Jan Mendling (Ed.)

BPM Demo Session 2006

at the
4th International Conference on
Business Process Management (BPM 2006)
in Vienna (Austria),
5-7 Sept 2006

Proceedings

Preface

This proceedings volume contains the papers presented at BPM Demo Session 2006 which was part of the 4th International Conference on Business Process Management (BPM 2006) in Vienna (Austria), on 5-7 Sept 2006.

BPM 2006 brought together research and industry as well as crossed the boundaries of existing research communities in all areas of business process management. We invited submissions for demos to be included in the BPM 2006 Demonstration Program. This program was intended to showcase innovative business process related implementations, technologies, and analysis tools.

Each demo paper was reviewed by at least three members of the program committee. Submissions were evaluated on the basis of their innovation, scientific contribution, presentation, industrial and application relevance. We encouraged the authors to make the tools available for download and to mention the web address of the tool in the paper. Each paper was presented at the BPM conference and a A/1 poster summarizing the tool and the demo paper was exhibited in the BPM conference lounge.

This proceedings volume includes six carefully selected papers presented at the BPM demo session. The topics of the demos ranges from querying process repositories, verification, configurable execution to modeling of service interaction and middleware support for BPEL process execution.

We thank the authors, the members of the program committee, and the local organization team of the BPM conference for contributing to the realization of this demo session.

Vienna, August 2006 Jan Mendling

Program Committee

Boudewijn van Dongen, TU Eindhoven, The Netherlands

Michael Hafner, University of Innsbruck, Austria

Patrick Hung, University of Ontario, Canada

Matjaz Juric, University of Maribor, Slovenia

Rania Khalaf, IBM Research, United States

Agnes Koschmider, University of Karlsruhe (TH), Germany

Kristian Bisgaard Lassen, University of Aarhus, Denmark

Marek Lehmann, University of Vienna, Austria

Jan Mendling, WU Vienna, Austria (Chair)

Frank Puhlmann, HPI Potsdam, Germany

Jan Recker, QUT Brisbane, Australia

Stefanie Rinderle, University of Ulm, Germany

Florian Rosenberg, TU Vienna, Austria

Nick Russell, QUT Brisbane, Australia

Oliver Thomas, University of Saarland, Germany

Martin Vasko, TU Vienna, Austria

Eric Verbeek, TU Eindhoven, The Netherlands

Barbara Weber, University of Innsbruck, Austria

Petia Wohed, Stockholm University, Sweden

Andreas Wombacher, University of Twente, The Netherlands

Table of Contents

ProcessGene Query – a Tool for Querying the Content Layer of Business Process Models

Avi Wasser[1], Maya Lincoln[1] Reuven Karni[1]

[1] ProcessGene Ltd.
15303 Ventura Boulevard, Sherman Oaks, California, 91403, USA
{avi.wasser, maya.lincoln, Reuven.karni}@processgene.com

Abstract. One of the main challenges currently facing the world of enterprise information technology in general and ERP/SCM/CRM systems in particular, is visibility into the business of organizations. While the phenomena of devising supporting tools for process execution frameworks is widespread in academia and practice, there have been few attempts to develop methodologies and software tools that support structured analysis of the business process content layer. The incorporation of content into a business process model produces complexity in the sense that it adds semantics and relationships of actual business data. To confront this complexity, this research suggests a framework and a supporting software tool "ProcessGene Query" for conducting search-queries on business process models.

1 Introduction

One of the main challenges currently facing the world of enterprise information technology, and ERP systems in particular, is visibility into the business of organizations, [10]. The prevalent approach utilizes conceptual business process modeling as the foundation for creating and managing this visibility, aiming to connect the business activity and its supporting information technology (IT) systems [6].

The current main thrust of business process modeling research has focused on the study of structural frameworks and execution patterns [9], putting little emphasis on the content layer that is supposed to populate these frameworks. "Real life" business process models, which contain practical content objects, have been disregarded [9], except in illustrative examples.

Structural process frameworks define formal architectures and standards for representing business activities and processes. The spectrum (Fig. 1) ranges from simple descriptive frameworks such as activity diagrams, suitable mostly for business users, through more formal frameworks such as OPM [11] and Petri-nets, suitable mostly for software implementers and IT system analysts, to code-compatible structures such as BPEL and XLANG [12], suitable for software developers.

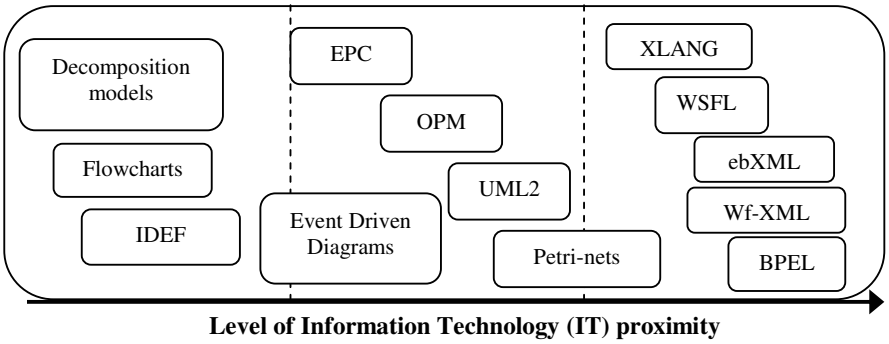


Fig. 1: The structural frameworks spectrum

The practical deployment of these frameworks, involves an attempt to enumerate *actual* business processes carried out within enterprises. Modeling in this context focuses on the *content* layer of business process models. We define the content layer as the itemization of the suite of actual business processes constituting the framework of business-related activity within a particular industrial sector, or, alternatively, within a particular enterprise. Only a few scientific publications address the topic of business process content [7]-[9]. On the other hand the initiative has been taken and business process content was developed and applied, by enterprise software vendors, IT integrators, and BPM commercial firms.

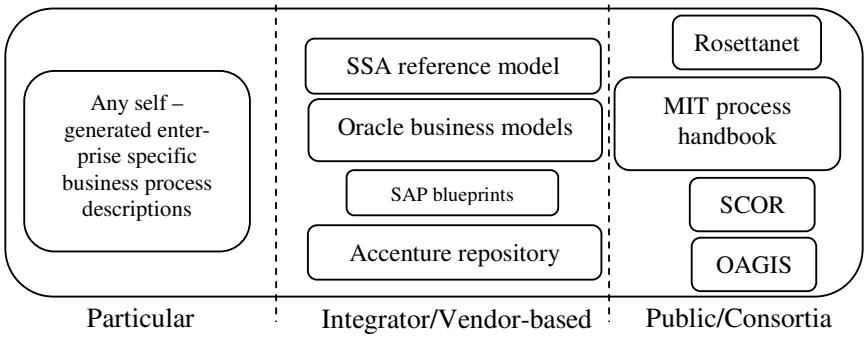


Fig. 2: Business process content examples

Fig. 2 presents some business process content compendia, divided into three main types: (a) particular, enterprise specific content; (b) vendor/integrator content such as the OBM (Oracle Business Models) library [4] and SAP solution maps [5] and (c) collaborative/consortia content frameworks such as the MIT process handbook [1], OAGIS [13] and Rosettanet [14]. Thus, while the phenomena of formulating structural execution frameworks is widespread in academia (e.g. [15]), there seem to be few attempts to develop theories, empirical studies and supporting tools [9] (such

as generation, customization, validation and search mechanisms) for "complete" business process models which incorporate an actual content layer (Fig. 3).

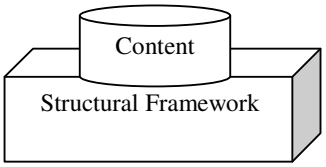


Fig. 3: A process model as a combination of structure and content layers

When this research addresses *business process models* it refers to "complete" models that also include a *content* layer, so that the combination of structure and content can display the actual suite of business processes constituting the framework of activity within the enterprise and enable subsequent implementation through IT. For example: a flowchart describing bottleneck leveling in production, or a Petri-net describing the process of managing a service request in CRM. Such business process models are considered complex since they include a large number of interconnected data objects (processes, roles, events, related data, etc.). This complexity increases when the models are to be expressed and actualized by a corresponding IT system (e.g. ERP/SCM/CRM), which requires verification and validation of the business process models from a functional and managerial point of view prior to actual implementation and subsequent execution. To confront this complexity, and in order to enable effective handling of the business process models content, this research suggests a framework and a supporting software tool for conducting search-queries on business process models.

The paper features the following sections: a demonstration of a standardized format for describing the content of a business processes based on current offerings of ERP vendors – (section 2); the "ProcessGene Query" methodology and tool for searching the content of process models (section 3); an example for running content search queries (section 4); conclusions and suggestions for further work (section 5).

2 Describing the Content of Business Process Models

Due to their dominance in industry, we will focus on content layers from vendor/integrator commercial business process models. These include, for example, SAP's industry and cross-industry Business Solution Maps [5], Lawson-Intentia's ERM (Enterprise Reference Models) [2] and Oracle's OBM (Oracle Business Models) library [4]. In the Oracle business process flows, for example (Table 1), the top level "high level flow" for an industrial sector presents names and descriptions of the high level functionalities for that industry (about 7), and their corresponding business flows (about 7). Business flows are then broken into activities and tasks, holding similar amount of items at each level.

Table 1: Oracle E-Business suite process content hierarchy

(1) High Level Flow = "Procure to Pay" (top hierarchal level)
(2) Business Flow = "Analyze to Agreement" (second level)
(3) Activity/Procedure = "Negotiate and Select Suppliers" (third level)
(4) Task = "Enter supplier information" (fourth level)- with a link to corresponding IT components such as setups and customizations of datasets

From these categorizations vendors and integrators develop a suite of processes, reflecting what an enterprise does, or needs to do, in order to achieve its objectives [3]. Furthermore- the content includes pointers to additional content items that are in use during an implementation process such as user requirements, test scripts, setup parameters, flow diagrams, workflows and related documents. If we assume an amount of seven items at each hierarchal content level we would reach almost 20,000 interconnected data items, not counting the additional process-related content items. It is also important to realize that each item holds a certain amount of metadata, which users may need to retrieve and review. Research into a vendor/integrator defined commercial business process models has introduced several concepts: (a) the necessity for a compendium of realistic business processes in order to be able to generate practical enterprise models; (b) the inclusion of cross-references between business processes, additional content items and IT components offered by software vendors; and (c) the complexity in a concurrent management of a relatively large dataset. To confront complexity, this research suggests a query methodology and supporting tool for assisting in the retrieval and management of the business process content layer.

3 Methodological Framework

In order to formulate and demonstrate the proposed query framework, we present two data models that organize the business process data and form the foundation for running search-queries on the content layer. Then we elaborate on the query method.

3.1 Process Data Structure Models

Process Descriptor Decomposition Model. This model introduces the basic ideas and notations for formally representing business process model content objects by a hierarchal graph of descriptors, as shown in Fig. 4. The process model contains n levels of process hierarchy (L1, L2, … , Ln). At each level, each process is represented by a single process descriptor, and each process descriptor consists of one action, one object that the action acts upon, and possibly one or more action qualifiers, object qualifiers and means.

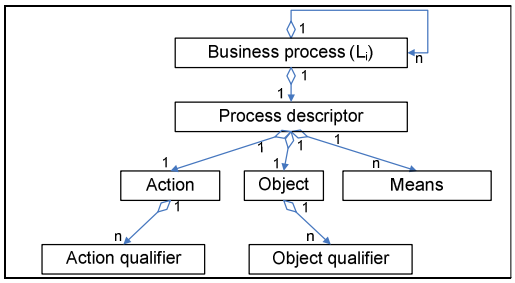


Fig. 4: The process descriptor decomposition model

For example, a process descriptor can be defined as: "Issue confirmed purchase order to local supplier by e-mail", comprising an object, an action and their qualifiers.

Business Action and Object Taxonomy Model. This model organizes a set of process descriptors, attempting to determine the relationships between business actions and objects both longitudinally (hierarchically) and latitudinally (in terms of execution order) as described in Fig 5. In this model an action is related to an object by an operability connector, e.g. the action "receive" is related to the object "invoice". Longitudinally- the action "issue" is considered a subclass (a more specific form) of "produce", and the object "purchase order" is a subclass of "purchasing document" (note that the operability connectivity applies also to relations between different hierarchy levels). Latitudinally, each object holds a list of ordered actions applied on that object (e.g. the object "product" is related to the actions "plan" followed by "produce"); (b) a list of ordered objects that express the object lifecycle (e.g. the following lifecycle sequence: "raw material"→, (…)→, "product"→, (…)→, "returns"→, (…)).

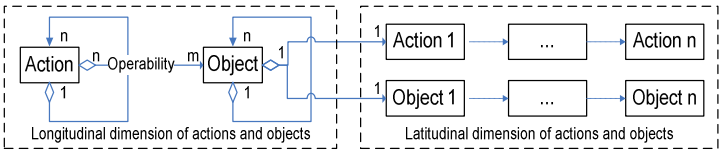


Fig 5: Business action and object taxonomy model

These longitudinal and latitudinal viewpoints contribute another dimension for analyzing and learning the business process model content layer in terms of identifying action and object hierarchies and execution sequences.

3.2 The ProcessGene Content Query Method

The method aims to provide a simple yet powerful query interface in which users are able to express and perform a large set of queries using intuitive definitions.

The ProcessGene Query mechanism includes four main components. At the front-end: a Scoping-Assistant (SA), for defining the content query range; and a Query Specification Interface (QSI), for expressing the user's data extraction requirements. At the back-end: a Query Interpreter (QI) for interpreting the user specification into a set of normalized queries; and a Query Results Packager (QRP) for packaging the retrieved results to include only data that is of interest to the user. The SA uses business processes as means for query focusing, since at any hierarchy level, these objects are related with all other data components. After defining the query's underlying data scope, the QSI enables users to specify data requirements. This module is based on two specification layers, offering at the first layer a simple interface, which enfolds more advanced options for users that wish to drill-down and expand the query capability. The first query specification layer presents all business process model component types as a flat checklist, enabling the user to select query components. Each component can then be expanded, presenting additional data fields and enabling the user to specify different criteria for each field. Conditions are expressed using regular expressions (strings, keywords, wildcards), or by selecting one or more values from a list of values, depending on the data field type. In addition, the QSI also assists in defining the query result structure and content. Instead of generating pre-defined result segment structures, the user can define which data components are to be included in the result set. After the specification phase, the QI analyzes the user request and composes a set of all compatible normalized queries. The QRP then modifies the retrieved results to include only data fields that were required by the user. These manipulated results are eventually presented to the user according to the business process model hierarchy.

4 Example: Process Content Query

To illustrate the proposed framework for supporting a search query on business process content we present an example, in which a user is interested to find out "how order-based decisions are handled by sales representatives". Using the SA, the user selects a level 1 process, "Order to Cash", based on the information that orders can be handled by sales representatives at pertaining lower-hierarchy business processes (e.g. Order Management, Shipping Management, …) (Fig. 6).

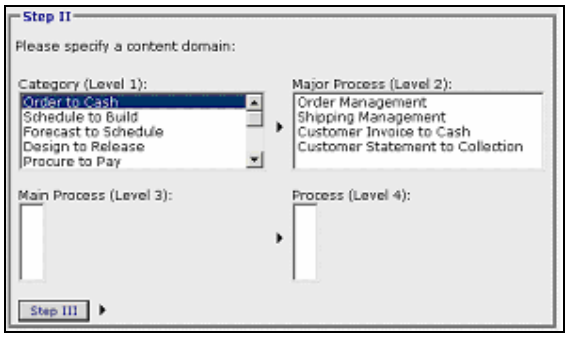


Fig. 6: The query scoping assistant (SA)

At the next step, the user uses the QSI to select process levels and define content requirements for the relevant process fields (Fig 7).

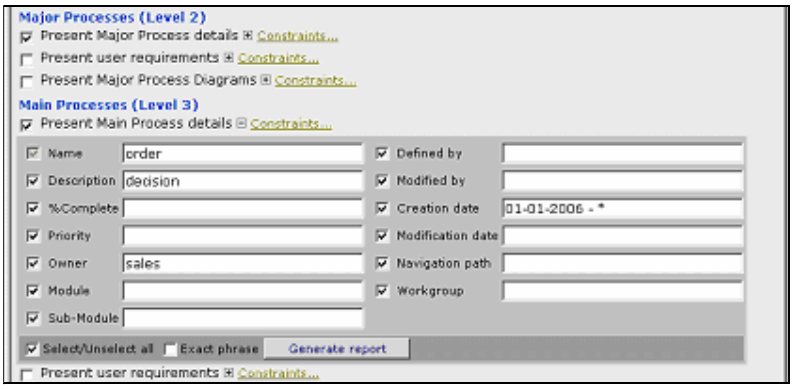


Fig 7. The query specification interface (QSI)

Following our example, the user will limit the "Name" field to include the string "order", the "Description" field to include the string "decision", and the "Owner" field to include the string "sales". He leaves the "exact phrase" option unchecked in order to retrieve more results. If, in addition, the user is interested only in "new" processes defined in the organization after a new sales strategy was implemented during 2006, he will add to the "Creation date" field the expression: "01-01-2006 - *". On top of these data fields the user can also check other required data fields. At the next step, the QI interprets QSI definitions into a set of normalized SQL queries, and the QRP joins all resulted data fields into query results ordered by hierarchal location witin the business process model. The example demonstrates how a user without any in-depth understanding of the data structure can extract relevant results for a relatively complex query – all by using the SA and the QSI.

5 Summary

The ProcessGene Query system provides a method for searching business processes, allowing users to phrase queries without extensive knowledge of the underlying database structure. Although the system provides a good starting point for developing the field of business process search queries, many innovations are needed to exploit open issues such as optimization of result sets, adding business logic for determining semantically related answers, query relaxation and the ranking of results. These issues were discussed extensively in the literature, but have not been addressed yet within the context of business process management.

It is hoped that by expanding the search and query capabilities on business processes content, researchers and IT practitioners will be able to generate complete and consistent business process models as part of their services to ERP/CRM/SCM community.

6 References

[1] Malone, T. W. *The Future of Work: How the New Order of Business Will Shape Your Organization, Your Management Style, and Your Life*. Boston, MA: Harvard Business School Press, 2004.

[2] Intentia. Reference Models, *http://www.intentia.com/WCW.nsf/pub/tools_index*, 2004.

[3] M. Lincoln, Karni R. A Generic Business Function Framework for Industrial Enterprises. *CD Proceedings of 17th ICPR Conference*, Blacksburg, VA, USA, October 2003.

[4] Oracle. Business Models (OBM), *http://www.oracle.com/consulting/offerings/implementation/methods_tools/*, 2006.

[5] SAP Solution Composer, *http://www.sap.com/solutions/businessmaps/composer/* 2006.

[6] C.P. Holland, B. Light, *A critical success factors model for ERP implementations*, IEEE Software 16, 1999, 30–35.

[7] Fettke, P.; Loos, P.; Zwicker, J.: Business Process Reference Models - Survey and Classification. In: Kindler, E.; Nottgens, M.: Business Process Reference Models – BPRM workshop proceedings: 1-15, 2005, BPM2005 workshop.

[8] A. Bernstein. Process Recombination: An Ontology Based Approach for Business Process Re-Design. *SAP Design Guild*, Vol. 7, October 2003.

[9] Avi Wasser, Maya Lincoln, Reuven Karni: Accelerated Enterprise Process Modeling Through a Formalized Functional Typology. BPM 2005: LNCS 3649 446-451.

[10] C. Rolland, N. Prakash, Bridging the gap between organizational needs and ERP functionality, Requirements Eng. 41, 2000, 180–193.

[11] Dov Dori, Iris Reinhartz-Berger: An OPM-Based Metamodel of System Development Process. ER 2003: 105-117

[12] G. Yang. Towards a Library for Process Programming. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *BPM 2003*, volume 2678 of *Lecture Notes in Computer Science*, pages 120-135. Springer-Verlag, Berlin, 2003.

[13] OAGIS. Best Practices and XML Content for Everywhere-to-Everywhere Integration, http://www.openapplications.org/, 2004.

[14] Rosettanet. Lingua Franca for Business, *http://www.rosettanet.org/*, 2004.

[15] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede: YAWL: yet another workflow language. Inf. Syst. 30(4): 245-275, 2005.

A Tool Chain for Lazy Soundness

Frank Puhlmann

Business Process Technology Group
Hasso-Plattner-Institute for IT Systems Engineering
at the University of Potsdam
D-14482 Potsdam, Germany
puhlmann@hpi.uni-potsdam.de

Abstract. This paper introduces a prototypic tool chain to investigate the feasibility of deciding lazy soundness for Business Process Diagrams (BPD). We utilize a graphical editor to create BPDs, export them to XML, convert them to formal π-calculus expressions, and finally use existing π-calculus reasoners to decide lazy soundness.

1 Introduction

Business Process Management (BPM) aims at designing, enacting, managing, analyzing, and adapting business processes [1]. This paper focuses on a special kind of analysis, called *verification*. Verification proves correctness of business processes regarding structural constrains like *deadlocks* or *livelocks* that require a formal semantics of the routing constructs contained in business processes. The correctness criterion investigated is called *Lazy Soundness* [2].

In order to show the feasibility of deciding lazy soundness for business processes we developed a prototypic tool chain. A business process diagrams (BPD) is created graphically using the Business Process Modeling Notation (BPMN) [3]. The BPDs are then exported to an intermediate XML file that provides a generic abstraction from concrete modeling notations. The business processes contained in the XML file can already be checked for structural constraints like connectedness of the nodes. To prove lazy soundness, the XML file is converted to π-calculus expressions. The π-calculus is a generic process algebra that is used to give formal semantics to common patterns of behavior found in business processes [4]. The formalizations are then applied to prove lazy soundness using existing tools.

The remainder of this paper is structured as follows. It starts by introducing the context, i.e. give a definition of lazy soundness and introduce the π-calculus representation of business processes. Thereafter we discuss the architecture of the tool chain and illustrate it using an example. Finally the paper is concluded by discussing related work and further developments.

2 Context

The tool chain is based on the concepts and algorithms for lazy soundness introduced in [2]. The theoretical foundations are given by the π-calculus [5]. While

lazy soundness is a new correctness criterion for the BPM domain, the π-calculus is already in discussion as a formal foundation for BPM [6,7]. Lazy soundness is based on *structural soundness*, informally given by:

> A business process is *structural sound* if and only if there is (a) exactly one initial activity, (b) exactly one final activity, and (c) all activities are on a path from the initial to the final activity.

Furthermore, lazy soundness requires *semantic reachability*, meaning that an activity B is reachable from another activity A (i.e. their exists a path between them) according to the semantics of all other activities such as splits and joins. Lazy soundness is then given by:

> A business process is *lazy sound* if and only if (a) the final activity is semantically reachable from every other activity semantically reachable from the initial activity until the final activity has been executed, and (b) the final activity is executed exactly once.

The definition states that a lazy sound business process is deadlock and livelock free as long as the final activity has not been executed. So called *lazy* activities might still be or become executed. Those are usually required for clean–up or subsequent activities. Examples are activities before a *discriminator* or *n–out–of–m–join* that has already been executed (i.e. receive remaining messages in interacting business processes) or activities triggered by *multiple–instances–without synchronization* patterns [8]. In terms of Petri nets, lazy soundness supports processes where tokens can remain in the net. Again, a detailed discussion can be found in [2].

A formal semantics for business processes is given by the π-calculus. In [4,9] we have shown how different routing patterns are mapped to π-calculus expressions. Basically, each activity of a business process is mapped to a corresponding π-calculus process. The processes then trigger themselves using a pre– and post-condition approach. Reasoning about lazy soundness is done using *weak open bisimulation*. Informally, two π-calculus processes are weak open bisimulation equivalent if they have the same observable behavior regarding certain observability predicates. Weak open bisimulation can be evaluated using existing tools.

3 Architecture

Figure 1 depicts the tool dependencies and document flows in the tool chain. Tools or scripts are shown as rectangles, whereas documents are denoted as notes. The components developed by our group are shown inside the dotted area.

First of all, we utilize a *graphical editor* for designing business process diagrams. The editor is equipped with a set of *BPMN stencils* annotated with additional information. Based on this information, an *XML exporter* script is able to generate an XML description of the business process diagram by interacting with the editor. The *XML* representation of the business process can already

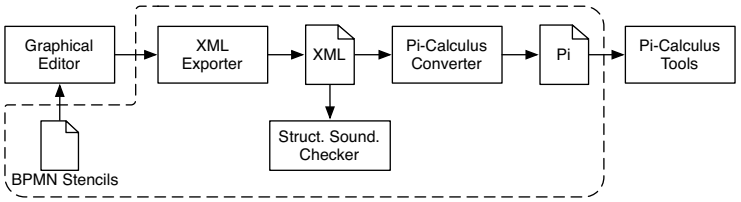


Fig. 1. Architecture of the tool chain.

be proved to be structural sound by a *structural soundness checker* script. Furthermore, it can be used as input for a *pi-calculus converter* script that maps the XML file to a proprietary ASCII notation representing π-calculus processes. The implemented algorithm is described in [2]. The file containing the π-calculus processes can then directly be used as an input for existing *π-calculus tools* for reasoning.

Technically, the feasibility study has been developed on Mac OS X. *OmniGraffle Professional* is utilized as a graphical editor[1], but other editors are also possible. OmniGraffle is fully programmable using *AppleScript* that was used for implementing the XML exporter. Both, OmniGraffle and AppleScript, provide an easy and convenient way of designing and exporting business process diagrams. The π-calculus converter and the structural soundness checker have been implemented as Ruby scripts, so they are OS–independent. The π-calculus tools compatible with our scripts are MWB and ABC, the two major reasoners for π-calculus [10,11]. Both are also available on various platforms.

4 Example

After introducing the theoretical foundations and architecture of the tool chain, we are now ready to give an illustrating example. Figure 2 shows a business process starting with a parallel split, leading to the parallel execution of activities A, B, and C. These activities can represent sub-processes for contacting three different experts for writing an expertise. A *2–out–of–3–join* continues the execution at activity D after two of them are ready. However, some cleanup work is left for the remaining activity, e.g. receiving the last expertise and paying the expert. Activity D spawns of three multiple instances of itself, sending the accepted expertises to three different involved persons. While the expertises are still in delivery, the business process is already finished.

The interesting point regarding lazy soundness are the lazy activities that are left behind. This might be one of A, B, or C, as well as the three instances of D. To prove the business process to be lazy sound, we need to export it from our graphical editor using the XML exporter tool. The tool creates an XML file representing a so called *process graph* of the BPD. A process graph is a

[1] `http://www.omnigroup.com/applications/omnigraffle`

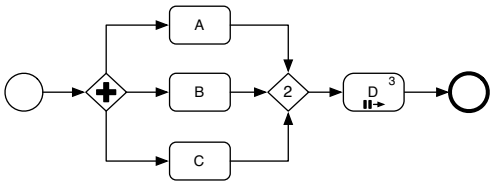


Fig. 2. Example business process diagram.

mathematical structure to describe the static aspects of a business process (see also [2]). The XML representation of the process graph looks as follows:

Example 1 (XML representation of the example).

```
<model>
   <process id="1" type="BPMN">
      <node id="1025" type="MI without Sync" name="D" count="3"/>
      <node id="538" type="End Event"/>
      <node id="748" type="N-out-of-M-Join" continue="2"/>
      <node id="790" type="Task" name="C"/>
      <node id="789" type="Task" name="B"/>
      <node id="717" type="AND Gateway"/>
      <node id="677" type="Task" name="A"/>
      <node id="534" type="Start Event"/>
      <flow id="799" type="Sequence Flow" from="1025" to="538"/>
      <flow id="798" type="Sequence Flow" from="748" to="1025"/>
      <flow id="797" type="Sequence Flow" from="790" to="748"/>
      <flow id="796" type="Sequence Flow" from="789" to="748"/>
      <flow id="795" type="Sequence Flow" from="677" to="748"/>
      <flow id="794" type="Sequence Flow" from="717" to="790"/>
      <flow id="792" type="Sequence Flow" from="717" to="789"/>
      <flow id="791" type="Sequence Flow" from="717" to="677"/>
      <flow id="671" type="Sequence Flow" from="534" to="717"/>
   </process>
</model>
```

Using the structural soundness checker script, the process graph contained in the XML file can be proved to be structural sound (omitted here). The dynamic aspects of the business process are generated out of the type descriptions for each node contained in the XML file by the π-calculus converter. The formal description is furthermore enhanced with lazy soundness annotations as well as a special process called S_{LAZY} used for reasoning later on:

Example 2 (π-calculus representation of the example).

```
agent N1025(e798,e799)=e798.(t.0 | t.0 | t.0 | 'e799.0 | N1025(e798,e799))
agent N717(e671,e794,e792,e791)=e671.t.('e794.0 | 'e792.0 | 'e791.0 |
      N717(e671,e794,e792,e791))
agent N677(e791,e795)=e791.t.('e795.0 | N677(e791,e795))
agent N534(e671,i)=i.t.'e671.0
agent N538(e799,o)=e799.t.'o.N538(e799,o)
agent N748(e797,e796,e795,e798)=(^h,run)(N748_1(e797,e796,e795,e798,h,run) |
      N748_2(e797,e796,e795,e798,h,run))
agent N748_1(e797,e796,e795,e798,h,run)=e797.'h.0 | e796.'h.0 | e795.'h.0
agent N748_2(e797,e796,e795,e798,h,run)=h.h.'run.h.N748(e797,e796,e795,e798) |
      run.t.'e798.0
agent N790(e794,e797)=e794.t.('e797.0 | N790(e794,e797))
agent N789(e792,e796)=e792.t.('e796.0 | N789(e792,e796))
agent N(i,o)=(^e799,e798,e797,e796,e795,e794,e792,e791,e671)(N1025(e798,e799) |
```

```
      N717(e671,e794,e792,e791) | N677(e791,e795) | N534(e671,i) | N538(e799,o) |
      N748(e797,e796,e795,e798) | N790(e794,e797) | N789(e792,e796))
agent S_LAZY(i,o)=i.t.'o.0
```

The input style generated corresponds to MWB as well as ABC. Each node of the XML file has been mapped to a π-calculus process (denoted as *agent* in the syntax). For instance, the initial node is given by $N534$, or the 2–out–of–3–join by $N748$. Helper agents are denoted with an index, like 748_1. BPMN sequence flows have been mapped to π-calculus names, representing dependencies between the agents. For instance, $N717$ can only start after $N534$ has emitted the name $e671$ (an agent emits a name using $'name$ and receives a name by simply stating it, i.e. $name$). To make reasoning possible, all agents representing nodes are placed in parallel in agent N. For accuracy, the identifiers provided by the graphical editor are used. The generated agents can now be imported into existing π-calculus reasoners such as MWB:

```
 The Mobility Workbench
 (MWB'99, version 4.136, built Fri Apr  7 16:02:07 2006)
1
MWB>input "agents.mwb"
MWB>weq N(i,o) S_LAZY(i,o)
The two agents are equal.
Bisimulation relation size = 317.
```

The first statement imports the π-calculus process definitions. Lazy soundness can now be decided using weak open bisimulation between process $N(i, o)$ and $S_{LAZY}(i, o)$. The parameters i and o can be observed for deciding whether the business process is started (by observing i) or the final activity is reached (by o). If o is not observed exactly once, the process is not lazy sound. S_{LAZY} is already proved to be lazy sound, since it simply receives i one time and emits o one time. The *weq* statement now checks if $N(i, o)$ equals S_{LAZY} regarding the observable behavior. As both are equal, also $N(i, o)$ is lazy sound. Interestingly, components of $N(i, o)$ representing lazy activities are still active. However, they do not trigger the final activity (the one that emits o) again.

A counterexample can be given by modifying the parallel split of figure 2 to an exclusive decision. This results in a change of agent $N717$ of the π-calculus representation:

```
agent N717(e671,e794,e792,e791)=e671.t.('e794.N717(e671,e794,e792,e791) +
         'e792.N717(e671,e794,e792,e791) + 'e791.N717(e671,e794,e792,e791))
```

Now, either activity A, B, or C are activated. As can easily be deduced, this leads to a deadlock since the 2–out–of–3–join expects at least two activities to be finished beforehand. By asking MWB using the changed agent $N717$ this can be proved:

```
MWB>weq N(i,o) S_LAZY(i,o)
The two agents are NOT equal.
```

Hence, the modified business process is not lazy sound.

Drawbacks. During early experiments using MWB and ABC for deciding lazy soundness of different business processes, we already discovered several issues. First of all, weak open bisimulation is undecidable in general. Thus, some inputs will not give a result. To make matters worse, current implementations of MWB and ABC rely on depth first search, wasting computing power where breadth

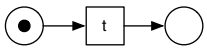
first search would already disprove lazy soundness (i.e. finding other paths that lead to deadlocks and livelocks). However, small to mid-size processes can be proved in reasonable time (see [2] for timing results). Furthermore, due to the non–local semantics of the synchronizing merge pattern (or–join), business processes containing this pattern are never lazy sound. As a concluding remark, also backtracking of errors found in the π-calculus representation to the graphical notation is currently quite difficult. Using optimized reasoners and enhancing the π-calculus representation with additional debugging information, most of the problems can be solved.

5 Related Work

An important piece of related work is Woflan (`http://is.tm.tue.nl/research/woflan.htm`). Woflan is able to prove if two Petri nets are in a certain inheritance relation [12]. Most interesting is checking for *projection inheritance*, that has been derived from process algebra [13]. An informal description is as follows:

> "If it is not possible to distinguish the behaviors of x and y [x and y are Petri nets] when arbitrary tasks of x are executed, but only the effects of tasks that are also present in y are considered, then x is a subclass of y." [12].

Hence, y represents the S_{LAZY} process and x an arbitrary Petri net to check for conformance. S_{LAZY} is given as a Petri net consisting of two places and a transition t:



The transition t can be enhanced with arbitrary process structures. Since projection inheritance ignores remaining tokens in the Petri net, lazy soundness for Petri nets can be proved using Woflan. However, just as with ABC and MWB for π-calculus, the only feedback is a yes/no answer. Furthermore, using Petri nets for proving business processes to be lazy sound has two major drawbacks. First of all, not all workflow patterns can be represented in low-level Petri nets [14]. Thus, the number of possible business processes is restricted. Second, branching bisimilarity used for projection inheritance does not take into account link passing mobility. Link passing mobility is used inside service oriented architectures to represent dynamic binding of interaction partners [9]. Since weak open bisimulation supports link passing mobility, lazy soundness can be extended to interaction soundness. Interaction soundness proves an orchestration to be (lazy) sound regarding also its interactions inside a choreography. Since not all of the interaction partners are statically known (i.e. connected) to the orchestration at design–time, but instead are bound at run–time, a bisimulation technique based on link passing mobility is required.

6 Conclusion

In this paper we introduced a first prototypic tool chain to show the feasibility of deciding lazy soundness using π-calculus. In order to evaluate the tool chain, the scripts and examples are provided at `http://pi-workflow.org`. While the graphical editing and export is currently OS depended (Mac OSX 10.4 required), the conversion of the examples to π-calculus and reasoning runs on a variety of platforms. The very next step regarding the tool chain is to create a stable implementation. This implementation can then be used to analyze existing π-calculus tools as well as the proposed pattern formalizations for conformance regarding lazy soundness.

References

1. van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M.: Business Process Management: A Survey. In van der Aalst, W.M.P., ter Hofstede, A.H., Weske, M., eds.: Proceedings of the 1st International Conference on Business Process Management, volume 2678 of LNCS, Berlin, Springer-Verlag (2003) 1–12
2. Puhlmann, F., Weske, M.: Investigations on Soundness Regarding Lazy Activities. In Dustdar, S., Fiadeiro, J., Sheth, A., eds.: Proceedings of the 4th International Conference on Business Process Management (BPM 2006), volume 4102 of LNCS, Berlin, Springer Verlag (2006) 145–160
3. BPMI.org: Business Process Modeling Notation. 1.0 edn. (2004)
4. Puhlmann, F., Weske, M.: Using the Pi-Calculus for Formalizing Workflow Patterns. In van der Aalst, W., Benatallah, B., Casati, F., eds.: Proceedings of the 3rd International Conference on Business Process Management, volume 3649 of LNCS, Berlin, Springer-Verlag (2005) 153–168
5. Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part I/II. Information and Computation **100** (1992) 1–77
6. Smith, H., Fingar, P.: Business Process Management – The Third Wave. Meghan-Kiffer Press, Tampa (2002)
7. Puhlmann, F.: Why do we actually need the Pi-Calculus for Business Process Management? In Abramowicz, W., Mayr, H., eds.: 9th International Conference on Business Information Systems (BIS 2006), volume P-85 of LNI, Bonn, Gesellschaft für Informatik (2006) 77–89
8. van der Aalst, W.M.P., ter Hofstede, A.H.M., Kiepuszewski, B., Barros, A.: Workflow Patterns. Technical Report BETA Working Paper Series, WP 47, Eindhoven University of Technology (2000)
9. Overdick, H., Puhlmann, F., Weske, M.: Towards a Formal Model for Agile Service Discovery and Integration. In Verma, K., Sheth, A., Zaremba, M., Bussler, C., eds.: Proceedings of the International Workshop on Dynamic Web Processes (DWP 2005). IBM technical report RC23822, Amsterdam (2005)
10. Briais, S.: ABC Bisimulation Checker. Available at: `http://lamp.epfl.ch/~sbriais/abc/abc.html` (2003)
11. Victor, B., Moller, F., Dam, M., Eriksson, L.H.: The Mobility Workbench. Available at: `http://www.it.uu.se/research/group/mobility/mwb` (2005)
12. van der Aalst, W., Basten, T.: Inheritance of Workflows: An approach to tackling problems related to change. Computing science reports 99/06, Eindhoven University of Technology, Eindhoven (1999)

13. Basten, T.: In Terms of Nets: System Design with Petri Nets and Process Algebra. PhD thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (1998)
14. van der Aalst, W.M.P., ter Hofstede, A.H.M.: YAWL: Yet Another Workflow Language (Revised version. Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane (2003)

Configurable Execution Environments for Medical Processes

S. Jablonski*, M. Faerber*, M. Götz*, B. Volz*, S. Dornstauder*, S. Müller**

*) Chair for Databases and Information Systems, University of Bayreuth
Universitätsstrasse 30, 95447 Bayreuth, Germany
**) Chair for Database Systems, University of Erlangen-Nuremberg
Martensstraße 3, 91058 Erlangen, Germany

Abstract. We present the process modeling and execution tools iPM and iPE that can dynamically be adjusted to domain specific requirements. While many publications about iPM are focusing on its modeling flexibility, this paper concentrates on its capability to execute such customized process models. The iPM execution environment (called iPE) is based upon a template driven generation approach: for each domain specific process modeling language, templates are provided which are compiled into a specialized execution environment.

1. Introduction

Gathering and organizing data is a major problem for many, especially complex application domains like the clinical / medical application domain. Here, data from different medical devices have to be collected and prepared for various analyzing steps such as diagnosis and reporting. We investigate such a clinical application currently in a research project funded by the German Research Society [1] at the ophthalmic department of the University of Erlangen-Nuremberg. Our task is to integrate various diagnostic data into a common database (the so called Glaucoma Register) and to provide physicians and researchers with data and tools to effectively and efficiently perform research in that medical realm.

Due to the dual purpose of the application – especially of the medical devices – as medical research infrastructure on one hand side and as infrastructure for the normal clinical consultation on the other hand side, the application scenarios for clinical devices become very difficult and complex. Thus, clinical processes are chosen to describe and illustrate these scenarios. In another DFG funded research project we found out that clinical processes are a powerful means to model complex application systems [2]. Some reasons foster this approach: first, (clinical) processes are recognized to illustrate complex scenarios in a clear and concise way such that a user can easily comprehend their content. Secondly, changes can easily be performed on process models – we will see that changes are a challenge in our application.

In a first step we used standard process modeling tools (e.g. ARIS, VISIO) to describe the clinical processes. However, we were experiencing two major problems:

- The semantics to be described was so complex that the resulting process models were hard to comprehend (Figure 1a, a VISIO example). The physicians could not understand – and thus could not review – this complex process model. As a conse-

quence from this observation, we created application specific process modeling elements that describe application semantics in a compact and comprehensible way which is a first contribution of this paper. Figure 1b depicts the resulting process model. The grey process step comprises the same semantics as the process of Figure 1a. By the way, for the purpose of this paper the content of the process depicted in Figure 1 it is not of interest; the comparison of the two representations in Figure 1a and Figure 1b should rather illustrate that tailored, compact modeling constructs facilitate the readability and comprehension of process models.

- Another major challenge of the clinical domain is its constant change; this feature is primarily due to the research character of the application. To meet this requirement we are also fostering process modeling to describe the application: thereby changes in the application can be reconstructed quickly in the process model. A challenge here is that such a change must not only be reflected in a process model but must also be reflected in the process execution environment. A second major contribution of this paper is to present the design and implementation of a process execution infrastructure that perfectly matches this requirement.

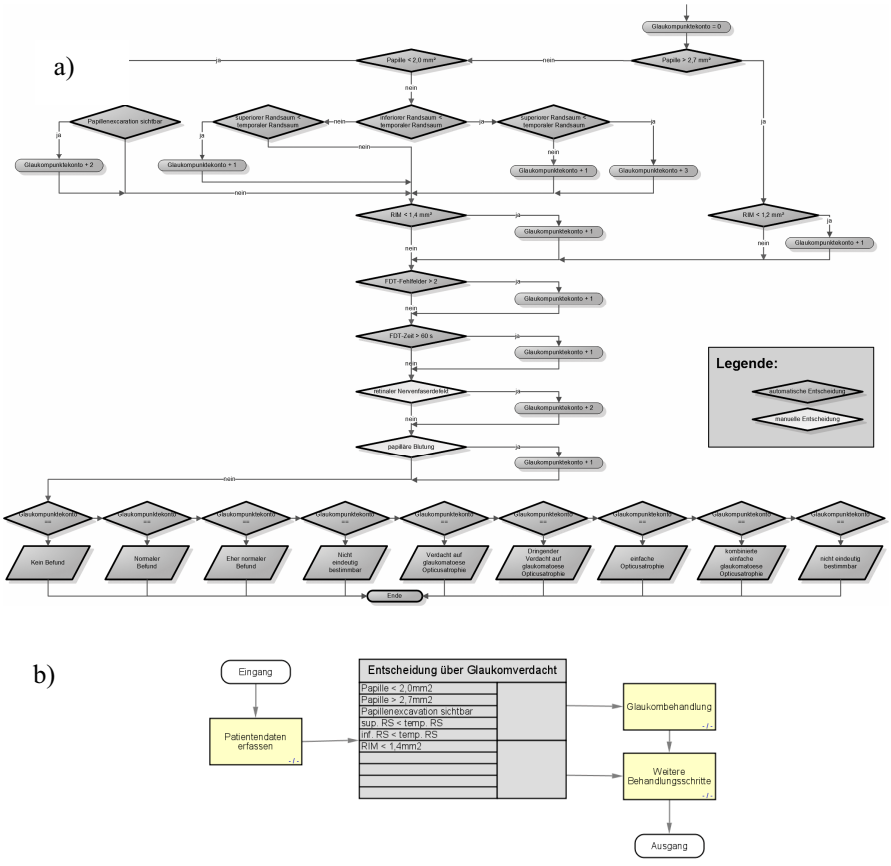


Figure 1: **The advantage of domain specific modeling**

From the above discussion two components of our solution approach can be derived:

- The required use of specialized process modeling constructs needs a domain specific modeling language (Section 2).
- The implementation of clinical process requires a flexible process execution infrastructure. This must be extremely adaptable, since it firstly must be able to interpret domain specific constructs introduced "on the fly", and secondly to interpret, i.e. to execute them in an efficient way. This is a demanding feature since adaptation and flexibility are – in principle – mutually contradictory (Section 3).

If these two requirements can be met, we are able to provide a powerful process modeling and execution environment for flexible process models.

2. Domain Specific Process Modeling

Domain-Specific Modeling is an approach that facilitates communication between domain experts and software developers, as it uses modeling concepts directly connected to the domain the software will be applied in [7]. This way errors caused by misunderstandings between modeling experts and domain experts can be avoided or can be detected earlier [6]. Consequently, for each domain a Domain Specific Language (DSL) must be constructed that contains domain specific modeling elements. However, in our case having a DSL is just the first step. Since modeled processes must be executed subsequently, a domain, i.e. a DSL specific process execution facility must also be provided. The modeling elements contained in a DSL for Process Modeling can be partitioned in two subcategories:

- Elements that are needed in every process model, independent of the current application domain (common process modeling elements)
- Elements that are especially introduced for one application domain (specific process modeling elements)

At the end of this section we will present some examples for common and for specific process modeling elements, respectively.

This partition and the flexibility demanded in Section 1 require a specific modeling concept. We have developed an approach that is called perspective oriented process modeling [2, 4]; this concept is implemented by our process modeling tool iPM (integrated ProcessManager) that will be demonstrated. Its main idea is the following: When a process model is needed that has to be adapted to new requirements frequently, it must be divided into small “pieces” in order to better cope with such changes. This is nothing but the well known principle of "divide and conquer", a synonym for the term "modularization" in software engineering.

The only firm and static part of our perspective oriented process model is referred to as the modeling construct skeleton. This skeleton knows that so-called perspectives are defined; together these perspectives establish a (process) modeling construct. The typical perspectives of a modeling construct for processes are [4]:

- The functional perspective describes the tasks of a process.
- The organizational perspective identifies the agents (e.g. physicians) responsible to perform a task.
- The operational aspect specifies applications needed to execute a process.

- The control flow perspective defines the execution order between workflows.
- The data perspective defines both the in and out parameters of workflows and how data flows between workflow steps.

In order to create a new modeling construct, each perspective must be customized. Therefore, for each perspective a meta model is provided that facilitates customization. Having the perspective oriented process model, (almost) arbitrary modeling languages can be constructed. Hereby, we use the following interpretation: a modeling language is constituted through the modeling constructs it comprises. Thus, for each application domain, a domain specific process modeling language can be provided.

It is out of the scope of this paper to present a complete set of modeling constructs for process modeling, separated into common and specific constructs. However, some examples should illustrate the meaning and purpose. The two major common modeling elements for the functional perspective are elementary and composite process steps. Elementary process steps cannot be refined any more, while composite process steps consist of elementary or other composite process steps. One of the major common modeling elements for the control flow perspective is the choice construct with two or more exits. In the application depicted in Figure 1a, a medical decision has to be made that determines whether a suspicion of Glaucoma must be confirmed or can be abandoned. The whole process consists of two sorts of elements: in the choices (diamonds) certain criteria are interrogated. Depending on the outcome of these questions, certain variables are increased (or decreased). Finally, a decision is taken. We kind of summarize this whole decision into the compact, domain specific construct of Figure 1b: the main criteria for this decision are depicted by the grey modeling construct and can therefore be read and interpreted by a physician who immediately understands which criteria are relevant in the decision process. In order to get more detailed information about the medical algorithm being executed, the physician can zoom into the modeling construct.

The example of Figure 1 shows that compact, customized modeling constructs increase the readability of process models: while the contents of Figure 1a and Figure 1b are the same, the physicians much easier and faster comprehend the meaning of the process in Figure 1b and therefore accept process modeling much more. As we already stated in Section 1 the main challenge is to also provide an execution environment for such flexible modeling constructs. The next section introduces iPE, the execution environment for process models developed with the iPM process modeling tool.

3. Executing Domain Specific Processes: iPE

3.1. Architecture of iPE

For each process model (defined in a domain specific process modeling language) a so-called iPE ("integrated Process Execution") application is generated by the iPE compiler (Figure 2). The iPE application is a domain specific execution environment for processes modeled in a domain specific process modeling language. There are two main inputs for the iPE compiler: first, the process model that has to be executed, and

second, a series of so called iPE templates. Among these iPE templates there is one template defining the domain specific process language used to describe the process model under consideration. Some other templates are needed to construct the components of the generated iPE application as they are also shown in Figure 2 (e.g. Runtime Persistence Service, Dynamic Navigation Service).

The generation of the iPE application starts with a transformation of the corresponding process model into a language neutral representation which can be seen as a superset of the modeling languages iPM can support principally. Secondly, the components of the iPE application are generated from the iPE templates. By exchanging the iPE templates different components can be created. For example, the Runtime Persistence Service can be based on a database system or on a file system, respectively.

The iPE application is based on a three tier architecture which separates the frontend, the data manipulation and the data storage layer from each other by well defined interfaces. The frontend of the iPE application implements the interface for the process participants. Typically, the process users have to input data and interact with the iPE application (e.g. determine the next steps to be executed). This input is then sent to the process control layer. The process control layer forwards the incoming calls to the attached services. According to the use of different iPE templates, different services can be deployed. For instance, the frontend (i.e. the user interface) could be implemented as a web application (running in a standard browser) or a stand-alone application. The following list gives a glimpse on the function of each service shown in Figure 2:

- Runtime Persistence Service: Stores the data of the process during execution in such a way that the path along a process can be easily tracked back afterwards.
- Offline Persistence Service: According to strict regulations in the medical application field, it is necessary to store some parts of the data into special databases for a longer period of time. This service synchronizes such databases by transferring the important data of a process from the Runtime Persistence Service to these storages
- Knowledge Management Service: This service supports the execution of one particular process by providing additional information from a knowledge management system. In the clinical application field one can think of providing information about medical treatment or diseases using this service.
- External Application Execution Service: Often it is necessary to execute external applications like a word processor or retrieving data from a medical device (e.g. an X-ray image, blood pressure).
- Dynamic Navigation Service: The path along a process is not fixed but will be determined during runtime of the process depending on the data available in the process and the process execution history.

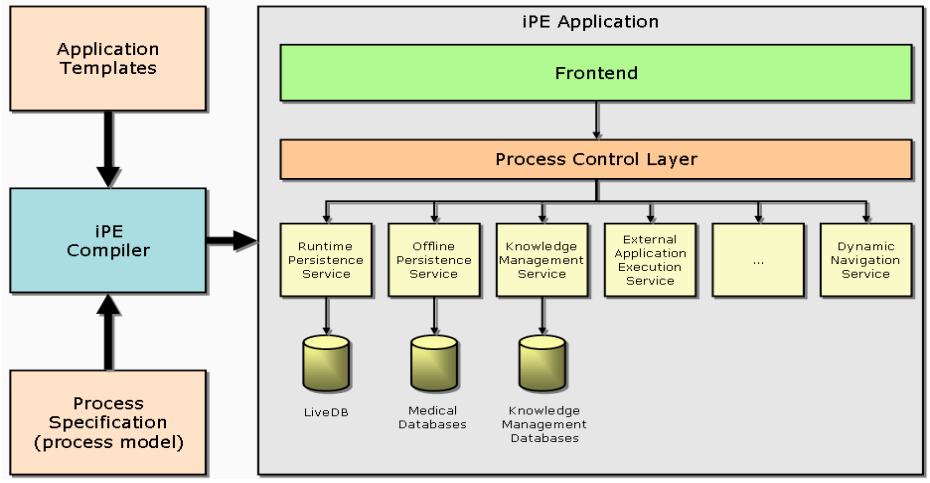


Figure 2: The iPE compiler generates the iPE application

We implemented prototypic templates for the frontend using JavaServer Faces [8]. The iPE compiler (implemented in Java 1.5) then generates a WAR file which can be deployed to every Servlet Container.

3.2. Executing Processes

This section provides a short introduction into the execution of processes by an iPE application. Therefore, we take a short process as example (Figure 3) and explain how the components of the iPE application are used to execute this sample process.

In the step “Anamnesis” patient data (e.g. name and age) are entered. These data are needed in the next step “Examination” as the tests are set up individually for patients of different age. In the last step, the finding of an examination is composed. Consequently, all data used in the process up to now, are needed.

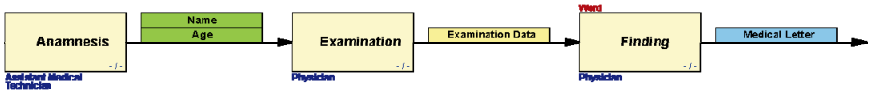


Figure 3: A sample process

To execute a process, e.g. the sample process of (Figure 3), a user must log into the iPE application using the frontend of the iPE application. In this example, we provide a default frontend which runs in a standard web browser. In the step “Anamnesis” some input fields like "name" and "age" are shown which have to be filled by the process user. Another default frontend to enter examination data is used in the process step "Examination". However, the third process step "Finding" is different: here, a word processor (e.g. MS Word) must be called in order to fill out a template for findings. In the following, we briefly show how the components of the iPE application are interacting in order to execute the sample process.

All data entered by the process users are collected by the Process Control Layer (Figure 2) and subsequently forwarded to the underlying services of the iPE application like Runtime Persistence Service (RPS), Knowledge Management Service (KMS), and External Application Execution Service (EAES). After a process step was executed, e.g. the step "Anamnesis", the data collected by the Process Control Layer are stored in a database by the RPS. After having entered such data, the KMS could be called to search for relevant information in the scope of interest (e.g. the glaucoma disease). This search is depending on the data collected at the corresponding process step. To call the KMS is optional whereas calling some other services like the RPS is mandatory. It must be specified in the iPE templates whether service calls are optional or mandatory. In the third step of our sample process model ("Finding"), the EAES needs to be called. It is responsible to start the MS Word text processing system in order to write the diagnostic findings. Hereby, a MS Word template for findings is used, which is specified in the process model. The finding will then be stored in the database (as "Medical Letter"). In order to accomplish this, the RPS is called. If in the course of process execution, this finding must be revisited again, the two services EAES and RPS are again responsible to present this finding to the process user.

The iPE application also assigns user or roles to process steps. For example, while the first step in the process (Figure 3) must be executed by an Assistant Medical Technician, the second and the third step must be performed by a physician.

According to the scenario presented above, the various services are called by the Process Control Layer in order to execute process models. However, even this small scenario depicts how flexible process execution can be organized. Consequently process models (i.e. the process languages) and iPE templates can be customized in such a way that they ideally fit the requirements of specific process execution semantics.

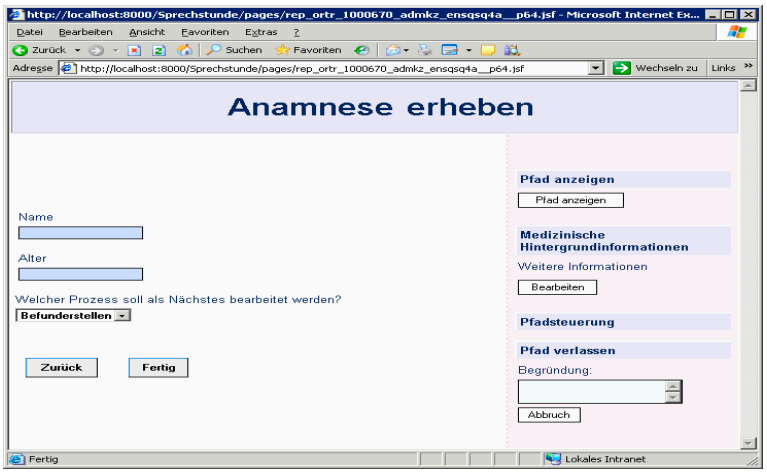
4. Demonstration: Presented Application Areas

While this paper mainly focuses on the medical domain, the iPE execution environment can also be used for other application scenarios. This is due to its Meta modeling approach and its modular architecture. To support another application domain means to provide an alternative domain specific process modeling language. While the process modeling language of our medical domain is called iPM4MED, we are also able to provide the process modeling language iPM4QM for the quality management domain. This process language is compliant to the reference models of the ISO 15504 quality standards [5]. In our demonstration we will present both modeling domains, the medical domain (iPM4MED, cf. [3]) and the quality management domain (iPM4QM). For both domains we will demonstrate

- how to specify new modeling constructs,
- how to define and modify new process models, and
- how to execute such process models for different application domains.

Through these three demonstrations we can show how to extend a modeling language and how to execute processes that are defined according to this customized modeling language. That way, the whole flexibility of our two tools iPM (for process modeling) and iPE (for process execution) will be shown. Below, you'll find a screenshot of the iPE frontend. On the left hand side, the fields to input data for the process step "An-

amneses" can be seen. On the right hand side, context information provided by the KMS can be shown.



References

1. Special Research Area 539: „Glaukome einschließlich Pseudoexfoliationssyndrom“ Homepage. http://www.sfb539.forschung.uni-erlangen.de/. Retrieved 2006-05-30.
2. Jablonski, S.; Lay, R.; Meiler, C.; Müller, S.; Hümmer, W.: Data Logistics as a Means of Integration in Healthcare Applications. Proceedings of the 2005 ACM Symposium on Applied Computing (SAC) - Special Track on Computer Appplications in Health Care, Santa Fe, New Mexico, 2005.
3. Jablonski, S.: Process Based Data Logistics: Data Integration for Healthcare Applications. ECEH 2006 (1st European Conference on eHealth Fribourg, Switzerland), 2006.
4. Jablonski S., Bußler C., 1996. Workflow management - modeling concepts, architecture and implementation. London. International Thomson Computer Press, 1996.
5. Jablonski, S.; Faerber, M.; Schlundt, J.; Bridging the gap between SPICE Reference Processes and OU Processes: An iterative business process modelling approach. SPiCE 2006 (Proceedings of the 6th International SPICE Conference on Process Assessment and Improvement (ISO/IEC 15504 standard)), 2006.
6. Cook, S.: Domain-Specific Modeling an Model Driven Architecture. MDA Journal, January 2004.
7. Luoma, J.; Kelly, S.; Tolvanen, J.: Defining Domain-Specific Modeling Languages: Collected Experiences. Proceedings of the 4th OOPSLA Workshop on Domain-Specific Modeling (DSM’04), Vancouver, British Columbia, Canada, Oct 2004, Computer Science and Information System Reports, Technical Reports, TR-33, University of Jyväskylä, Finland, 2004
8. Sun Developer Network: JavaServer Faces Technology, http://java.sun.com/javaee/javaserverfaces/, retrieved 2006-07-11

Repository for Business Processes and Arbitrary Associated Metadata

Jussi Vanhatalo[1,2], Jana Koehler[1], and Frank Leymann[2]

[1] IBM Research GmbH, Zurich Research Laboratory,
Säumerstrasse 4, 8803 Rüschlikon, Switzerland
{juv, koe}@zurich.ibm.com
[2] Institute of Architecture of Application Systems, Universität Stuttgart,
Universitätsstraße 38, 70569 Stuttgart, Germany
frank.leymann@informatik.uni-stuttgart.de

Abstract. We have published a repository for storing business processes and associated metadata. The BPEL Repository is an Eclipse plug-in originally built for BPEL business processes and other related XML data. It provides a framework for storing, finding and using these documents. Other research prototypes can reuse these features and build on top of it. The repository can easily be extended with new types of XML documents. It provides a Java API for manipulating the XML files as Java objects hiding the serialization and de-serialization from a user. This has the advantage that the user can manipulate the data as more convenient Java objects, although the data is stored as XML files compliant with the standard XML schemas. The data can be queried as Java objects using an object-oriented query language, namely the Object Constraint Language (OCL). Moreover, the flexible design allows the OCL query engine to be replaced with another engine based on other query language.

1 Introduction

Interoperability based on several XML standards is one of the corner stones of Web services. The Business Process Execution Language for Web Services (BPEL) [2] is the defacto industry standard for representing business processes. It is tightly related to other XML standards, such as the Web Service Definition Language (WSDL) and XML Schema. In addition, arbitrary metadata represented in XML format may be associated to business processes depending on the context and applications that use the data.

XML data is commonly used by different applications. However, currently managing the documents and searching information from their contents is laborious and inefficient. It is beneficial to store the data in a repository that takes care of data access and executes queries. Although it is important for interoperability to exchange data in XML format across organizations and systems, it is often more convenient for a developer to manipulate the data as Java objects, instead of XML. Our goal was to build a business process repository that stores data as documents compliant to the XML standards, but allows applications to be implemented directly on the Java representation of the data model.

We have implemented the BPEL Repository, which is an Eclipse plug-in built to store business processes together with other XML data. It provides a framework for storing, finding and using these documents. Other research prototypes can reuse these features and build on top of it. The repository can easily be extended with additional XML schemas because of its flexible architecture. By default it supports the common Web service standards, such as BPEL, WSDL and XML schema, and it can easily be extended to support other XML schemas for business processes and other data.

The object-oriented approach frees developers from the burden of the underlying XML data model and allows them to concentrate on the object model of their application, which they usually know well. The Eclipse Modeling Framework (EMF) [10] is used to hide data serialization and de-serialization from the user. The framework takes care of representing the XML data as EMF objects that are Java objects. As a novel feature, it is possible to query the XML files as EMF objects using an object-oriented query language, namely the Object Constraint Language (OCL) [7] that is part of the UML specification. Native XML databases support typically XQuery as their query language. A major advantage of OCL over an XQuery is its ability to navigate through the data model and follow all the associations of an object model. In contrast, XQuery forces the user to formulate the queries based on the tree structure of the underlying XML schema.

In contrast to our file system based solution, there are other business process repositories [12] [14] that are built on top of a database system. However, their database schemas are created manually. Flexibility is an advantage of the BPEL Repository, because EMF is used to automatically generate support for new and modified XML schemas. In research projects, data structures are often modified and new ones are introduced. The automatization makes adapting these changes easier. Nevertheless, repositories based on a database have typically better performance and scalability than our solution.

The BPEL Repository was recently published in IBM alphaWorks [16] with special licensing terms for academic use. The software has been integrated with a change management system called CHAMPS [3], [13].

2 Solution

The architecture of the BPEL Repository is presented in Figure 1. All components are plug-ins on the Eclipse platform. The core component of the solution is the Repository API, which provides an application programming interface for external software to build on.

The Repository User Interface (UI) is an example implementation that uses the Repository API. However, it is also a useful graphical user interface to manage the contents of the repository. The user interface is integrated in the Eclipse workbench and built on the Standard Widget Toolkit (SWT) and JFace libraries.

The Data Handler is a sub-component that takes care of the data access on a file system. It abstracts the choice of the storage medium from the other

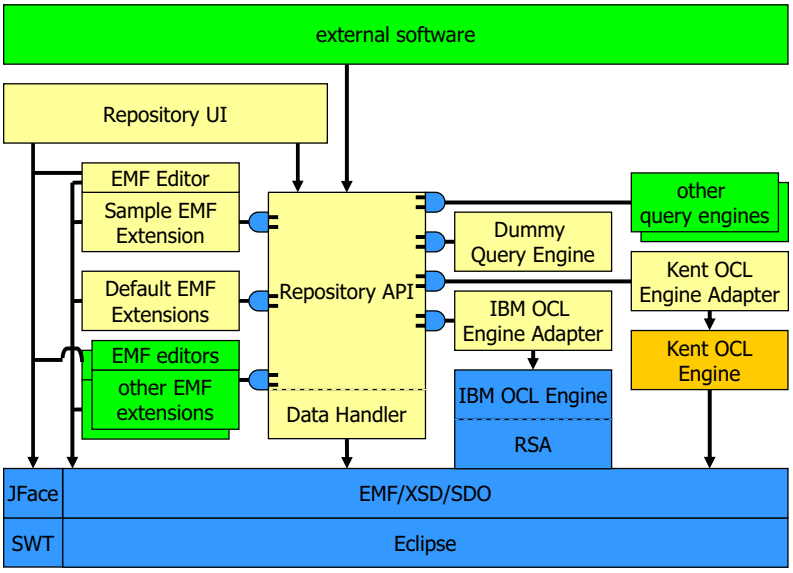


Fig. 1. Repository architecture showing components and technologies used.

components. The data access component could be replaced with another one storing data in a database by using a technology such as the Service Data Objects [10]. The Data Handler uses the Eclipse Modeling Framework to serialize EMF objects into XML files and de-serialize the files back to EMF objects. Thus, all repository components manipulate data as EMF objects rather than of XML.

2.1 Flexibility of Manipulating Data as EMF Objects

In the repository, data is represented as EMF objects. Therefore, all data must have an EMF model. However, the EMF model can be automatically generated from an XML schema, a UML class diagram or Java classes [4], [10]. As in the business process management context data is often stored as XML conforming standardized XML schemas, it is trivial to obtain EMF models for XML files. The repository can be extended to support a new data type by plugging in the EMF model of this new data type.

The components providing EMF models for the repository are shown on the left-hand side of the Repository API in Figure 1. The Default EMF Extensions plug-in contains EMF models for BPEL, WSDL and XML schema standards. Thus, the repository supports the respective file types by default. This component can be replaced by another component supporting a different version of these standards or completely different file types. Because the Eclipse plug-in mechanism is used, this does not require any modifications in the other parts of the repository.

Similarly, other EMF extensions can be plugged into the repository. In the evolving research community, extensibility is an asset. For instance, in the context of combining business process management with semantic Web, the repository can easily be extended to support a new document type containing metadata related to a business process.

The Sample EMF Extension contains an EMF model that is used in the user guide [16] to illustrate, step by step, how to create an EMF model and plug it into the repository. It is also explained how EMF can be used to automatically generate a graphical editor for the instances of an EMF model.

First, the data structure can be modeled as a UML class diagram, which is usually much faster than describing the same structure as an XML schema. Next, the UML class diagram is transformed to an EMF model. An editor can be automatically generated for the EMF model. An XML schema can be generated from the EMF model, if desired. In any case, instances of an EMF model can be serialized to interoperable XML files. Instances of the model can be generated for testing purposes with the editor, which takes care of proper syntax. Finally, the EMF model is plugged into the repository, which persists the data and provides capabilities for querying data. A chief advantage is that queries can be formulated using the same object-oriented model as was used to create the data structure in the first place. Thus, the XML representation is used only for interoperability with other systems, and the developers need not bother with the concrete XML syntax.

2.2 Query Engines

We used existing query engines with the repository. The repository handles the iteration over the queried objects, but each sub-query is executed in the query engine plugged into to the repository. It is possible to change the query engine to another pre-registered one between queries. The available query engines are shown on the right-hand side of the Repository API in Figure 1. The repository has been tested with two OCL query engines, that query Java objects with an object-oriented query language, namely OCL.

If the repository is installed on top of the IBM Rational Software Architect (RSA) product, the OCL engine of the latter can be used. However, as we did not want to limit the repository to a single commercial query engine or a specific query language, the query engine interface has been built generic. Therefore, the IBM OCL engine is plugged into the repository using an adapter. The IBM OCL Engine Adapter is delivered together with the repository.

Another OCL engine was built at the University of Kent [1]. It is an open source tool that can be plugged into the repository using the Kent OCL Engine Adapter. The Eclipse Modeling Framework Technology project [11] is building another open source OCL engine that could also be adapted to the repository. We have not yet implemented the corresponding adapter because this OCL engine is still under development.

It is straightforward to plug a new query engine into the repository or adapt an existing query engine for it. An example of how to adapt an OCL engine

is included in the user guide [16]. The Dummy Query Engine is an example implementation to show how a new query engine can directly be integrated into the repository. Thus, also query engines based on another query languages can be used.

The repository is not aware of the query language that is used. The repository merely passes the query and other parameters from the user interface or external software to the query engine selected together with the EMF object that is to be queried. Thus, any query engine that can execute queries on EMF objects can be plugged into the repository.

One limitation of the query mechanism is that the performance is only linear compared with the number of documents that are queried. Indexing data or other ways to improve the query performance are not used. However, this performance has been sufficient for research prototypes. For example, querying 100 BPEL files took 3 seconds on a laptop in our performance tests [15]. One way to improve OCL queries would be to map them to a query language, such as XQuery, that is natively supported by a database system. In that case, the repository would also be built directly on top of the database system. Some work on mapping OCL to XQuery already exists [5], [6].

2.3 Data Structure

The data is organized in a tree of organizations. The organizations are mapped to directories in a file system. Each organization may contain a business process and associated metadata grouping the related files together. In addition to these data documents, a descriptor document is stored in each organization. It contains the file type and the role of each data document in the organization. This information is used to make the conversion between EMF objects and XML files. In addition, the role describes how the data document is related to the other documents in the organization. For instance, a WSDL file stored with the repository may contain the public interface or the partner links of the BPEL business process.

Queries can be applied to files with a specified role in an organization, a list of organizations, or a list of sub-trees in the organization hierarchy. Related files can be searched based on their roles.

Data can be accessed from the file system as XML files and through the repository as EMF objects. Any directory in a file system can act as the root organization of the repository contents. The data in the repository can be moved to another location or a computer as simply as copying the directories.

2.4 Usage Scenario

The repository has been deployed with a change management system called CHAMPS. As part of the solution, planning algorithms are used to facilitate the automatization of the change and configuration management. The plans are stored as BPEL files into the BPEL Repository. In addition, the plans are analyzed and the results are stored as metadata associated to the plans. The

metadata includes information about the plan such as its number of activities, degree of concurrency, execution duration and correctness [13].

Storing the analysis results as metadata enables reuse of the data, and unnecessary recomputing of the results can be avoided. The suitable plans are found from the repository by querying the content of the plans and their associated metadata. For example, the plans that are structurally correct can be found by querying the metadata. Among these plans the ones that reach a specified goal can be found with a subsequent query.

During the development of the system, trying out different alternatives of the metadata schema was uncomplicated, since the data structure was designed as a UML class diagram and the corresponding EMF classes were used as the basis of the implementation and the OCL queries. The developers were able to avoid completely working with the XML representation of the data, because it was automatically generated and used only as the serialization format behind the scenes.

3 Conclusion

The repository has already been proved useful for IBM internal research prototypes [13]. By publishing the repository, we wanted to make it widely available to the research community as we are interested in more user experience with it. We would also be interested in finding out how convenient users find OCL as a query language, because currently OCL is more common as a language to express constraints rather than queries.

As next steps, we plan to contribute our experiences gained while building the repository to the IP-SUPER project [8], [9] funded by European Union. The project merges business process management with semantic Web services. As part of the project, we plan to build a business process library, most likely on a database system rather than a file system in order to improve the query performance for more extensive querying purposes. This would also be beneficial, when the repository is used for searches based on an ontology or as a component of a business process execution engine.

References

1. Dave Akehurst and Octavian Patrascoiu. Object constraint language library. Web site, June 2004. http://www.cs.kent.ac.uk/projects/ocl/.
2. Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *Business Process Execution Language for Web Services.* OASIS Org., 2003. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.
3. Aaron B. Brown, Alexander Keller, and Joseph L. Hellerstein. A model of configuration complexity and its application to a change management system. In *Proceedings of the 9th International IFIP/IEEE Symposium on Integrated Management (IM 2005)*, May, 2005.

4. Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *Eclipse Modeling Framework*. The Eclipse Series. Addison-Wesley Professional, 2003.
5. Ahmed Gaafar and Sherif Sakr. Proposed framework for integrating XML/XQuery and UML/OCL. In *Proceedings of the 7th Conference in the UML series (UML2004)*, pages 241–259, Lisbon, Portugal, 2004.
6. Ahmed Gaafar and Sherif Sakr. Towards a framework for mapping between UML/OCL and XML/XQuery. In *Proceedings of the IADIS e-Society 2004 Conference (ES2004)*, pages 241–259, 2004.
7. Object Management Group. *OCL 2.0 Specification*. OMG, 2005. http://www.omg.org/docs/ptc/05-06-06.pdf.
8. Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic business process management: A vision towards using semantic web services for business process management. In *Proceedings of the IEEE ICEBE 2005*, pages 535–540, Beijing, China, October 2005.
9. IP-SUPER. Semantics utilised for process management within and between enterprises. Web site, April 2006. http://www.ip-super.org/.
10. Eclipse Org. Eclipse modeling framework. Web site. http://www.eclipse.org/emf/.
11. Eclipse Org. Eclipse modeling framework technology. Web site, 2006. http://www.eclipse.org/emft/projects/ocl/.
12. Minrong Song, John Miller, and Ismailcem Arpinar. RepoX: XML repository for workflow designs and specifications. Technical Report #UGA-CS-LSDIS-TR-01-011, University of Georgia, August 2001.
13. Biplav Srivastava, Jussi Vanhatalo, and Jana Koehler. Managing the life cycle of plans. In *Proceedings of the 17th Innovative Applications of Artificial Intelligence Conference*, pages 1569–1575, Pittsburgh, Pennsylvania, USA, 2005.
14. Tammo van Lessen. Konzipierung und Entwicklung eines Repository für Geschäftsprozesse. Master's thesis, Institute of Architecture of Application Systems, University of Stuttgart, March 2006.
15. Jussi Vanhatalo. Building and querying a repository of BPEL process specifications. Master's thesis, Helsinki University of Technology, Institute Eurecom and University of Nice – Sophia Antipolis, September 2004.
16. Jussi Vanhatalo. BPEL Repository. IBM alphaWorks, April 2006. http://www.alphaworks.ibm.com/tech/bpelrepository.

Maestro for Let's Dance: An Environment for Modeling Service Interactions

Gero Decker[1], Margarit Kirov[1], Johannes Maria Zaha[2], Marlon Dumas[2]

[1] SAP Research Centre, Brisbane, Australia
(g.decker,margarit.kirov)@sap.com
[2] Queensland University of Technology, Brisbane, Australia
(j.zaha,m.dumas)@qut.edu.au

Abstract. In emerging web service development approaches, the description of interactions both from a global and from a local perspective plays an increasingly important role. In earlier work we presented a visual language (namely Let's Dance) for modeling service interactions at different levels of abstraction. In this paper we present a modeling tool for Let's Dance. The tool supports the static analysis of global models, the generation of local models from global ones, and the interactive simulation of both local and global models.

1 Introduction

As the first generation of web service technology based on XML, SOAP, and WSDL gains maturity, a second generation targeting collaborative business processes is gestating. Development methods associated to this second generation of web services generally rely on the explicit representation of service interaction behavior from two complementary perspectives: one where interactions are seen from the perspective of each participating service, and the other where they are seen from a global perspective. This leads to two types of models: In a *global model* (also called a *choreography*) interactions and their dependencies are captured from the viewpoint of an ideal observer who oversees all interactions between a set of services. Meanwhile, a *local model* focuses on the perspective of a given service, capturing only those interactions that directly involve it. Local models are suitable for implementing individual services while choreographies are instrumental during the early phases of analysis and design, when domain analysts need a global picture of the system.

In previous work [4] we have identified requirements for a service interaction modeling language and argued that existing languages, e.g. WS-CDL [2], fail to fulfill these requirements. Indeed, existing languages are targeted at application developers rather than at domain analysts who play a key role in the construction of these models. Accordingly, we have designed a language, namely Let's Dance, intended to support analysts in capturing both global and local service interaction models. This paper introduces a tool that enables analysts to capture and to analyze Let's Dance choreographies. After analysis, the local models for

the parties involved in the choreography can be generated and the execution of both global and local models can be interactively simulated.

The paper is structured as follows. Section 2 gives an overview of the Let's Dance language. The architecture and the features of the tool are presented in Section 3. Section 4 discusses future extensions.

2 Let's Dance Choreographies

A choreography consists of a set of interrelated service interactions corresponding to message exchanges. At the lowest level of abstraction, an interaction is composed of a message sending action and a message receipt action (referred to as communication actions). Communication actions are represented by non-regular pentagons (symbol ▷ for send and ▷ for receive) that are juxtaposed to form a rectangle denoting an elementary interaction. As illustrated in Figure 1, a communication action is performed by an actor playing a role, specified at the top corner of a communication action. Roles are written in uppercase and the actor playing this role (the "actor reference") is written in lowercase between brackets. The name of the message type for the receive actions can be omitted (since the same type applies for both send and receive).

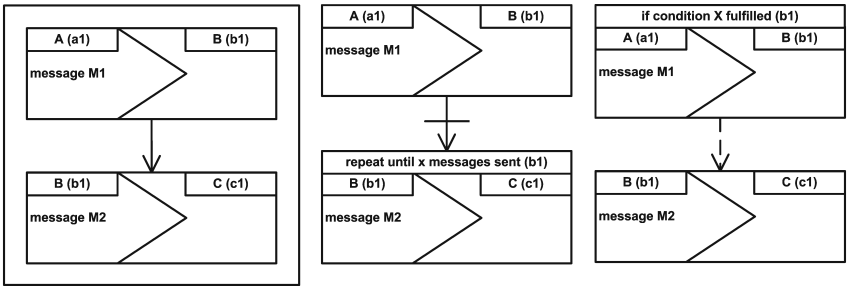


Fig. 1. Constructs of Let's Dance

Interactions can be inter-related using the constructs depicted in Figure 1. The relationship on the left-hand side is called "precedes" and is depicted by a directed edge: the source interaction can only occur after the target interaction has occurred. That is, after the receipt of a message "M1" by "B", "B" is able to send a message "M2" to "C". The rectangle surrounding these two interactions denotes a composite interaction, which can be related with other interactions with any type of relationship. The relationship at the center of the figure is called "inhibits", depicted by a crossed directed edge. It denotes that after the source interaction has occurred, the target interaction can no longer occur. That is, after "B" has received a message "M1" from "A", it may not send a message "M2" to "C". The latter interaction can be repeated until "x" messages have been sent, which is indicated by the header on top of the interaction. The actor

executing the repetition instruction is noted in brackets. Finally, the relationship on the right-hand side of the figure, called "weak-precedes", denotes that "B" is not able to send a message "M2" until "A" has sent a message "M1" or until this interaction has been inhibited. That is, the target interaction can only occur after the source interaction has reached a final status, which may be "completed" or "skipped" (i.e. "inhibited"). In the example, the upper interaction has a guard assigned, which is denoted by the header on top of the interaction. This interaction is only executed if the guard evaluates to true. The actor who evaluates the guard is noted in brackets.

3 Tool Overview

Figure 2 shows a screen shot of Maestro for Let's Dance. The palette on the left-hand side contains the diagram elements. Below this palette, layout options are accessible. The main drawing area in the middle. On the right, there is a pane for editing the properties of the selected element as well as a navigator that provides an overview over the diagram. The analysis and simulation functionality can be accessed via the "Let's Dance" menu item in the menu bar at the top. Analysis results and simulation status are shown within the editing area.

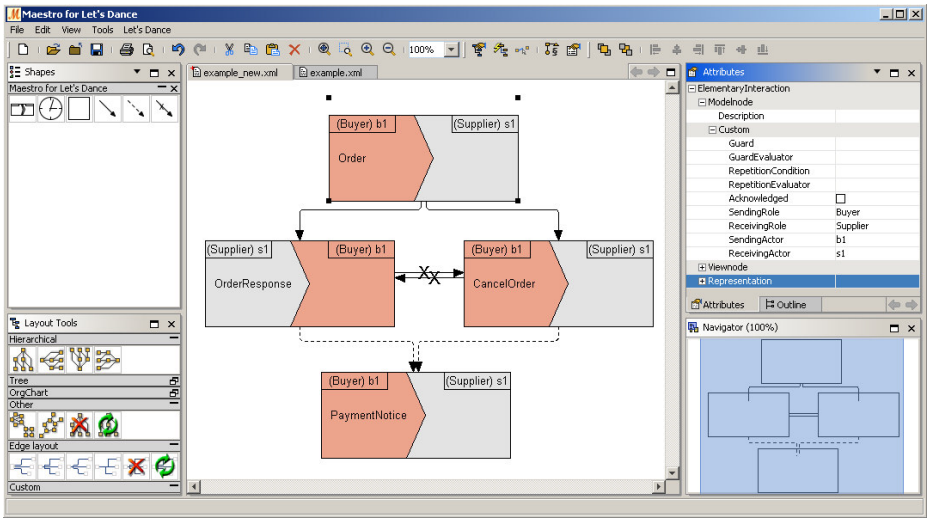


Fig. 2. Screen shot of Maestro for Let's Dance

Maestro for Let's Dance is built on top of the Maestro visual language framework developed at SAP. It is the foundation for various modeling environments including: Maestro for BPMN, an editor for the Business Process Modeling Notation, Maestro for BPEL, a modeling environment for the Business Process Execution Language, and Maestro for SAM (Status-and-Action Management), a modeling and simulation environment for business object lifecycles.

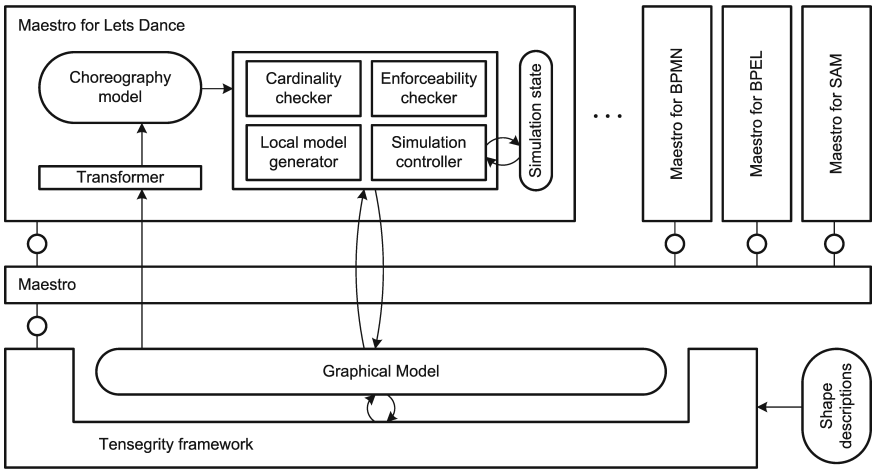


Fig. 3. Architecture overview

Figure 3 describes the main architecture of Maestro for Let's Dance using the Fundamental Modeling Concepts (FMC) block diagram notation [3]. A core component of the Let's Dance prototype is the Tensegrity framework[3]. It provides basic functionality that is needed for graphical editors such as rendering functionality, editing facilities (selection, creation, deletion, resizing, etc. of diagram elements), event propagation mechanisms, a command stack and a persistency service for diagrams. The Maestro framework extends Tensegrity and provides utility functionality for attribute management and refined application skeletons.

Tensegrity and Maestro can be compared to eclipse GMF[4] (Graphical Modeling Framework). Shape descriptions define what diagrams consist of and how they look like. Anchor points, resizing behavior, color schemes, line widths and line decorations can be defined for diagram elements. Furthermore, the palette is configured. A strong point about GMF is that it very clearly separates the data model from its graphical representation. However, GMF was not chosen because at the time of development releases were still in a pre-1.0 state and significant changes in the API from release to release caused major refactorings. It would have been an option to use eclipse GEF[5] (Graphical Editing Framework), the foundation of GMF, without using GMF itself. The fact that GEF does not include layouting functionality like it is the case for Tensegrity and the possibility of integrating the tool with Maestro for BPMN and Maestro for BPEL in the future were the final arguments for choosing Maestro.

Choreographies are described using two different data structures in the tool. The diagram data structure is optimized for the usage within the editor. E.g. entities in the data structure directly relate to edges and nodes in the diagram and layout information is attached to the entities. The choreography model data

[3] See http://www.tensegrity-software.com/
[4] See http://www.eclipse.org/gmf/
[5] See http://www.eclipse.org/gef/

structure does not contain any layout information and follows the Let's Dance meta-model introduced in [4]. The choreography model is then used as input for the analysis plugins and the simulation engine. A model transformation takes place every time a check is triggered or a simulation is started. A reverse mapping from entities in the choreography model to the entities in the diagram model is later on used for displaying output of the checkers or the simulation engine. Since EMF[6] (Eclipse Modeling Framework) includes functionality to produce XMI[7](XML Metadata Interchange)-compliant files and since there is a good integration of EMF with the UML modeling environment Rationale Rose[8], EMF was used for implementing the choreography data model.

A precondition for analyzing a choreography model is that it complies to the well-formedness criteria defined in [5]. Therefore, a well-formedness check precedes every analysis as well as the generation of local models. Examples for ill-formedness could be e.g. an unspecified actor for a send or receive action or a cycle of precedes and weak-precedes relationships.

Cardinality checker. Maestro for Let's Dance provides a *cardinality checker* that is able to identify how many times an interaction can occur, at least and at most, within one execution of a choreography. The cardinality checker takes as input a choreography and classifies its interactions into five groups: (0,0): interactions that will never be executed (i.e. unreachable); (0,1): interactions that may be executed once or skipped altogether (i.e. conditional); (1,1): interactions that will be executed exactly once; (0,n): interactions that may be executed any number of times; and (1,n): interactions that will be executed at least one.

The cardinality checker can help modelers to detect semantical errors, such as unreachable interactions or interactions that may be skipped against the modeler's intent. It can also be used to determine characteristics required from the communication channels. For example, if an interaction can be executed multiple times, the corresponding channel may be required to guarantee orderly delivery.

Cardinality analysis is largely based on reachability analysis for which we have presented two alternatives in previous work: In [1] the algorithm is based on π-calculus bi-simulation whereas in [5] an ad-hoc algorithm is introduced. It turned out that because of the computational complexity of bi-simulation analysis only very small choreographies could be processed in a reasonable time. The second algorithm has low polynomial complexity and scales up to real-world choreographies. Therefore, it was chosen for the tool.

Enforceability checker. It has been shown that there might be relations between interactions in a choreography model that cannot be enforced locally. I.e. It turns out that not all global models can be mapped into local ones in such a way that the resulting local models only contain interactions explicitly captured in the choreography model and they collectively enforce all the constraints expressed in

[6] See http://www.eclipse.org/emf/
[7] See http://www.omg.org/technology/documents/formal/xmi.htm
[8] See http://www.ibm.com/software/rational

the global model. A counter-example is given in Figure 4. In this choreography it is not possible to enforce the "precedes" constraint between the two interactions without introducing additional interactions. Indeed, how can actors 'c' and/or 'd' know that the interaction between 'a' and 'b' has taken place in the absence of any interaction between actors 'a' and 'b' on the one hand, and actors 'c and 'd' on the other? Thus, either the model needs to be enhanced with an interaction between a/b and c/d, or the sequential execution constraint will not be enforced (i.e. the interactions can occur in the opposite order from the perspective of an ideal observer). Since domain analysts are supposed to sign off on a choreography model it is undesirable to automatically introduce implicit interactions to ensure the fulfilment of constraints. Instead, the modeler should be warned of such issues, so that (s)he can refine the model as needed.

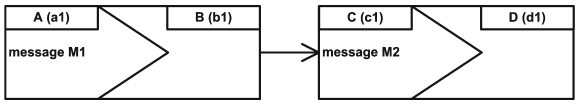


Fig. 4. Example of a non-locally enforceable choreography

Accordingly, Maestro for Let's Dance incorporates an enforceability checker that identifies non-enforceable constraints in a choreography model and reports them to the modeler. The algorithm used for this purpose is described in [5].

Simulation In order to give choreography modelers a better idea of the semantics of their models, the tool offers the possibility to simulate choreography instances. The formalization of the execution semantics of Let's Dance were presented in [1]. It was used as a blueprint for the implementation of the simulation engine.

An interaction can be in one of the four states *initialized* (visualized as yellow), *enabled* (green), *skipped* (red) and *completed* (gray). Each instance starts in the state *initialized*. The instance can now be skipped or it can be enabled. Therefore, the transitions to the states *skipped* and *enabled* are possible. If an instance is in state *skipped* nothing can happen to it any more. If an instance is in state *enabled* the interaction can execute. During the execution the instance can be skipped. Otherwise, it will eventually complete execution and move to state *completed*. In the case of guarded interactions the user can decide whether the interaction should be executed or if it should be skipped. In the case of repeated interactions the user can decide whether the interaction should be executed once more or if the repetition is terminated.

Local model generator. In [5] we have presented an algorithm for generating local models. This algorithm is implemented in Maestro for Let's Dance. A new diagram containing all the interactions where a specific actor is involved is generated and the layout functionality is applied to it.

4 Outlook

The next step in the development of "Maestro for Let's Dance" is the refinement of the generation of local models. The generation of local models with implementation configurations will be an important step towards the generation of executable models, where BPEL will be the first target language. Other analysis functions will be added. Conformance checking between local models and choreography models will be one of the focus areas.

References

1. G. Decker, J. M. Zaha, M. Dumas: *Execution Semantics for Service Choreographies.* Preprint # 4329, Faculty of IT, Queensland University of Technology, May 2006. http://eprints.qut.edu.au/archive/00004329
2. N. Kavantzas, D. Burdett, G. Ritzinger, and Y. Lafon. *Web Services Choreography Description Language Version 1.0*, W3C Candidate Recommendation, November 2005. http://www.w3.org/TR/ws-cdl-10.
3. A. Knopfel, B. Grone, P. Tabeling: *Fundamental Modeling Concepts: Effective Communication of IT Systems.* Wiley, May 2006.
4. J. M. Zaha, A. Barros, M. Dumas, A. ter Hofstede: *Lets Dance: A Language for Service Behavior Modeling.* Preprint # 4468, Faculty of IT, Queensland University of Technology, February 2006. http://eprints.qut.edu.au/archive/00004468
5. J. M. Zaha, M. Dumas, A. ter Hofstede, A. Barros, G. Decker: *Service Interaction Modeling: Bridging Global and Local Views.* In Proceedings of the 10th International EDOC Conference, Hong Kong, 2006.

Middleware Support for BPEL Workflows in the AO4BPEL Engine

Anis Charfi, Mira Mezini

Software Technology Group
Darmstadt University of Technology
{charfi,mezini}@informatik.tu-darmstadt.de

Abstract. This paper focuses on middleware concerns in BPEL workflows. When looking at those workflows from the implementation perspective, we observe that they have several BPEL-specific middleware requirements, which are not supported by current WS-* specifications and by most BPEL engines available to date. This demo paper will show the AO4BPEL Engine, which implements a container framework that allows the specification and enforcement of middleware requirements in BPEL processes. A *deployment descriptor* is used to specify the quality of service requirements of BPEL activities. A light-weight and aspect-based *process container* is used to enforce those requirements by calling dedicated *middleware Web Services*. We implemented those middleware Web Services by extending open source implementations of WS-* specifications for security, reliable messaging, and transactions.

1 Introduction

In BPEL [11], a composite Web Service is implemented by means of a workflow process, which consists of a set of interactions between the composition and the partner Web Services along with the flow of control and data around those interactions. Whilst the functional side of the composition is specified by the BPEL process, it is unclear how to handle non-functional middleware concerns such as security, reliable messaging, and transactions.

The BPEL specification does not address middleware issues and leaves that for good reasons to BPEL implementations, which should support these "deployment issues" somehow. We observe however, that current BPEL engines do no provide appropriate middleware support for BPEL processes. In addition, it is widely assumed that WS-* specifications such as WS-Security [18] and WS-ReliableMessaging [4] are sufficent to cope with the middleware requirements of BPEL processes, for example by attaching appropriate policies [5] to the WSDL of the composite Web Service or its partners. We argue that WS-* specifications support only some of BPEL middleware requirements, namely those which have corresponding operations and messages in WSDL. There are other BPEL-specific middleware requirements that cannot be mapped to WSDL and SOAP as they require knowledge about the process, its activities, and BPEL semantics. These requirements are not supported by WS-* specifications.

In [7], we presented a container framework, which addresses the problems of *specification* and *enforcement* of middleware requirements in BPEL processes. The framework introduces a *deployment descriptor*, which specifies the middleware requirements of the process activities and a *process container*, which intercepts the execution of the activities at well-defined points and plugs in calls for dedicated *middleware Web Services* to enforce those requirements. In this paper, we present an implementation of that framework on top of the AO4BPEL engine, which is an aspect-aware engine based on IBM's BPWS4J [15]. We will also present some BPEL middleware Web Services that we developed by extending open source implementations of WS-* specifications.

2 Middleware Requirements in BPEL Workflows

We distinguish *simple requirements*, which correspond to BPEL messaging activities and could be supported somehow by using existing WS-* specifications and *complex requirements*, which are specific to BPEL. We will elaborate on reliable messaging, security, and transactions.

2.1 Reliable Messaging

A simple requirement of BPEL messaging activities is the reliable delivery of their respective SOAP messages with guaranteed delivery assurances (e.g., without message loss and/or duplication). Consider for instance two messaging activities such as *reply* or one-way *invoke* that target different partners and are nested in a *sequence*. The corresponding messages should be delivered in the order, in which the activities appear in the *sequence*, i.e, the SOAP message of the first activity should be received by the respective partner before the SOAP message of the second activity. Current reliable messaging specifications [17, 4] do not support this complex requirement because it involves more than two end points (the process and the two partners). This requirement is about the ordered message delivery in multi-party BPEL interactions [10].

2.2 Security

Messaging activities have simple requirements such as integrity, confidentiality, and authentication, which are supported by WS-Security [18]. Complex requirements arise when we consider issues such as secure conversations, trust, and federation in the context of BPEL processes. For example, we assume that we have a *sequence*, which contains many messaging activities targeting the same partner. From a performance point of view, it is inefficient to secure each messaging activity individually as in WS-Security. Instead, using a security context (according to WS-SecureConversation [14]) for all interactions with that partner would improve the performance significantly.

2.3 Transaction

Composite BPEL activities might require transaction semantics e.g., in a *sequence* with two *invoke* activities it might be necessary that either both invocations succeed or both must be undone. It is also essential that the process and the partners decide together the outcome of the transaction, which is called *external coordination* [19]. This feature is not supported in BPEL. Moreover, a way is needed to flexibly configure the transactional requirements of activities according to the application semantics e.g., a *sequence* might need to be executed as an *atomic transaction* or as a *compensation-based transaction*. WS-* transaction specifications [12, 13] can support the transactional requirements of BPEL activities if the BPEL engine is integrated properly with the Web Service transaction middleware.

2.4 Discussion

The problem of middleware requirements in BPEL is two-fold. On the one hand, appropriate means are needed to specify the requirements of BPEL activities. On the other hand, appropriate infrastructure is necessary to enforce those requirements during process execution. Even for simple requirements, most current BPEL engine do not provide a way to say "this invoke or that reply must be secure". Moreover, the integration of BPEL engines with existing WS-* middleware is still problematic.

3 The AO4BPEL Engine and the Container Framework

Our framework has three main components: a deployment descriptor, a process container, and a set of middleware Web Services [7].

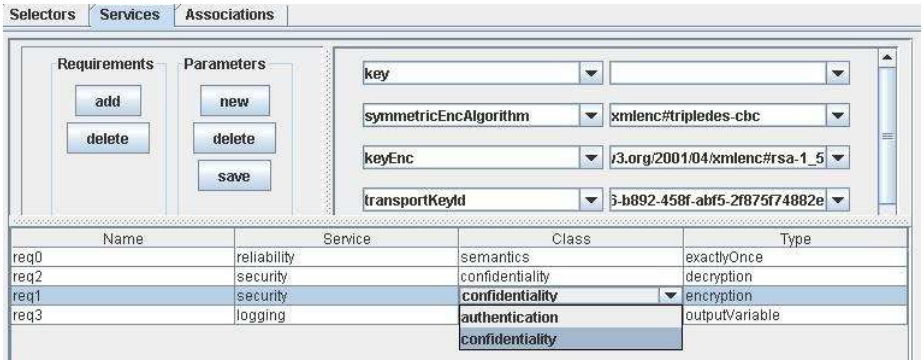


Fig. 1. Deployment Descriptor GUI Tool

3.1 The Deployment Descriptor

An XML-based deployment descriptor specifies declaratively the middleware requirements of the BPEL activities and the parameters that are needed to accomplish those requirements. Listing 1. shows an excerpt of a deployment descriptor for a bank transfer process, which calls the operations *credit* and *debit* on two partner Web Services.

The deployment descriptor defines one or more activity *selectors*, which are XPath expressions to identify the activities that will be associated with some requirement. The attribute *selectorid* attribute of the *requirement* element is used for this association. The *service* elements group requirements that belong to a specific middleware service.

```
<bpel-dd xmlns="http://www.st.informatik.tu-darmstadt.de/bpel-dd">
  <selectors>
    <selector id="0" name="credit" type="activity">/process//invoke[@operation="credit"]</selector>
    <selector id="1" name="debit" type="activity">/process//invoke[@operation="debit"]</selector>
  </selectors>
  <services>
    <service name="reliability">...
      <requirement name="req0" class="semantics" type="exactlyOnce" selectorid="0"/>...
    </service>
    <service name="security">...
      <requirement name="req2" class="confidentiality" type="decryption" selectorid="1"/>
       <parameters>
        <parameter name="symmetricEncAlgorithm">xmlenc#tripledes-cbc</parameter>
        <parameter name="keyEnc">http://www.w3.org/2001/04/xmlenc#rsa-1_5</parameter>
        <parameter name="transportKeyId">16c73ab6-b892-458f-abf5-2f875f74882e</parameter>...
       </parameters>
       </requirement>...
    </service>
  </services>
</bpel-dd>
```

Listing 1. The deployment descriptor

The deployment descriptor is the only component of the framework that the BPEL programmer needs to know about. He/she could write it manually or use the GUI tool shown in Fig. 1 to generate it.

One major advantage of the deployment descriptor against policies [16] is that the deployment descriptor allows the specification of the necessary parameters to enforce a given requirement. With policies, which are too declarative, this is not possible. For instance, in a policy is not possible to specify the user name and password that should be used to support authentication.

At process deployment time, the deployment descriptor file has to be specified in addition to the BPEL file as shown in Figure 2.

3.2 The Process Container

The process container is an implementation concept that BPEL programmers do not need to know about. It intercepts the execution of BPEL activities at well-defined points and calls dedicated middleware Web Services that provide the necessary functionality to enforce the middleware requirements.

We implemented a light-weight and aspect-based process container using a set of AO4BPEL aspects [6] that are automatically generated from the deployment descriptor. Automatic aspect generation is possible because the advices used

Fig. 2. Process Deployment in AO4BPEL

for integrating middleware Web Services follow well-defined patterns. E.g., for reliable messaging, there are recurring advice patterns for sending a message with exactly-once semantics, etc.

AO4BPEL [6, 9] is an aspect-oriented extension to BPEL. AO4BPEL supports *process-level join points* and *interpretation-level join points*. The former capture the execution of an activity. The latter capture internal points during the interpretation of an activity e.g., the point where a SOAP message of an *invoke* activity has been created. In AO4BPEL, an aspect defines one or more pointcuts and advices. A *pointcut* is a construct for selecting a set of join points e.g., to intercept some activities that have a common requirement. XPath is used as pointcut language in AO4BPEL. The *advice* is a BPEL activity that specifies some crosscutting functionality and can be used for example to enforce a requirement by calling a middleware Web Service.

After deploying the BPEL process (cf. Fig. 1), container aspects are automatically generated and deployed. We can see these aspects by switching to the process-list view of the AO4BPEL engine shown in Fig. 3.

Listing 2 shows the second aspect in the aspects list of Fig. 3. This aspect declares the reliable messaging service (RM) as partner and two variables for the input and output parameters of the call to *sendWithExactlyOnceSemantics*.

The pointcut of this aspect intercepts the *invoke* activity that calls the operation *credit* at the point where the SOAP request message has been created (this is the semantics of the advice type *around soapmessageout*). The advice contains an *assign* activity, which sets the input parameters of the call to the operation *sendWithExactlyOnceSemantics* of the RM Service. The SOAP message corresponding to the current join point (i.e., the *invoke* activity that calls the operation *credit*) is accessed by means of the special AO4BPEL context collection variable *soapmessage*. The second *assign* activity is used to pass the response message for the current join point activity from the RM Service to the BPEL process by using the special AO4BPEL context collection variable *newsoapmessage*. This is necessary because the request message for calling the operation *credit* was sent by the RM service on behalf of the BPEL process.

Fig. 3. The List of Container Aspects in AO4BPEL

```
<aspect name="credit_semantics_exactlyOnce">
  <partners><partner name="rmService" partnerLinkType="rms:RMService"/></partners>
  <variables>
    <variable messageType="rms:sendWithExactlyOnceSemanticsRequest" name="inputMessage"/>
    <variable messageType="rms:sendWithExactlyOnceSemanticsResponse" name="outputMessage"/>
  </variables>
  <pointcut name="creditExactlyOnce">/process//invoke[@operation="credit"]</pointcut>
  <advice type="around soapmessageout">
    <bpws:sequence>
      <bpws:assign>
        <bpws:copy><bpws:from part="message" variable="soapmessage"/>
                   <bpws:to part="message" variable="inputMessage"/></bpws:copy>
        <bpws:copy><bpws:from part="isInonly" variable="ThisJPActivity"/>
                   <bpws:to part="inonly" variable="inputMessage"/></bpws:copy>
        <bpws:copy><bpws:from part="partnerEndpoint" variable="ThisJPActivity"/>
                   <bpws:to part="endpoint" variable="inputMessage"/></bpws:copy>
      </bpws:assign>
      <bpws:invoke name="rmInvoke" operation="sendWithExactlyOnceSemantics" partner="rmService"
              inputVariable="inputMessage" outputVariable="outputMessage" portType="rms:RMService"/>
      <bpws:assign>
        <bpws:copy><bpws:from part="sendWithExactlyOnceSemanticsReturn" variable="outputMessage"/>
        <bpws:to part="newmessage" variable="newsoapmessage"/></bpws:copy>
      </bpws:assign>
    </bpws:sequence>
  </advice>
</aspect>
```

Listing 2. A container aspect for reliable messaging

3.3 The Middleware Web Services

The middleware Web Services are not part of the AO4BPEL engine. They are based on open source implementations of WS-* specifications.

The reliable messaging service [10] provides operations that are called by the container to enforce a delivery assurance for messaging activities (e.g., exactly-once) and to support the in-order delivery of messages even between more than two endpoints. Our implementation of this service is based on Apache Sandesha [1], which was extended to support mutli-party reliable messaging [10].

The security service [8] provides two port types: one for *secure messaging* according to WS-Security and one for *secure conversations* according to WS-SecureConversation with operations such as *createContext*, *encryptWithContext*, etc. The implementation of this service is based on Apache WSS4J [2].

The transaction service provides operations that are called by the container to enforce atomic transactions [12] such as *begin(transid)*, *participate(transid, soap)*, and *commit(transid)*. The implementation of this service is based on Apache Kandula [3], an implementation of WS-Coordination and WS-AtomicTransaction.

4 The Demo

In this demo, we will use a travel agency scenario with several BPEL processes that compose the Web Services of airline companies and hotel chains. We will deploy these processes on the AO4BPEL engine and specify a deployment descriptor file that defines the middleware requirements of the process activities.

Once the process is deployed, the audience will see how AO4BPEL container aspects will be generated automatically and activated. Then, we will start some instances of the deployed processes and use a tool to monitor the SOAP messages that are exchanged between the processes and their partners. Thus, we verify that the middleware Web Services are called correctly by the process container and the requirements of the different activities are fulfilled.

5 Conclusion

In this paper, we presented a user-friendly implementation of a container framework for the specification and enforcement of the middleware requirements of BPEL processes. The framework was inspired from enterprise component models and it can be reused with other BPEL engines. The engine has to provide a process container that is able to intercept the execution of BPEL activities and call the middleware Web Services. The deployment descriptor and the middleware Web Services could be reused without any changes.

The AO4BPEL engine presented here provides support for many BPEL-specific middleware requirements and paves the way toward a new breed of BPEL engines that we devise *application servers for BPEL*.

References

1. Apache. Sandehsa 1.0, July 2005.
2. Apache. Wss4j, March 2005.
3. Apache. Kandula 0.2, May 2006.
4. C. Ferris and D. Langworthy (Eds.). Web Services Reliable Messaging Protocol (WS-ReliableMessaging), February 2005.
5. C. Sharp (Eds.). Web Services Policy Attachment (WS-PolicyAttachment), September 2004.
6. Anis Charfi and Mira Mezini. Aspect-Oriented Web Service Composition with AO4BPEL. In *Proceedings of the European Conference on Web Services (ECOWS)*, volume 3250 of *LNCS*, pages 168–182. Springer, September 2004.
7. Anis Charfi and Mira Mezini. An Aspect-based Process Container for BPEL. In *Proceedings of the 1st Workshop on Aspect-Oriented Middleware Development (AOMD)*, November 2005.

8. Anis Charfi and Mira Mezini. Using Aspects for Security Engineering of Web Service Compositions. In *Proceedings of the IEEE International Conference on Web Services (ICWS), Volume I*, pages 59–66. IEEE Computer Society, July 2005.
9. Anis Charfi and Mira Mezini. AO4BPEL: An Aspect-Oriented Extension to BPEL. *World Wide Web Journal: Recent Advances on Web Services (special issue), to appear*, 2006.
10. Anis Charfi, Benjamin Schmeling, and Mira Mezini. Reliable messaging in bpel processes. In *Proceedings of the 3rd IEEE International Conference on Web Services (ICWS), to appear*, September 2006.
11. F. Curbera, Y. Goland, J. Klein, et al. Business Process Execution Language for Web Services (BPEL4WS) Version 1.1, May 2003.
12. D. Langworthy (Eds.). Web Services Atomic Transaction (WS-AtomicTransaction), November 2004.
13. D. Langworthy (Eds.). Web Services Business Activity (WS-BusinessActivity), November 2004.
14. M. Gudgin and A. Nadalin (Eds.). Web Service Secure Conversation Language (WS-SecureConversation) Version 1.0, February 2005.
15. IBM. The BPEL4WS Java Run Time, August 2002.
16. J. Schlimmer (Eds.). Web Services Policy Framework (WS-Policy)., September 2004.
17. OASIS. Web Services Reliable Messaging TC WS-Reliability 1.1, 15 November 2004.
18. OASIS. Web Services Security: SOAP Message Security Version 1.0, March 2004.
19. Stefan Tai, Rania Khalaf, and Thomas Mikalsen. Composition of coordinated web services. In *Proceeding of ACM/IFIP/USENIX International Middleware Conference (Middleware)*, volume 3231 of *LNCS*, pages 294–310. Springer, October 2004.