Managing Process Customizability and Customization: Model, Language and Process

Alexander Lazovik[1] and Heiko Ludwig[2]

[1] University of Trento, Italy
lazovik@dit.unitn.it
[2] IBM TJ Watson Research Center, USA
hludwig@us.ibm.com

Abstract. One of the fundamental ideas of services and service oriented architecture is the possibility to develop new applications by composing existing services into business processes. However, only little effort has been devoted so far to the problem of maintenance and customization of already composed processes. In the context of global service delivery, where process is delivered to clients, it is critical to have a possibility to customize the standardized reference process for each particular customer. Having a standardized delivery process yields many benefits: interchangeable delivery teams, enabling 24/7 operations, labor cost arbitrage, specialization of delivery teams, making knowledge shared between all customers, optimization of standardized processes by re-engineering and automation. In the paper we propose an approach, where reference process models are used explicitly in the process lifecycle, where customer-specific process instantiations are obtained by a series of customization steps over reference processes. To show the feasibility of the approach, we developed a process-independent language to express different customization options for the reference business processes. We also provided an implementation that extends WebSphere BPEL4WS Editor to introduce process customizations to BPEL4WS processes.

1 Introduction and motivation

Customizing processes from reference processes or templates is common practice to reduce the time and effort to design and deploy processes on all levels. It finds application in high-level business processes design and execution, Web services compositions, and also workflows on a technical level such as integration processes and scientific workflows. Customizing reference processes usually involves adding, removing or modifying process elements such as activities, control flow and data flow connectors. Oftentimes, however, there is a business need to limit the customizability of a process, the extent to which a process can be modified.

We look at the area of management of IT service delivery as an application scenario as managing process variations is a big challenge [10]. It comprises processes for provisioning servers, creating user IDs, managing storage and the like. In an outsourcing scenario, a service provider runs similar processes for different

clients but more often than not clients will require that their specific needs be addressed. While a service provider often maintains a set of best practices as reference processes, these processes will be customized when signing up a new client. However, there are limitations to these customizations based on specific technical properties and dependencies of the system used to implement the service, staff qualifications and availability, time zone constraints and many other reasons. For example, certain activities cannot be removed as they are mandatory for subsequent activities or a regulatory requirement. In server provisioning, an operating system image must be installed prior to installing an application. Other activities must be performed in specific sequence, no other activities added in between. This is particularly the case if subsequent activities are performed by the same person in a particular location in the data center. It might not be economical to let the person wait while something else happens.

Creation and customization of processes is a topic that has undergone significant research attention, e.g., by Rosemann et al. [14] and Becker et al. [3], and is a standard practice in business consulting and system integration of large standard application packages such as ERP products. Limiting and shaping customization of process reference models, however, is a nascent topic that has only received limited attention so far.

In this paper, we propose a model of reference process that includes a definition of the limitations of how it can be customized. In addition, we propose an XML-based language to express these annotations and associate them with a business process definition to express our extended notion of a reference process.

We proceed as follows: In the next section, we discuss in more detail the types of customization operations that might be required for process models. Subsequently, we introduce a formal model of a business process, its customizing operations and the constraints that can be applied. Section 5 briefly introduces an implementation of a process editor extension to the WebSphere Integration Developer BPEL editor. Finally, we discuss related work and conclude.

2 Managing the customization life-cycle

The use of reference processes primarily improves and accelerates the definition of process models or templates to be instantiate at runtime. They are particularly useful in use cases where similar processes are defined for different organizational deployments, e.g., by a business process outsourcer or a consulting or system integration company. The use of reference processes extends the traditional process build time run time life-cycle model to a model of reference modeling customization run time. Reference process and customization are new conceptual entities that must be subjected to a government process in an organization. Staying with our scenario of service delivery management, we will use an example of a simple server provisioning process to illustrate what needs for customization and which limitations may exist.

In the above example process, the first step may be left out if the process is to be deployed in a re-provision scenario in which hardware is already in place. OS

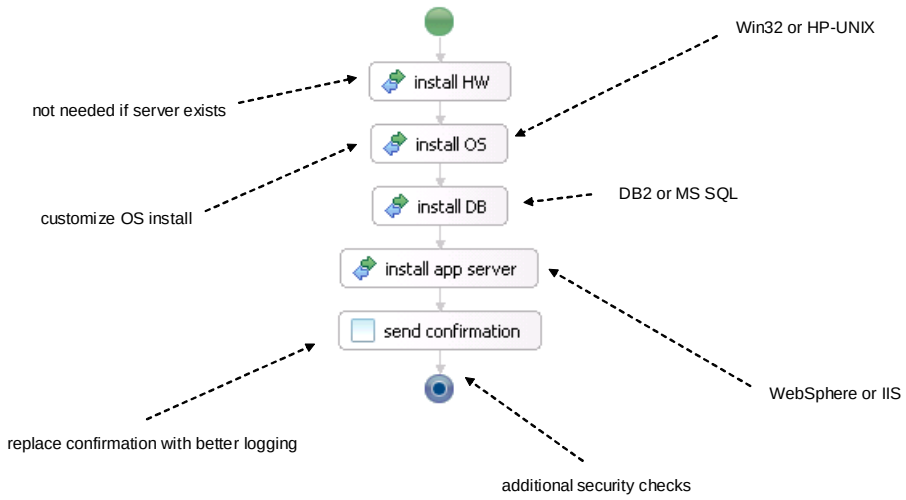

Fig. 1. Customization examples for a server build process.

installation customization may be limited to the choice between a Windows or UNIX installation sub-process; likewise for applications running on the system. The send confirmation step may be replaced with any activity, e.g., writing a log entry. While we may want to maintain the specific order of these steps, additional steps might be added to the end of the process, e.g., for additional security checks. While this process offers many opportunities to customize it, the environment of process execution, e.g., the operations of a service provider data center limit the extent to which the service provider can accept modifications to this process.

2.1 Customization Operations

While customization refers to the process of adapting a copy of a reference process in general, a specific *customization operation* is an operation on a copy of a reference process that changes (customizes) it. The specific place where a customization operation is applied is called *customization point*. The potential customization operations at a customization point are called customization options. We consider two types of customization operations: *object-based* and *process-based*. Object-based ones target a single process element without affecting the rest of the process. Process elements are primarily the activities of a process but also control flow elements such as joins and forks. Object-based customization operations are the following:

- Adding a new process element;
- Removing a process element;

– Replacing one process element with another. This also comprises a change to the definition of an object, e.g., changing an activity description or a control flow condition.

In contrast to the contained effect of object-based Process-based customization options affect the whole process. We distinguish the following cases:

– Restricting possible provider assignment for a role.

The actual provider for an activity is typically chosen at run-time from a list of possible providers. The role-restricting customization option allows restricting the possible provider assignments. For example, we may have specialized providers for operating system installations that can run either Windows or one of the Linux installation activities. If we require the installation of Linux-specific software on a server, we want to restrict the possible operating system installation provider to the one offering Linux installations.

– Adding global customization variables.

We often assume that the person who customizes or instantiates a business process has complete knowledge of the underlying process semantics. However, in practice the requester, customizer and process owner are different. Sometimes, a user understands process variables but has limited or no knowledge about the process itself. Consider the following example: When buying a computer using an on-line configuration tool, a user can customize a computer by expressing his or her preference in terms of memory size, processor, mount unit format etc. However, the choices of process variables determine the customization of the assembly process implicitly. We want to be able to express this relationship and enable global customization variables. Based on explicitly defining the set of potential customization operations and their subjects, we can define means of expressing the limits of customization that must be observed.

3 Formal model for customizable reference process

Business process is a series of a linked set of activities that is designed to accomplish some goal, where activities represent some form of atomic operations. Business process can be seen as a generic process model that represents the best practices for some particular business domain. For web service domain, activities are web services or other processes. There are many process languages used for process specification, for example, BPEL for web services. However, ideas introduced in this paper are process-independent, that is, developed customization rules and mechanisms can be applied to any of the process specification, as far as it consists of activities, control flow constructs, etc. To be precise, let us introduce a formal definition of some abstract process specification that can be used as a reference point for our customizations. In practice, we assume that these formal definitions refer to some actual specifications language. For our example introduced later are developed for processed expressed in BPEL.

Definition 1 (Process). *A process is a tuple* $B = \langle name_B, A, M, F, T, D, R, r \rangle$, *where:*

- $name_B$ *is the name of the process;*
- A *is a set of activities with one activity marked as start activity* a_0*;*
- M *is a set of possible process messages, every message is defined as a set of variables;*
- F *is a set of connectors, all of type* $\{XOR, OR, AND\}$*;*
- $D_{in} : A \rightarrow M$ *is a data transition function,* $D(a)$ *defines the input parameters for the activity* a*;*
- $D_{out} : A \rightarrow 2^M$ *is a data transition function,* $D(a)$ *defines the output parameters for the activity* a*;*
- $T : U \rightarrow 2^U$ *is a transition function, where* U *is a set of process elements and is defined as* $U = A \cup F$*. Every transition* $T(a)$ *corresponds to one of the completion states of* a*, and, from this it follows that* $|D_{out}(a)| = T(a)|$*;*
- R *is a set of roles, with* $r : A \rightarrow R$ *being a function that associates the activity with its role.*

To make our formal model simple, the definition of D_{in} indicates that each activity has only one input message, which is not necessarily always the case, since an activity can have more than one input messages. However, since every message is represented as a set of variables (and, by that, it may actually consist of several other messages), this assumption does imply any serious restrictions on the formal model. In general, a process may be seen as an atomic operation itself. In this case we assume, that for such process $B : D_{in}(B) = D_{in}(a_0)$, and $D_{out}(B) = \{D_{out}(a_i)\}$, where a_0 is a starting activity for the process B, and a_i are terminating activities. We use service registries to map activities to possible providers. Providers are usually chosen at the begin of or during process execution. We define service registries as follows:

Definition 2 (Service Registry). *A service registry* S *for process* B *is defined by the following tuple:* $S = \langle name, P, s \rangle$*, where:*

- $name$ *is a service registry reference name;*
- P *is a set of providers;*
- $s : R \rightarrow 2P$ *is a function that associates the roles* R *from process* B *with providers* P*.*

A process with customization points is called a reference process. Conceptually, a reference process defines a set of possible processes. It can be seen as a function $B(c_1, \ldots, c_n)$ over customization operations c_i. Assignments to c_i instantiate the reference process. Depending on the resolved customization options, different business processes can be potentially instantiated. This is *generating function* for the customizable reference process B_c. The outputs of the function are *instances* of the reference processes. However, this functional dependency is not function with its traditional meaning, since customization may have interrelations. That is, by resolving of some customization options other

customization options may be affected. In our setting, reference process is defined on top of the business process, and seen as original process specification plus possible customization options. Formally, we define a reference process as follows:

Definition 3 (Customizable Reference Process). *A customizable reference process B_c is a reference process B and its service registry S enriched with customizable elements and is defined by the following tuple $B_c = \langle name, name_B, S, C, t \rangle$, where:*

- *name is a name of the customizable reference process;*
- *$name_B$ is a name of the process that is made customizable;*
- *C is a set of allowed customization operations for the business process;*
- *$t(C)$ is a function that defines a set of additional properties for each of the customization, as when the customization is resolved, by what role, etc.*

In practice we work with more specific classes of customization operations, and often we define customizations implicitly, as it is described in the following sections.

4 Customizations

In this section we define possible customization operations and their formal semantics. As a result of customization we have a customized process that is identical to the original one with changes applied in potential customization points according to customization operational semantics. Now we are ready to formally define a customization of the reference process:

Definition 4 (Customization of the Reference Process). *A customization of the reference process $B_1 = \langle \ldots \rangle$ is a process $B_2 = \langle \ldots \rangle$, that is derived from B_1 by resolving customization operations.*

A customization option can be seen as an operation of modification of the process. In practice, these possible changes are defined implicitly, since the actual process customization is performed in two steps: first, by describing of the possible customization options, and, as a second, resolving the customization option to some value. In the rest of the section we discuss semantic transition rules for the customization options introduced in Section 2.1.

Adding new activity a2 between activities a1 and a3:

Requirements:

- $D_{in}(a_2) \subseteq D_{out}(a_1)$;
- $D_{in}(a_3) \subseteq D_{out}(a_2)$.

Semantic transition rules:

- $A_2 = A_1 + a_2$;
- $T_2(a_1) = T_1(a_1) \backslash a_3 + a_2$;
- $T_2(a_2) = a_3$.

Customization operation	Requirements	Semantic transition rule
Adding a_2 between a_1 and a_3	$D_{in}(a_2) \subseteq D_{out}(a_1)$ $D_{in}(a_3) \subseteq D_{out}(a_2)$	$A_2 = A_1 + a_2$ $T_2(a_1) = T_1(a_1) \backslash a_3 + a_2$ $T_2(a_2) = a_3$
Removing a between a_1 and a_2	$D_{in}(a_2) \subseteq D_{out}(a_1)$	$A_2 = A_1 \backslash a$ $T_2(a_1) = T_1(a_1) \backslash a + a_2$
Replacing $a1$ with $a2$ between $a3$ and $a4$	$D_{in}(a_2) \subseteq D_{out}(a_3)$ $D_{in}(a_4) \subseteq D_{out}(a_2)$	$A_2 = A_1\ a_1 + a_2$ $T_2(a) = T_1(a)\ a_1 + a_2$ $T_2(a_2) =$ one of $T(a_1)$
Restricting provider assignment for a role r to a set of providers I	$I \subseteq s_1(r)$	$s_2(r) = I$
Adding global customization variables	-	-

Table 1. Definition of customization operations.

Removing activity a between activities a1 and a2:

Requirements:

- $D_{in}(a_2) \subseteq D_{out}(a_1)$.

Semantic transition rules:

- $A_2 = A_1 \backslash a$;
- $T_2(a_1) = T_1(a_1) \backslash a + a_2$.

Replacing activity a1 with a2 after activity a:

Requirements:

- $D_{in}(a_2) \subseteq D_{out}(a_3)$;
- $D_{in}(a_4) \subseteq D_{out}(a_2)$.

Semantic transition rules:

- $A_2 = A_1\ a_1 + a_2$;
- $T_2(a) = T_1(a)\ a_1 + a_2$;
- $T_2(a_2) =$ one of $T(a_1)$.

Restricting a provider assignment for a role r to a set of providers:

Requirements:

- $I \subseteq s_1(r)$.

Semantic transition rules:

- $s_2(r) = I$.

Adding global customization variables:

Requirements:

- A variable may be taken from the process variables and named as customization variable.

Semantic transition rules:

- No direct impact. The actual impact of the global customization variables is performed through guidelines. This type of customizations may be seen as an abstraction that allows implicitly orchestrate the customization process without getting too much into details of actual process semantics.

A summary of atomic customization operations is provided in Table 1. Consider, for example, adding an activity a_1 between activities a_2 and a_3. When described, a customization operation usually explicitly defines the activities a_2 and a_3, that is, the place where some (not yet defined) activity may be inserted. This type of customizations may be seen as an abstraction that allows implicitly orchestrate the customization process without getting too much into details of actual process semantics. A variable is taken from the process variables and named as customization variable.

From now, when we refer to customization option, we assume that it is defined in some parametric way as described above. Within this setting, resolving the customization option is the actual customization choice (activity to be inserted in the above example).

5 Implementation

We implemented an extension to the BPEL editor of the WebSphere Integration Developer (WID) application. It is realized in the form of an Eclipse plug-in. This allows us to use BPEL Editor also as an editor for reference processes, with BPEL process enriched with customization elements. Customizations are stored in a separate file in the same folder as the customized BPEL process.

The reference implementation architecture is shown in Figure 2. WID uses the same scheme of extensions based on plug-ins as the original Eclipse platform (www.eclipse.org). To allow customizations of BPEL processes, we extended the BPEL Editor with a set of plug-ins:

- EMF customization model extends the BPEL EMF model, and is recognized by the BPEL editor as BPEL extension elements;
- To save/load BPEL processes, if the process contains any customization extensions, they are saved/loaded using in/from a separate file, making the on-the-fly validation against customization XML schema. If the validation fails, the problems tab points to a set of validation errors;

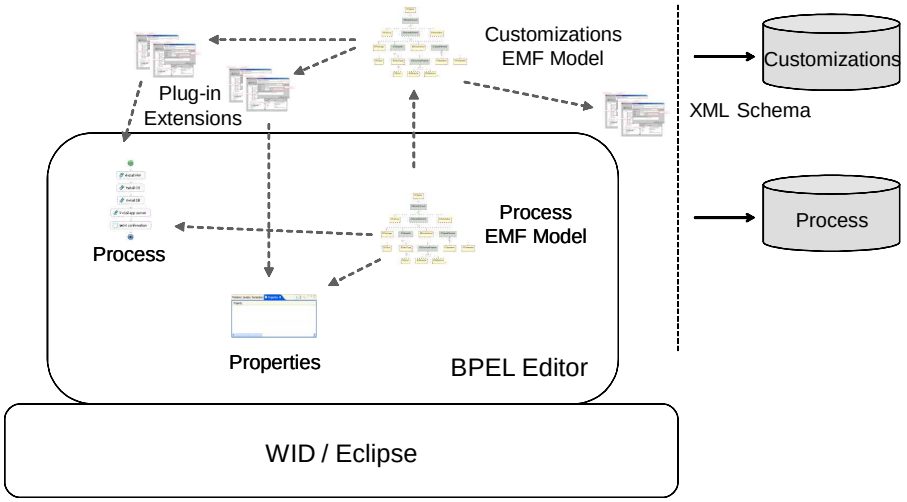

Fig. 2. Implementation of a customizability annotator as extension to a BPEL editor.

– To fully support visual editing of customizations, for each element we added a separate visual extension element to BPEL Editor, as shown in Figure 3. Context-aware property tabs are activated, when the customized element is chosen. To define relations between customizations in the form of guidelines and requirements the separate view is used as shown in Figure 3.

The reference process XML representation generated by the tool is used as input to a customization tool.

6 Related work

In[14] the authors propose configurable event-driven process chains (C-EPC) as an extended reference modeling language which allows capturing the core configuration patterns. The general requirements for configurable reference modeling languages are identified and formalization of C-EPC model is provided as an example. Comprehensive discussion and overview of variability in process models is provided in [6, 1]. Modeling variability by UML Use Case diagrams is introduced in [20]. However, the authors do not deal with the issue of customizations specification. The relations and inheritance between processes and the possibilities of transforming of one process to another are discussed in [2]. SAP [5, 15] (as well as other Enterprise Systems reference models) uses reference models as application process models that document the functionality of off-the-shelf-solutions. However, these models are specific to domains for which their processes were initially developed with only limited customization options whereas address the issue of a general customizability approach. Rational Method Composer (RMC) [13] delivers a unified software process with the possibility to configure the software

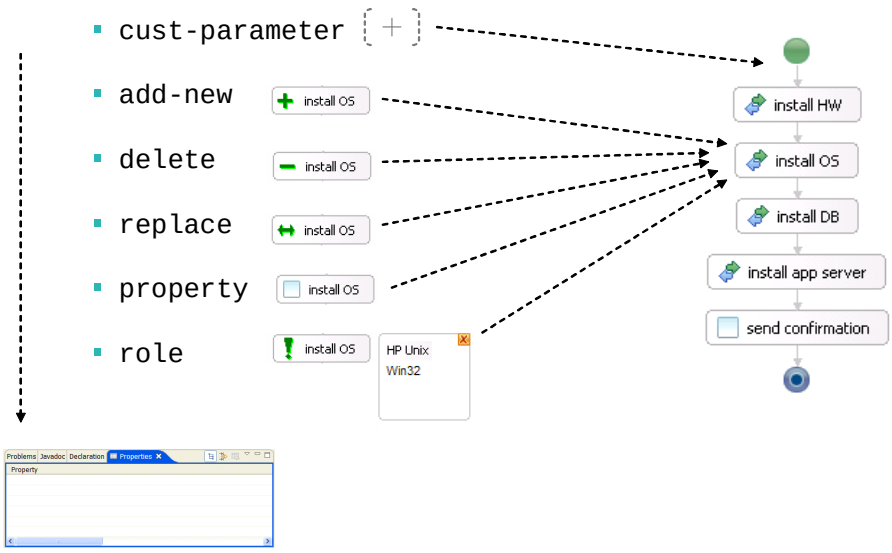

Fig. 3. Customization elements attached to a process activity.

process in a customer-specific way on a base of initial configuration. While the customization possibilities of RMC are extensive, there is no or little control over how the customization process happens. In contrast, this work not only provides the language for customization options, but also the mechanism to control and validate the customization process by means of requirements and guidelines. The concept of guidelines and requirements is very similar to those defined in [12]. Requirements allow designers to limit the configuration space, while guidelines allow them to shape the configuration process. However, the actual validation and application of requirements and guidelines within the process lifecycle has received only limited attention in [14, 12]. Capturing variability of business process models in a language-independent way is not new [19]. Here the authors represent business processes with labeled transition systems and use simple atomic operations as hiding and blocking to restrict the process behavior. Study of BPEL-specific customization may be found in [7], where run-time configuration capabilities are offered, and in [8], where BPEL processes are derived from collaboration templates.

There are a number of alternative approaches to build customer-specific processes. For example, automatic service composition techniques allow delivering a user-specific process based on a set of pre-defined objectives and preferences. The techniques can be based on template planning [17], data dependency [18] or finite state machines [4]. Planner such as Golog [11] and the HTN planner SHOP2 are applied for web service composition [16]. In [9] a service request language (XSRL) is introduced to compose the executable process based on process reference models and the goal describing desired service attributes and function-

alities, including temporal and non-temporal constraints. The drawback of the approach is the limited control of the customer on the actual process synthesis and execution. A general drawback of automatic composition is that in general it is difficult to maintain the semantic descriptions of process activities up-to-date, making altering the service implementations difficult. Hence, while automated generation approaches implicitly contain a constraint on the set of process that can be created it is usually not in a form in which it can be usefully managed by a person who wants to restrict reference process use.

7 Conclusion

This paper outlines the issue of constraining the customizability of reference processes and introduces a model and representation for defining customization options on reference processes. The proposed approach allows a reference process editor to explicit control the customizability of the reference process. We defined a formal model for customizations with detailed operation semantics rules for each customization type. The formal model is built on top of existing process definitions. The XML-based language for customization definitions can be used in the context of existing process definition language. It is designed with an idea of being independent from particular process notation. To demonstrate the feasibility of the approach, we developed a reference prototype as an extension to WebSphere Integration Developer BPEL Editor.

This is an important contribution to the management of process variability that is in particular relevant for service providers that deal with numerous instances of a similar process entailing high life-cycle costs.

In the next step we plan to further investigate the implication of customization operations. For example, application of object-based customization operations (like removing or modifying an activity) does not always have only local implications. What if there are connectors in-between the activities where a new activity is inserted? It would be also interesting to understand the implications on the behavioral correctness (e.g. removing an activity may lead to a deadlock). As a separate line of research, we also plan to address different ways of executing customization options, which depend to a large extent on the richness of the choices and the level of skills of the customizer. We plan to extend the provided implementation to validate and simulate customization options. Extensions for other process notations, e.g., BPMN, are also planned.

References

1. F. Bachmann and L. Bass. *Managing Variability in Software Architecture.* ACM Press, 2001.
2. T. Basten and W. M. P. van der Aalst. Inheritance of behavior. *J. Log. Algebr. Program.*, 47(2):47–145, 2001.
3. J. Becker, P. Delfmann, and R. Knackstedt. Adaptative reference modeling: Integrating configurative and generic adaptation techniques for information models. In *Reference Modeling Conference*, 2006.

4. D. Berardi, D. Calvanese, G. De Giacomo, M. Lenzerini, and M. Mecella. Automatic composition of e-services that export their behavior. In *ICSOC-03*, 2003.
5. T. Curran and G. Keller. *SAP R/3 Business Blueprint: Understanding the Business Process Reference Model*. Upper Saddle River, 1997.
6. G. Halmans and K. Pohl. Communicating the variability of a software-product family to customers. *Software and System Modeling*, 2(1):15–36, 2003.
7. D. Karastoyanova, F. Leymann, J. Nitzsche, B. Wetzstein, and D. Wutke. Parameterized bpel processes: Concepts and implementation. *Business Process Management*, pages 471–476, 2006.
8. R. Khalaf. From rosettanet pips to bpel processes: A three level approach for business protocols. *Business Process Management*, pages 364–373, 2005.
9. A. Lazovik, M. Aiello, and M. Papazoglou. Planning and monitoring the execution of web service requests. *Journal on Digital Libraries*, 2005.
10. H. Ludwig, J. Hogan, R. Jaluka, D. Loewenstern, S. Kumaran, A. Gilbert, A. Roy, and M. Surendra T. Nellutla. Catalog-based service request management in the bluecat environment. *IBM Systems Journal*, 46(3), 2007.
11. S. McIlraith and T. C. Son. Adapting Golog for composition of semantic webservices. In *Conf. on principles of Knowledge Representation (KR)*, 2002.
12. J. Mendling, J. Recker, M. Rosemann, and W. M. P. van der Aalst. Towards the interchange of configurable EPCs. In *EMISA*, pages 8–21, 2005.
13. RMC. Rational method composer. `http://www-128.ibm.com/developerworks/rational/products/rup`.
14. M. Rosemann and W. van der Aalst. A configurable reference modelling language. *Inf. Syst.*, 31(1):1–32, 2007.
15. SAP. Online documentation mySAP ERP. `http://help.sap.com`, 2005.
16. E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4), 2004.
17. S. Thakkar, J. Ambite, C. Knoblock, and C. Shahabi. Dynamically composing web services from on-line sources. In *AAAI Workshop Intelligent Service Integr.*, 2002.
18. S. Thakkar, J. Ambite, C. Knoblock, and C. Shahabi. Dynamically composing web services from on-line sources. In *AAAI Workshop Intelligent Service Integr.*, 2002.
19. W. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M. Jansen-Vullers. Configurable process models as a basis for reference modeling. In *BPM Workshops*, pages 512–518, 2005.
20. T. von der Massen and H. Lichter. Modeling variability by uml use case diagrams. In *REPL 02*, 2002. Technical Report: ALR-2002-033, AVAYA labs.