdahl-King, and S. Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[2] A. Alves, A. Arkin, S. Askary, C. Barreto, B. Bloch, F. Curbera, M. Ford, Y. Goland, A. Guíar, N. Kartha, C. K. Liu, R. Khalaf, D. König, M. Marin, V. Mehta, S. Thatte, D. van der Rijn, P. Yendluri, and A. Yiu. *Web Services Business Process Execution Language 2.0*. OASIS, 2007.

[3] J. Becker, M. Kugeler, and M. Rosemann. *Prozessmanagement - Ein Leitfaden zur prozessorientierten Organisationsgestaltung*. Springer, 5 edition, 2005. (In German).

[4] S. Burbeck. The tao of e-business services - the evolution of web applications into service-oriented components with web services. *IBM developerWorks*, 2000.

[5] S. H. Chang and S. D. Kim. A variability modeling method for adaptable services in service-oriented computing. In *Proceedings of the 11th International Software Product Line Conference (SPLC 2007)*, pages 261–268. IEEE Computer Society, 2007.

[6] P. Dolog. Designing adaptive web applications. In *Proceedings of the 34th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, volume 4910/2008, pages 23–33. Springer, 2008.

[7] T. Erl. *SOA Principles of Service Design*. Prentice Hall, 2007.

[8] P. Fettke and P. Loos. Using reference models for business engineering - state-of-the-art and future developments. In *Proceedings of Innovations in Information Technology*, pages 1–5, 2006.

[9] P. Fettke, P. Loos, and J. Zwicker. Business process reference models: Survey and classification. In *Proceedings of Business Process Management Workshops*, volume 3812/2006, pages 469–483. Springer, 2006.

[10] R. Khalaf. From rosettanet pips to bpel processes: A three level approach for business protocols. *Data Knowledge Engineering*, 61(1):23–38, 2007.

[11] R. Khalaf and F. Leymann. Role-based decomposition of business processes using bpel. In *Proceedings of the IEEE International Conference on Web Services (ICWS 2006)*, pages 770–780. IEEE Computer Society, 2006.

[12] R. Y. Khalaf. *Supporting Business Process Fragmentation While Maintaining Operational Semantics: A BPEL Perspective*. PhD thesis, Institut für Architektur von Anwendungssystemen der Universität Stuttgart, 2008.

[13] K.-H. Kim, J.-K. Won, and C.-M. Kim. A fragment-driven process modeling methodology. In *Proceedings of International Conference on Computational Science and Its Applications (ICCSA 2005)*, volume 3482/2005, pages 817–826. Springer, 2005.

[14] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. Ws-bpel extension for sub-processes - bpel-spe - a joint white paper by ibm and sap. 2005.

[15] W. Lam. Process reuse using a template approach: a case-study from avionics. *ACM SIGSOFT Software Engineering Notes*, 22(2):35–38, 1997.

[16] F. Leymann and D. Roller. Workflow-based applications. *IBM System Journal*, 36(1), 1997.

[17] W. C. Lim. Effects of reuse on quality, productivity, and economics. *IEEE Software*, 11(5):23–30, 1994.

[18] F. Lindert and W. Deiters. Modelling inter-organizational processes with process model fragments. In *Proceedings of Workshop Informatik '99: Enterprise-wide and Cross-enterprise Workflow Management: Concepts, Systems, Applications*, volume 24, pages 33–41. CEUR Workshop Proceedings, 1999.

[19] Z. Ma and F. Leymann. A lifecycle model for using process fragment in business process modeling. In *Proceedings of the 9th Workshop on Business Process Modeling, Development, and Support (BPDMS 2008)*, pages 1–9, 2008.

[20] P. Mohagheghi and R. Conradi. Quality, productivity and economic benefits of software reuse: a review of industrial studies. *Empirical Software Engineering*, 12(5):471–516, 2007.

[21] M. Rosemann. 22 potential pitfalls of process modeling. *Business Process Management Journal*, 11&12, 2006.

[22] N. Russell, A. H. M. T. Hofstede, W. M. van der Aalst, and N. Mulyar. Workflow control-flow patterns: A revised view. Technical report, BPM Center, 2006.

[23] H. J. Schmelzer and W. Sesselmann. *Geschätsprozessmanagement in der Praxis. Kunden zufrieden stellen, Produktivität steigern, Wert erhöhen: Kunden zufrieden stellen - Produktivität steigern - Wert erhöhen*. Hanser Fachbuch, 6 edition, 2007. (In German).

[24] W. Tan and Y. Fan. Dynamic workflow model fragmentation for distributed execution. *Journal of Computer in Industry*, 58(5):381–391, 2007.

[25] I. Trickovic. Modularization and reuse in ws-bpel. *SAP Developer Network*, 2005.

[26] W. M. P. van der Aalst. Business process management: A personal view. *Business Process Management Journal*, 10(2), 2004.

[27] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the International Conference on Business Process Management (BPM 2003)*, volume 2678, pages 1–12. Springer, 2003.

[28] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and more focused control-flow analysis for business process models through sese decomposition. In *Proceedings of the 15th International Conference on Service-Oriented Computing (ICSOC 2007)*, volume 4749/2007, pages 43–55. Springer, 2007.

[29] S. Weerawarana, F. Curbera, F. Leymann, T. Storey, and D. F. Ferguson. *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More*. Prentice Hall PTR, 2005.

[30] G. Wiederhold, P. Wegner, and S. Ceri. Toward megaprogramming. *Communication ACM*, 35(11):89–99, 1992.