The Frame Problem in Web Service Specifications

George Baryannis and Dimitris Plexousakis
Department of Computer Science
University of Crete
GR 71409 Heraklion, Greece
{gmparg, dp}@csd.uoc.gr

Abstract

This work explores the frame problem and its effects in devising Web service specifications. The frame problem encompasses the issues raised when trying to concisely state in a specification that nothing changes except when explicitly mentioned otherwise. A motivating example of a composite service specification is presented and a solution approach is proposed, based on knowledge gained from related research on the frame problem in procedure specifications. Finally, an algorithm that applies the presented solution in order to transform existing OWL-S service descriptions to ones that are free from the frame problem is presented.

1. Introduction

Web services and service-oriented architecture (SOA) in general have emerged in recent years as a major technology for deploying automated interactions between distributed and heterogeneous applications and have motivated a great deal of research on many topics such as service foundations, composition, management and monitoring as well as service design and development [1]. An inherent issue in all these topics is how to devise a formal specification of a Web service.

Formal specifications allow for a precise description of what a Web service is supposed to do, e.g. in terms of its inputs, outputs, preconditions and effects. A complete formal Web service specification should allow us to thoroughly know and, in most cases, predict the service behavior under any circumstance, effectively answering in the best possible way a simple question: "What does this service do?". This answer should not be limited to an interface description, as provided by current standards such as WSDL and should not have to deal with the service's inner workings, e.g. its source code, which a service consumer most probably has no access to and no understanding of.

Preparing formal specifications, however, comes with a great deal of issues that need to be solved. One particular family of problems that emerge in formal specifications expressed in the form of preconditions and postconditions is referred to in the field of Artificial Intelligence as the frame problem [2]. The frame problem stems from the fact that including clauses that state only what is changed when preparing formal specifications is inadequate. Instead, one should also include clauses, called frame axioms, that explicitly state that apart from the changes declared in the rest of the specification, nothing else changes. Solving the frame problem essentially means finding a way to state frame axioms concisely without resulting in extremely lengthy, complex, possibly inconsistent, obscure specifications and at the same time retaining the ability of proving formal properties of the specifications.

While the frame problem has been thoroughly described and addressed in the AI literature, it has not been adequately examined with regard to its existence in Web service specifications and the effects it has in service-oriented architecture in general. The precondition and postcondition notation is universally used in functional descriptions of Semantic Web services, e.g. in the Service Profile class of OWL-S or the Web service capability subcomponent of WSMO. Thus, one should expect that the frame problem will also be encountered in formal Semantic Web service descriptions.

The rest of this document is organized as follows. Section 2 presents a motivating example in which the frame problem is encountered when devising a formal specification for a composite service. Section 3 presents a solution to the frame problem in Web service specifications inspired by an existing solution

978-1-4244-3716-0/09/$25.00 © 2009 IEEE

in the AI literature. Section 4 offers a brief description of work related to the frame problem in the area of Web services and Section 5 concludes.

2. A Motivating Example

In this section, we present a simple yet indicative example of the issues caused by the frame problem when devising a complete specification for a composite Web service.

Let's consider the case of an online shop that provides wish list and recommendations functionalities. When an order is completed, the items that were purchased must be removed from the wish list of the buyer and, for each purchased item, the most closely associated one must be chosen and included in the recommendations list of the particular user. The preconditions and postconditions for two services performing these actions, written in First-Order Predicate Logic, are shown in Figures 1 and 2.

PRE: completed(order, item) ∧
included(buyersWishList, item)
POST: ¬included(buyersWishList, item)

Figure 1. Wish list service

PRE: completed(order, item) ∧
¬included(buyersRecoms, associatedItem)
POST: included(buyersRecoms, associatedItem)

Figure 2. Recommendations list service

PRE: completed(order, item) ∧
included(buyersWishList, item)
POST: ¬included(buyersWishList, item) ∧
∀x, y [x ≠ buyersWishList ∨ y ≠ item ⇒
included(x, y) = included′(x, y)] ∧
∀x, y [completed(x, y) ≡ completed′(x, y)]
PRE: completed(order, item) ∧
¬included(buyersRecoms, associatedItem)
POST: included(buyersRecoms, associatedItem) ∧
∀x, y [x ≠ buyersRecoms ∨ y ≠ associatedItem ⇒
included(x, y) = included′(x, y)] ∧
∀x, y [completed(x, y) ≡ completed′(x, y)]

Figure 3. Service specifications with frame axioms

The specifications presented above cannot be considered complete, as no care has been taken concerning the frame axioms. We need to state for what argument values each of the predicates remain unchanged. The specifications with the addition of the frame axioms are shown in Figure 3. Note that primed predicates denote that they are evaluated in the final state, while unprimed predicates are evaluated in the initial state.

In our example, after an order is completed, we need a service that updates the wish list and recommendations list. Attempting to compose the two services according to the composition rules defined in [3] for a parallel composition, results in the specification shown in Figure 4. However, this specification is inconsistent, since contradicting statements are made for the circumstances under which the predicate included remains unchanged. This is a direct result of including frame axioms in our specification since we are forced to explicitly state what does not change in each separate specification, which will eventually lead to inconsistencies when we attempt to conjoin Web services that use the same predicates in their specifications.

In general, the frame problem is bound to appear and possibly cause inconsistencies in composite service specifications due to two main reasons. First of all, conjunctions are heavily used in almost all composition schemas, whether it is sequential, parallel, iterational, or conditional composition and so on, as it has been examined in [3]. It is also worth noting that attempting to introduce inheritance in service specifications will also lead to the frame problem. Second, it is highly possible that services being composed together will deal with the same knowledge, hence having specifications containing some common predicates, associated however with contradicting frame axioms.

PRE: completed(order, item) ∧
included(buyersWishList, item) ∧
¬included(buyersRecoms, associatedItem)
POST: ¬included(buyersWishList, item) ∧
included(buyersRecoms, associatedItem) ∧
∀x, y [x ≠ buyersWishList ∨ y ≠ item ⇒
included(x, y) = included′(x, y)] ∧
∀x, y [x ≠ buyersRecoms ∨ y ≠ associatedItem ⇒
included(x, y) = included′(x, y)] ∧
∀x, y [completed(x, y) ≡ completed′(x, y)]

Figure 4. Composite service specification

3. Addressing the Frame Problem

The frame axioms, as expressed in the motivating example, offer a procedure-oriented perspective to the frame problem, explicitly asserting what predicates each procedure does not change in addition to those it changes. In [4], the authors identified this fact as the source of the frame problem and aimed to replace the procedure-oriented with a state-oriented one, which we will explore in this section.

Instead of declaring what predicates don't change in each Web service specification, we can reverse our viewpoint and declare, for each element of the service specifications we are creating, which services may result in changing them. Thus, we don't aim to write a set of frame axioms for each Web service specification, but we create assertions, called explanation closure axioms or change axioms in [4], that explain the circumstances under which each predicate or function might be modified from one state to another.

3.1. Expressing change axioms

To be able to express the change axioms, a simple extension to the first-order predicate logic is proposed, that adds a special predicate symbol, named *Occur* and a special variable symbol named *α*. Variable α is used to refer to services taking part in the specification. Occur(α) is a predicate of arity 1 that is true if and only if the service denoted by the variable α has executed successfully. It is possible to negate Occur using ¬ in order to express the opposite semantics.

PRE: completed(order, item) ∧
included(buyersWishList, item) ∧
¬included (buyersRecoms, associatedItem)
CHANGE: ∀α ∀order, item
(completed(order, item) ∧
¬completed′(order, item) ∧ Occur(α)) ⇒ false
∀α ∀ x, item
(included (x, item) ∧
¬ included′(x, item) ∧ Occur(α))
⇒ α = updWishList ∨ α = updRecList

Figure 5. Composite service specification with change axioms

The specification for the composite service of the example, including change axioms is shown in Figure 5. It essentially states that, for the service to begin execution, the known preconditions must be met and any change to the predicate *included* signifies that a successful execution of one of the two atomic services of the example has taken place.

3.2. Change axioms in OWL-S

The solution proposed above is, at its basis, a reformulation of existing first-order logic specifications to ones that use the special predicate Occur. Most Semantic Web service specification frameworks support languages that include first-order logic notations. To express first-order logic formulas in OWL-S, an extension to SWRL, called SWRL-FOL has been proposed. In SWRL-FOL, Occur can be expressed as a unary predicate while the variable α can be expressed as an individual variable. Figure 6 contains a set of SWRL-FOL rules that express the change axioms included in the composite service specification that was presented in this section. The rules are written in the abstract syntax of SWRL-FOL and it's straightforward to transform them to an XML concrete syntax. Similar rules can be expressed in other Semantic Web service specification frameworks such as WSMO and SWSO.

forall (I-variable(a) I-variable(o1) I-variable(i1)
implies
(*Antecendent* (completed(I-variable(o1) I-variable(i1))
and neg (completedpr((I-variable(o1) I-variable(i1))
and Occur(I-variable(a)) *Consequent*())
forall (I-variable(a) I-variable(x1) I-variable(i1)
implies
(*Antecendent* (included(I-variable(x1) I-variable(i1))
and neg (includedpr((I-variable(x1) I-variable(i1))
and Occur(I-variable(a))
Consequent(I-variable(a) = updWishList or
I-variable(a) = updRecList)

Figure 6. Change axioms as SWRL-FOL rules

3.3. An algorithm for producing change axioms

Having defined our proposed solution for the frame problem, we turn our focus on sketching an algorithm for automatically producing change axioms, given a service description using the precondition/ postcondition notation. A complete set of change axioms should contain one axiom for every predicate contained in all the postconditions stated in the description, which may contain more than one

participating services. For each one of these predicates, we either add a change axiom or modify an existing one, depending on whether the predicate remains unchanged. The above are encoded in the algorithm in Figure 7.

```
For each participating service {
  For each predicate {
    If(corresponding frame axiom exists){
      If(predicate remains unchanged)
        do nothing
      else
        add a=<service-name> in change axiom }
    else {
      If(predicate remains unchanged)
        add change with false as consequent
      else
        add c.a. with a=<service-name> as cons. }
}}
```

Figure 7. Algorithm for composite service descriptions

If we consider a composition with m participating services, with each service having n distinct predicates, and assuming that it costs no more than O(logn) to check if a change axiom already exists, the complexity of the algorithm can be equal to O(m n logn). This remains to be confirmed with suitable testing of an implementation of the algorithm.

5. Related Work

The frame problem with regard to its effects in Web service specifications has only been addressed in few publications. In [5], a mapping from OWL-S to the situation calculus is proposed, allowing the authors to use the solution proposed by Reiter [6] for the frame problem in the situation calculus. Successor state axioms are also used in [7], where the authors extend Golog to support generic programs and be more suitable for service description and automatic Web service composition. This, along with similar efforts, inspired by the solution proposed by Reiter, have the disadvantage that they use logic formalisms that are not supported by any current Semantic Web service frameworks, in direct contrast to first-order predicate logic which is universally supported. Also, Golog may not be suitable for Web services in some cases, as it is not possible to present and reason about multiple copies of literals in world states.

6. Conclusions

In this work, we explored the frame problem with regard to its existence and possible solution in the field of Web services. We argued that Web service specifications, become problematic when trying to explicitly express that nothing changes, except when it is stated. We presented a solution approach that includes axioms that state precisely which service execution leads to what predicate changes.

An implementation of the algorithm presented is currently at an early stage. It is a matter of future work to complete the implementation and assure that it covers all composition schemas and any special issues that may be directly associated to a specific schema.

Acknowledgement

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

References

[1] M.P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, "Service-Oriented Computing: State of the Art and Research Challenges", *Computer*, IEEE Computer Society, November 2007, pp. 38-45.

[2] J. McCarthy, and P. Hayes, "Some Philosophical Problems from the Standpoint of Artificial Intelligence", *Machine Intelligence*, Edinburgh Univ. Press, Vol. 4, 1969, pp. 463-502.

[3] V. Alevizou, and D. Plexousakis, "Enhanced Specifications for Web Service Composition", *Proceedings of the 4th IEEE European Conference on Web Services*, Zurich, Switzerland, 2006, pp. 223-232.

[4] A. Borgida, J. Mylopoulos, and R. Reiter, "On the Frame Problem in Procedure Specifications", *IEEE Transactions on Software Engineering*, IEEE, Vol. 21, No. 10, pp. 785-798.

[5] S. Narayanan, and S.A. McIlraith, "Simulation, Verification and Automated Composition of Web Services", *Proceedings of the 11th International Conference on World Wide Web*, Honolulu, Hawaii, USA, 2002, pp. 77-88.

[6] R. Reiter, "The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal Regression", *Artificial Intelligence and the Mathematical Theory of Computation: Papers in Honor of John McCarthy*, Academic Press, San Diego, California, 1991, pp. 359-380.

[7] S.A. McIlraith, and T.C. Son "Adapting Golog for Composition of Semantic Web Services", *Proceedings of the International Conference on the Principles of Knowledge Representation and Reasoning (KRR'02)*, 2002, pp. 291-302.

A.2 A Model-Driven Approach to Implementing Coordination Protocols in BPEL

Authors:

USTUTT: Oliver Kopp

USTUTT: Branimir Wetzstein

USTUTT: Ralph Mietzner

USTUTT: Stefan Pottinger

USTUTT: Dimka Karastoyanova

USTUTT: Frank Leymann

- Submitted to: MDE4BPM 2008

Institute of Architecture of Application Systems

A Model-Driven Approach to Implementing Coordination Protocols in BPEL

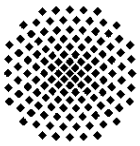
Oliver Kopp[1], Branimir Wetzstein[1], Ralph Mietzner[1], Stefan Pottinger[2], Dimka Karastoyanova[1], Frank Leymann[1]

[1]Institute of Architecture of Application Systems, University of Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

[2]IPL Information Processing Ltd, United Kingdom
Stefan.pottinger@ipl.com

BibTeX:

```
@inproceedings{CPG,
  author    = {Oliver Kopp and others},
  title     = {A Model-Driven Approach to
               Implementing Coordination Protocols in {BPEL}},
  booktitle = {MDE4BPM},
  year      = {2008},
  pages     = {188-199},
  doi       = {10.1007/978-3-642-00328-8_19},
  publisher = {Springer}
}
```

Universität Stuttgart
Germany

A Model-Driven Approach to Implementing Coordination Protocols in BPEL

Oliver Kopp[1], Branimir Wetzstein[1], Ralph Mietzner[1], Stefan Pottinger[2], Dimka Karastoyanova[1], and Frank Leymann[1]

[1] Institute of Architecture of Application Systems, University of Stuttgart, Germany
[2] IPL Information Processing Ltd, United Kingdom

Abstract. WS-Coordination defines a framework for establishing protocols for coordinating the outcome agreement within distributed applications. The framework is extensible and allows support for multiple coordination protocols. To facilitate the realization of new coordination protocols we present a model-driven approach for the generation of BPEL processes used as implementation of coordination protocols. We show how coordination protocols can be modeled in domain-specific graph-based diagrams and how to transform such graphs into abstract BPEL process models representing the behavior of the coordinator and the participants in the protocol.

1 Introduction

Web services are the most recent middleware technology for application integration within and across enterprises [1]. Through the use of standards like SOAP, WSDL, UDDI, and the Web Services Business Process Execution Language (BPEL, [2]) the Web service technology enables interoperable service interactions in heterogeneous environments. Coordination is an important mechanism used in distributed computations with multiple participants that must jointly agree on the outcome of the computation. A well-known example for the use of coordination are distributed transactions using atomic commitment protocols to agree on the success or failure of a transaction [3]. The aspect of coordination in the domain of Web services is addressed by WS-Coordination [4]: it defines an extensible framework for coordinating the outcome of a set of Web services contributing to a distributed computation using a generalized notion of a coordinator and the so-called coordination protocols. In the context of WS-Coordination, coordination protocols describe the messages exchanged between the coordinator and the participants of a distributed computation and thus realize a one-to-many coordination. Two types of protocols (aka coordination types) have already been defined to cover "traditional" atomic transactions (WS-AtomicTransaction [5]) and long-running business transactions (WS-BusinessActivity [6]). However, the use of WS-Coordination is not restricted to transaction processing systems only. Other types of coordination protocols have also been defined for distributed computations such as protocols describing auctions [7], protocols for split BPEL

loops and split BPEL scopes [8] and protocols for externalizing the coordination of BPEL scopes as a whole [9].

Coordination protocols can be quite complex. The coordinator has to deal with a variable number of participants. Each participant is in a well-defined state that potentially differs from the state of another participant at the same time. The implementation of a coordination protocol is difficult and error-prone. To simplify and accelerate the implementation, and eliminate errors, in this paper we propose a model-driven architecture (MDA) approach: The protocol is first modeled as a state-based graph, which we call coordination protocol graph (CPG). A CPG captures the different states and state changes based on the messages exchanged between coordinator and participant. The graph diagram is the domain specific language (DSL) we use for specifying coordination protocols. It contains only those elements which are needed for coordination protocol modeling and is therefore well suited for protocol designers. In MDA terms a coordination protocol graph specifies a Platform Independent Model (PIM) [10]. The CPG is independent of any hardware or programming platform.

We have decided to represent the Platform Specific Model (PSM) in terms of BPEL since, in general, coordination protocols define a sequence of steps and messages to be exchanged between participants in a coordinated interaction, timing issues, and how exceptional situations must be tackled. In that respect, modeling coordination protocols is similar to modeling business processes. In this work we generate abstract BPEL processes for both the coordinator and the participant roles in coordination protocols. These BPEL process models capture the essential parts of the message exchange between the parties and the resulting protocol state changes. The generated code reduces the need for tedious and error-prone programming concerning the communication between the coordinator and participants in the protocol. Additional protocol logic, which cannot be captured in the CPG, has to be manually added by the programmer.

The rest of the paper is organized as follows: Section 2 gives an overview of BPEL and WS-Coordination. In Section 3 we present the syntax and semantics of the coordination protocol graph (CPG). After depicting our model-driven approach in Section 4, we describe the generation of the BPEL process models in Section 5. We finalize with the discussion of related work, conclusion and future work.

2 Background

WS-Coordination [4] defines an extensible framework for coordinating interactions between Web services. Coordinated interactions are called (coordinated) activities in the context of WS-Coordination. The framework enables participants to reach agreement on the outcome of distributed activities using a coordinator and an extensible set of coordination protocols. The framework defines three services a coordinator has to provide: activation service, registration service, and protocol services. When an application, in the role of an initiator, wants to start a coordinated activity, it requests a coordination context from

an activation service. The coordination context contains an activity identifier, the coordination type (e.g. atomic transaction) as requested by the initiator, and the endpoint reference of the registration service. When the initiator distributes work, it sends the coordination context with the application message to the participant. Before starting work, the participant registers at the registration service of the coordinator. At some later point the protocol service, which coordinates the outcome according to the specific protocol of the coordination type, is started.

While the logic of the activation and registration service are fixed, the framework allows the definition of arbitrary coordination types as well as their implemenation by means of different protocol services. In the following when referring to "coordinator" and "participant", we mean the protocol service implementations at the coordinator and participant, respectively.

The *Web Services Business Process Execution Language* (BPEL) is an orchestration language for Web services. A BPEL process is a composition of Web services, which are accessed through partner links referencing their WSDL port types. The process is itself exposed as a Web service.

The BPEL process model comprises two types of activities: basic activities cope with invoking other Web services (invoke), providing operations to other Web services (e.g. receive and reply), timing issues and fault handling; structured activities nest other activities and deal with parallel (flow) and sequential execution (sequence), conditional behavior and event processing. Process data is stored in variables, while the assign activity is used for data manipulation. Activities can be enclosed in scopes to denote sets of activities that are to be dealt with as a unit of work. Scopes can be modeled to ensure all-or-nothing behavior, support data scoping, exception handling, compensation, and sophisticated event handling. Instance management is done using correlation sets. Correlation sets define which fields in incoming messages are to be used as identifiers to route the messages to one of possibly several running instances of the same process model.

BPEL processes can be either abstract or executable. An executable BPEL process provides a process model definition with enough information to be interpreted by a BPEL process engine. An abstract BPEL process hides some of the information needed for execution and is associated with a process profile defining restrictions and the indented usage of the abstract process. The profile used in our approach is the abstract process profile for templates. It allows marking sections of the process model as "opaque" using opaque tokens. It is thus explicitly specified which sections of the process model have to be later replaced by concrete activities, expressions etc. to make the process executable.

3 Modeling Coordination Protocols

There is no standard notation for modeling coordination protocols. The specifications in this area use either a proprietary or a generic diagram type (e.g. UML sequence diagram), or a combination of these. For modeling coordination

protocols we have adopted the diagram type from the WS-AtomicTransaction (WS-AT) and WS-BusinessActivity (WS-BA) specifications. This diagram type can be seen as a domain specific language for modeling coordination protocols. WS-BA contains two protocols: *WS-BA with Participant Completion*, where the participant signals when it has completed its work and *WS-BA with Coordinator Completion*, where the coordinator notifies the participant when it has to complete his work. Figure 1 shows the WS-BA with Participant Completion protocol as an example, which we will also use in the rest of the paper for illustration of mapping concepts.

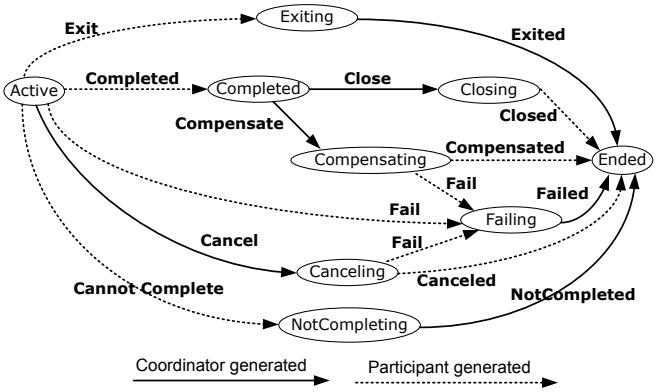


Fig. 1. WS-BA with Participant Completion Protocol [6]

The diagram defines a state-based graph, which we name coordination protocol graph (CPG). A CPG is a directed graph with labeled edges and labeled nodes. The nodes denote the states of the coordination protocol between a coordinator and a participant. The node labels describe the semantics of the states. The edges depict the messages exchanged by the protocol parties; the edge labels describe the semantics of the message. Since messages can be sent by a participant and by a coordinator, the set of all edges is divided into two disjoint sets: edges denoting coordinator messages (solid lines) and edges denoting participant messages (dashed lines). Each CPG has exactly one node with no incoming edges (source) and at least one node without outgoing edges (sink). No two coordinator edges or participant edges with the same label may leave the same node, because this would lead to non-determinism. A CPG does not contain cycles. The conclusion section includes a discussion about cycles in a CPG and the possibilities to support cyclic CPGs. At a certain point in time each participant can be in a different state. For example, one participant can be in the state "Failing" while another is in the state "Closing". Since coordinator usually interacts with more than one participant, the coordinator has to hold the state of each state machine.

Outgoing edges of a CPG denote messages which may be sent and each state denotes the possible state of a participant. The sender of the message (participant or coordinator) transitions to the next state when sending a message. The recipient of the message transitions to the next state when receiving the message. For the period of time when the message is transported, the coordinator and participant thus are in different states. In addition to the obvious behavior of state changes there are three special cases: (i) ignoring same messages which are sent more than once, (ii) precedence of participant messages over coordinator messages, (iii) invalid messages.

If the message leading to a new state is received more than once, it is simply ignored. For example, if the coordinator being in state "Exiting" receives the message "Exit" again, that message is ignored. This case can arise, when messages are resent because it is suspected that the first message hasn't been transmitted successfully.

If a state has both outgoing participant and coordinator messages, then it can happen that the coordinator sends a protocol message and enters the corresponding new state, but later receives a protocol message from the participant which is consistent with the former state. This can happen when both the coordinator and the participant send their messages at about the same time, which leads to different views on the protocol state on coordinator and participant side. In that case the participant messages have precedence over coordinator messages. In Figure 1 the state is "Active" at the beginning of the protocol. Let us assume the coordinator sends "Cancel" to the participant and sets the state to "Canceling". At the same time, however, the participant sends the message "Completed" and changes his state to "Completed". When the coordinator receives the message "Completed" while being in state "Canceling" for the participant, he has to revert to the former state "Active", accept the notification "Completed" and change the state to "Completed". The participant on the other side just discards the coordinator message "Cancel".

Finally, if in a state other messages than the allowed ones are received, a fault message should be generated and sent to the sender of the invalid message. The protocol execution is aborted.

It is important to note, that a CPG captures only the possible interactions and state changes between the coordinator and participant. A CPG does not capture the reason of these state changes. For example, if a participant is in the state "Completed" it can receive either a "Close" or a "Compensate" message from the coordinator. Which of the two messages is sent, is part of the protocol logic. For example, if another participant has failed and all-or-noting semantics is needed a "Compensate" message would be sent. Because not all of the protocol logic is captured by the graph, it has to be additionally implemented after the generation of the BPEL process.

The CPG and its semantics are derived from WS-BA and WS-AT protocols. In summary, the CPG graph captures the exchanged messages between a coordinator and a participant, and the resulting state changes, however not the cause of the state changes.

4 Model-Driven Implementation Approach

For the implementation of coordination protocols we adopt a model-driven approach. Our goal is to model the coordination protocol using a domain-specific language suitable for coordination protocol designers, and then generate BPEL code which implements the coordination protocol.

The DSL, in our case the CPG, is used for creating a platform-independent model (PIM) of the coordination protocol. The PIM can be transformed to platform-specific models (PSM) for different kind of platforms. In this paper we use BPEL and the Web service platform, in particular WS-Coordination as the coordination framework.

As the CPG does not contain enough information to be executed, the additional information has to be added to the PSM after generation. We thus do not achieve 100% BPEL code generation, but still avoid much of tedious and error-prone programming. The use of a model-driven approach ensures higher productivity in development and better quality of the implemented code.

Using BPEL for implementation of coordination protocols has several benefits when compared to a 3GL programming language such as Java. BPEL enables programming on a higher abstraction level which makes the code generation easier. BPEL has native WSDL support needed for interoperability and native support for concurrency, backward and forward recovery. As BPEL supports graph-based models, coordination protocol graphs can be more easily and naturally transformed into BPEL. A BPEL engine persistently stores all events related to process execution in an audit log and thus automatically supports reliable recording of coordination protocol execution out of the box. The audit log enables checking the execution of coordination for compliance with the protocol. A BPEL engine typically also provides a monitoring tool, which enables observing the execution of coordination protocols in real-time. Finally, as the state of a BPEL process instance is persistently saved after each state transition, the coordination protocol can be stopped and resumed at any time using a monitoring tool.

The approach is shown in Figure 2. In the first step, the CPG is created using a corresponding graphical CPG modeling tool. The CPG models the interaction between the coordinator and the participant in a platform-independent way.

In the next step, the CPG is transformed into two abstract BPEL processes, one for the coordinator and one for the participants. Therefore, the abstract BPEL processes and corresponding WSDL definitions are generated. If the WSDL definitions already exist, as for example in the case of the WS-AT and WS-BA specification, then the CPG has to be correspondingly marked. One has to specify the names of the WSDL port types for both the participant and coordinator process, the WSDL message and operation names, which correspond to the labels of the state transitions in the CPG, etc. That ensures that the generated BPEL processes and WSDL descriptions are compliant to the already existing probably standardized ones. The two corresponding WSDL interface descriptions of the processes can be completely generated. Using standardized WSDL interfaces ensures that the coordinator process can be used to

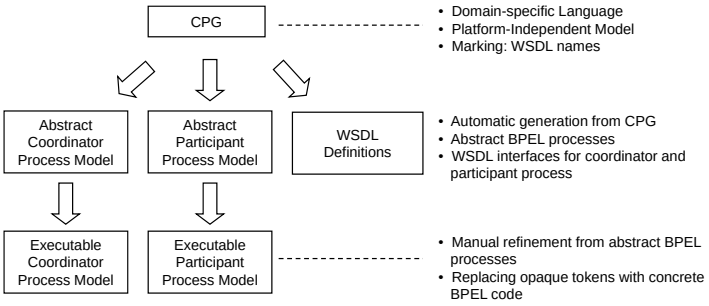


Fig. 2. Model-Driven Implementation Approach

coordinate arbitrary protocol participants apart from the generated BPEL-based participants. This is also the case for the generated participants, which can be used with another protocol-compliant coordinator. Thus, our approach supports heterogeneous environments.

As discussed in the previous section, the generated process models cannot be executable, because the CPG does not capture the whole protocol logic. The locations in the process model where missing logic has to be added are "marked" in the generated BPEL code using opaque tokens, as defined in the abstract process profile for templates [2]. These opaque tokens show to the developer where additional logic has to be added to make the process executable. The abstract BPEL process profile for observable behavior [2] cannot be used, since it does not allow the addition of interaction activities with existing partner links when replacing opaque activities. However, that is needed in certain cases: For example, in the coordinator process after the interaction activity receiving a "Fail" from one participant, one might want to add interaction activities (BPEL invoke) which send "Cancel" notifications to other participants.

As already described in Section 2, WS-Coordination defines three services a coordinator has to provide: activation, registration, and protocol services. While protocol services can be additionally defined in separate specifications such as WS-BA, the implementation of the activation and registration services stays the same. The activation and registration service of the coordinator can thus be fully generated. Both services in addition to the protocol service are implemented by the coordinator process model (see Section 5).

After generation, the abstract process models are refined manually by a developer who replaces the opaque tokens by the missing coordination protocol logic. The resulting executable BPEL process models can finally be deployed on a BPEL engine.

5 Generating BPEL Process Models

In the following we describe in detail how CPG graphs are transformed to abstract BPEL process models. We generate two abstract BPEL process models, one for the coordinator and one for the participants.

We have chosen different approaches for the generation of the two process models. For the participant process model, we keep the graphical structure of the CPG in the BPEL process model by mapping the CPG graph directly to a BPEL flow. The BPEL flow activity together with BPEL links enables graph-based workflow modeling. The generated BPEL process structure closely resembles the CPG structure and thus increases the readability of the process. The generated BPEL constructs are described in detail in [11].

For the coordinator process model the participant approach is not feasible, since the coordinator holds a different state for each participant. The coordinator cannot leave the scope "Active" until all registered participants have been handled for that scope. In the meantime, however, several participants could have declared that they want to exit the protocol by sending the message "Exit" to the coordinator. In that case the coordinator should immediately send the notification "Exited" to the participant. However, this is not possible, since the coordinator is in the scope "Active" and waits for other participants to complete their work. When the coordinator finally leaves the scope "Active", a new participant could register for the protocol. Since the scope "Active" has already finished, the new participant cannot be handled. Therefore, we define global event handlers for each message that can be received by the coordinator. That means, we implement a state-machine by specifying rules of the form: if received message x, then perform some logic which handles that message x.

Figure 3 illustrates the pattern for the implementation of the coordinator scope. An instance of the coordinator process model is started when a new WS-Coordination activity is created. This is done by an application by sending a "CreateCoordinationContext" message to the coordinator endpoint which replies with a "CreatedCoordinationContext" message to indicate successful creation of the context.

Having received such a message the coordinator process is now ready to accept registration messages from participants that wish to participate in the coordination, and to react on messages sent by participants that have already registered. The coordinator leaves this state if the application determines that the coordination should end and sends a corresponding message.

The abstract BPEL process template for the coordinator is generated as follows: In order to manage the participants for the activity an array is generated that holds the status of all participants of the activity as well as the endpoint references of the participants. The endpoint references are obtained during registration and are needed to send coordination messages to the right endpoints.

Regarding the control flow, at first a process instance creating receive activity is added that is triggered by WS-Coordination "CreateCoordinationContext" messages. The user can then replace the following opaque activity by inserting arbitrary BPEL activities that handle the message. Afterwards the confirmation

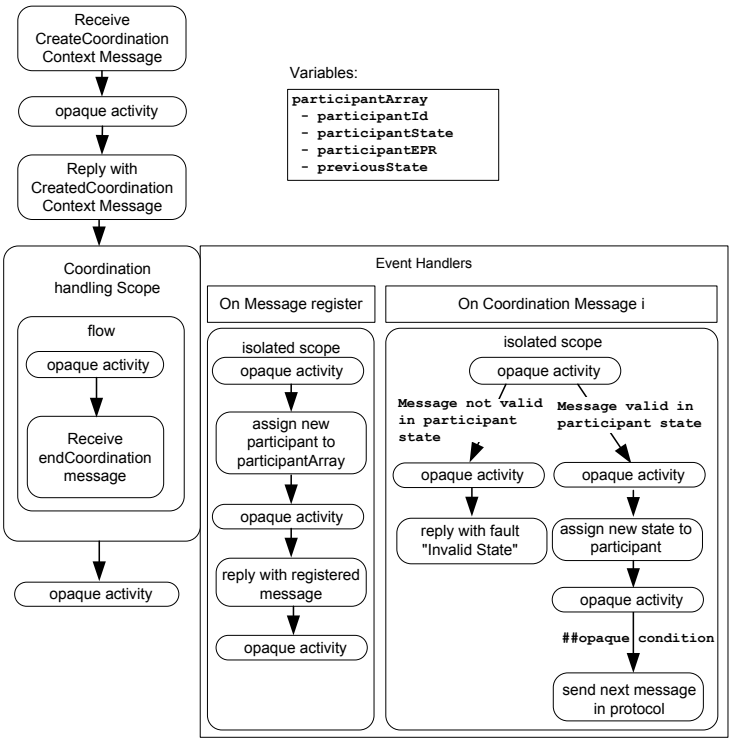


Fig. 3. Pattern for the generation of coordinator scopes

for the successful creation of the coordination context is sent. The control-flow now enters the scope that handles the coordination protocol specific messages as well as the registration of participants for the activity. Both types of messages are received and handled via event handlers.

We place opaque activities throughout the process template during generation to allow the actual coordination logic to be inserted as needed. We do not explicitly mention those in the following discussion, but Figure 3 shows where the opaque activities are placed in detail. In the following description we concentrate on the control flow and leave out details such as correlation of messages to the right process instance. For now, we assume that upon reception of each message the coordinator knows which participant has sent the message and that messages only are received by coordinator process instances that handle the participant that has sent the message. Means to ensure these assumptions are presented in [11].

Registration of Participants As shown in Figure 3 registration of participants is handled via a dedicated event handler. The event handler includes an assign activity that adds the new participant into the participant array and sets its current state to the first state that follows registration in the coordination protocol

the coordinator has been created for. Afterwards the event handler responds with a "Registered" message. Both "Register" and "Registered" messages are defined in WS-Coordination. Opaque activities allow the handling of special cases by special coordination logic. Such a case may be the reception of registration messages after other participants have already faulted or completed. For example, WS-BA demands that such cases are allowed.

Handling of Protocol Specific Messages For each participant generated message a separate event handler is created that handles that type of message. Upon receipt of a participant message, one out of two paths can be followed: The first path is followed if the message is not allowed in the state of the participant. In that case the "Invalid State" message is sent back. In case the message is allowed in the current state, the state of the participant is updated via an assign activity. The generated model contains opaque activities that can be replaced by arbitrary BPEL activities that perform the actual coordinator logic. For example, one or more invoke activities can be inserted that send the corresponding messages that follow the received message in the coordination protocol. The decision whether a and which message is sent depends on the actual coordinator logic. Thus the transition condition is marked as opaque and needs to be completed during the customization of the template.

The second path also handles two special cases: (i) ignoring messages which were resent by the participant, (ii) reverting to a previous state. Both the WS-AtomicTransaction and the WS-BusinessActivity specification demand that not only messages that are allowed in the current state of the participant are allowed but also messages corresponding to the previous state of the participant. In order to comply with this demand an additional field in the array is introduced that stores the previous state. On reception of a message a new assign activity is introduced that reverts the state of a participant if a message corresponding to that state is received. Then the control flow can proceed as if it had originally received the message in the correct state.

Concurrent Reception of Messages All messages that can be received concurrently by the coordinator are handled by event handlers. Thus, we ensure that the BPEL engine can deal with the concurrent message reception. However in order to ensure that concurrent access to shared variables, such as the participant arrays, and resulting problems are avoided the logic of the event handlers is placed in isolated scopes. An isolated scope is a BPEL means to synchronize parallel access to variables.

6 Related Work

There are several approaches to map business processes modeled graphically to BPEL (e.g. [12, 13]). The approaches are similar to our work, since they are also generating BPEL processes, but the authors deal with generating a single BPEL process: they focus on orchestrations only. Hence, these approaches do not tackle

the communication between processes as it is the case between the coordinator and the participant process.

In contrast to orchestrations, choreographies provide a global view on the interactions of all participants involved. In a coordination, the set of participants is unknown in advance. All choreography languages targeted to Web service technology either do not support modeling of a-priori unknown number of participants (WS-CDL [14, 15]) or do not support modeling the assignment of a participant to a different set (BPMN [16] and BPEL4Chor [17], Let's Dance [18]).

Another approach to model transactions is the UN/CEFACT's Modeling Methodology (UMM, [19]). While UMM can be mapped to BPEL [20], UMM does not support modeling of sets of a-priori unknown participants.

7 Conclusions and Future Work

The main contributions of this paper are: (i) the introduction of a model-driven approach for implementing coordination protocols, (ii) the concrete transformation of the CPG graph to abstract BPEL process models.

We have shown how a WS-Coordination-based coordination protocol can be modeled as a CPG graph. A CPG graph captures the essence of a coordination protocol: the states of the protocol and messages produced by both the coordinator and the participant. The generated BPEL processes are abstract and comply with the abstract process profile for templates. Opaque activities and expressions mark the locations where the programmer can include additional protocol logic not captured by the CPG to make the processes executable.

We demanded CPGs to be acyclic, since BPEL supports structured loops only. While this works for the protocols described in WS-AtomicTransaction and WS-BusinessActivity, there are coordination protocols such as the protocol for split loops [8]. We used an event handler approach for the coordinator to deal with the different states of each participant, which enables support for loops, too. For the participant model, we generated a BPEL process where the structure of the process directly reflects the structure of the CPG. Basically, when mapping CPGs with structured loops to BPEL, these loops can be captured using BPEL loop constructs. The current mapping style to participant processes does not support loops, since the BPEL flow activity only supports acyclic graphs. When mapping unstructured loops to a BPEL process there are two general approaches: (i) mirror the semantics using event handlers and (ii) untangle the loop by duplication of the activities [21]. The event handler approach is similar to the presented realization of the coordinator. However, the approach has the drawback that the control-flow is captured using event-action rules and not the "usual" BPEL constructs to model the main path of execution. The second approach uses the BPEL flow activity but duplicates the activities. This duplication can be avoided if sub-processes (as defined in BPEL-SPE [22]) are used: instead of mapping each activity directly, each original activity is mapped to a sub-process call. In addition, for each original activity, a separate sub-process is generated. The details of the transition conditions, data passing to and from the

sub-process are open issues. Our future work is to evaluate the two possibilities in depth and to realize the more suitable one.

Acknowledgments. The research leading to these results has received partial funding from the European Community's 7^{th} Framework Programme under the Network of Excellence S-Cube (Grant Agreement no. 215483) and the German Federal Ministry of Education and Research (Tools4BPEL, project number 01ISE08B).

References

1. Curbera, F., et al.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging and More. Prentice Hall PTR
2. OASIS: Web Services Business Process Execution Language Version 2.0
3. Gray, J., Reuter, A.: Transaction Processing: concepts and techniques. Morgan Kaufman
4. OASIS: Web Services Coordination. Version 1.1
5. OASIS: Web Services Atomic Transaction. Version 1.1
6. OASIS: Web Services Business Activity Framework. Version 1.1
7. Leymann, F., Pottinger, S.: Rethinking the Coordination Models of WS-Coordination and WS-CF. In: ECOWS 2005
8. Khalaf, R., Leymann, F.: Coordination Protocols for Split BPEL Loops and Scopes. Technical Report Computer Science 2007/01, University of Stuttgart
9. Pottinger, S., et al.: Coordinate BPEL Scopes and Processes by Extending the WS-Business Activity Framework. In: CoopIS 2007
10. Frankel, D.S.: Model Driven Architecture: Applying MDA to Enterprise Computing. Wiley (2003)
11. Kopp, O., et al.: A Model-Driven Approach to Implementing Coordination Protocols in BPEL. Technical Report 2008/02, University of Stuttgart
12. Mendling, J., Lassen, K.B., Zdun, U.: On the Transformation of Control Flow between Block-Oriented and Graph-Oriented Process Modeling Languages. IJBPIM **3**(2) (2008)
13. Ouyang, C., et al.: Translating Standard Process Models to BPEL. In: Advanced Information Systems Engineering. (2006)
14. Kavantzas, N., et al.: Web Services Choreography Description Language Version 1.0, W3C Candidate Recommendation
15. Decker, G., et al.: On the Suitability of WS-CDL for Choreography Modeling. In: EMISA 2006
16. Object Management Group: Business Process Modeling Notation, V1.1
17. Decker, G., et al.: BPEL4Chor: Extending BPEL for Modeling Choreographies. In: ICWS 2007
18. Zaha, J.M., et al.: A Language for Service Behavior Modeling. In: CoopIS 2006
19. UN/CEFACT: UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module. `http://www.unece.org/cefact/umm/UMM_Foundation_Module.pdf`.
20. Hofreiter, B., et al.: Deriving executable BPEL from UMM Business Transactions. In: SCC 2007
21. Zhao, W., et al.: Compiling business processes: untangling unstructured loops in irreducible flow graphs. International Journal of Web and Grid Services **2** (2006)
22. IBM, SAP: WS-BPEL Extension for Sub-processes – BPEL-SPE. (2005)

A.3 Modeling Context-aware and Socially-enriched Mashups

Authors:

TUW: Martin Treiber

POLIMI: Kyriakos Kritikos

TUW: Daniel Schall

UOC: Dimitris Plexousakis

TUW: Schahram Dustdar

- Submitted to: Mashups 09 @ OOPSLA

Modeling Context-aware and Socially-enriched Mashups

Martin Treiber[1], Kyriakos Kritikos[2], Daniel Schall[1], Dimitris Plexousakis[3], Schahram Dustdar[1]

[1]Vienna University of Technology, [2]Politecnico di Milano, [3]University of Crete

{m.treiber, schall, dustdar}@infosys.tuwien.ac.at, kritikos@elet.polimi.it, dp@csd.uoc.gr

Abstract

Mashup platforms and end-user centric composition tools have become increasingly popular. Most tools provide Web interfaces and visual programming languages to create compositions. Much of the previous work has not considered compositions comprising human provided services (HPS) and software-based services (SBS). We introduce a novel HPS aware service mashup model which we call socially oriented mashups (SOM). The inclusion of HPS in service mashups raises many challenges such as a QoS model that must account for human aspects and the need for flexible execution of mashups. We propose human quality attributes, for example delegation, and a context model capturing various information including location and time. The QoS and context model is used at design-time and for runtime adaptation of mashups. In this paper, we show how to model context-aware SOMs that include HPS and SBS and demonstrate the first results of our working prototype.

Categories and Subject Descriptors D.2.2 [*Software Engineering*]: Design Tools and Techniques; H.3.5 [*Online Information Services*]: Web-based Services

General Terms Mashups, Composition, Context, QoS

1. Introduction

The role of humans in service compositions and workflows has gained tremendous attention. With the proliferation of Web service mashups [21], human aspects became important for designers and the end-users as well. The ability of non experts to create mashup applications for their personal use is considered to increase the productivity of employees.

Currently there is little support for the exploitation of these dynamic aspects in workflows, where humans are integrated into service mashups. In order to tap resources for the creation of flexible and human oriented service compositions, a framework is required that offers the required flexibility and simplicity for the end-user. In particular, we propose the seamless integration of HPS into service mashups. In this context, we refer to these human augmented service mashups as *socially oriented mashups* (SOM) because of phenomena like (sub-)task delegation to other humans during the execution of the mashup. Furthermore, if we consider humans as part of service mashups, we add automatically reasoning capabilities to the mashup. Thus, we make a service mashup flexible and adaptive to unpredictable situations, where human expertise is needed to adapt to new situations.

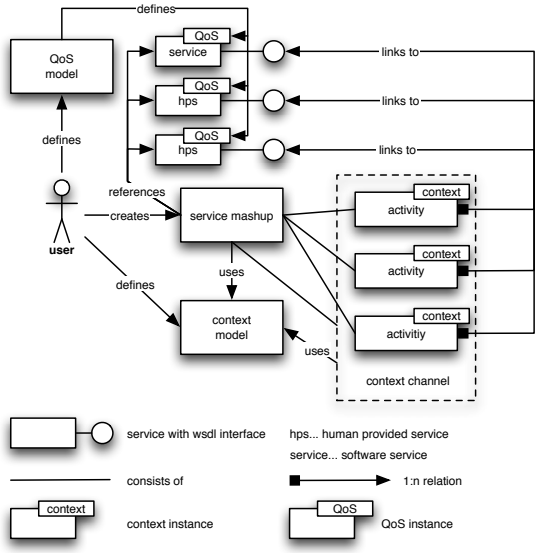
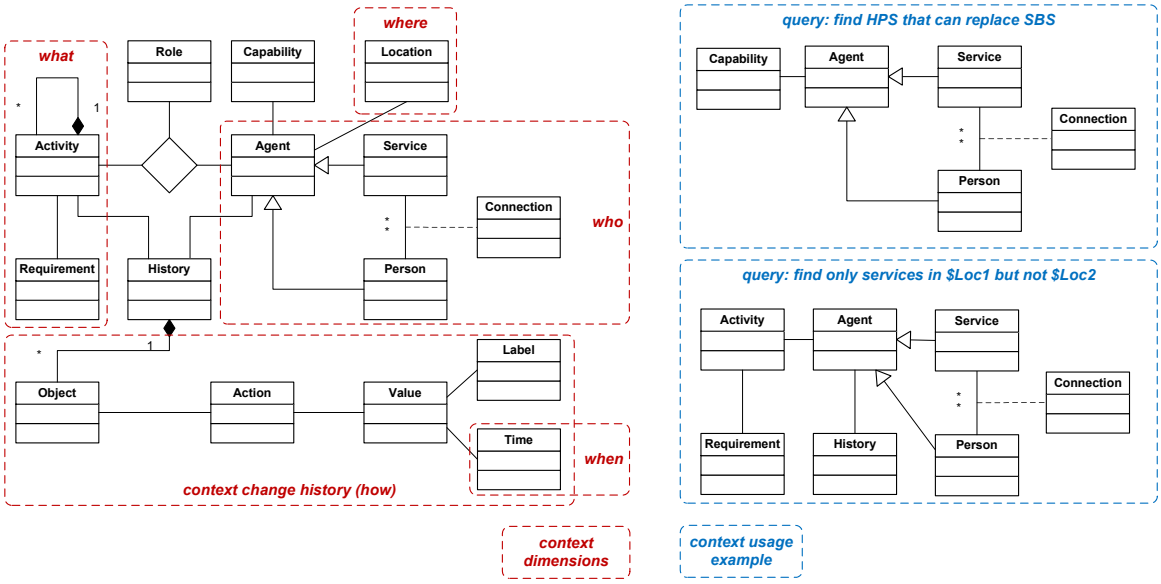


Figure 1. Conceptual Architecture of SOM

However, with the additional flexibility we also face a set of challenges which we address in our work. In particular, we address these challenges with a service mashup model which includes hooks for *context information* and human related QoS attributes that can be exploited during the execution of the mashup (see Figure 1).

In the first step, we create a *mashup model* consisting of HPS and traditional SBS using a lightweight composition language. The focus of this step lies on the functional part of the composition. Since HPS offer the same interface as tradi-



(a) Different Dimensions of Context (b) Using the Context Model

Figure 2. Context Model for Social based Mashup Applications

tional SBS, we do not require additional considerations concerning interface compatibility of HPS and SBS. Considering plain SBS, their description follows the WSDL de-facto standard. By following the HPS concept proposed by [26] we do not require special considerations for the creation of HPS service descriptions. We use WSDL based descriptions that serve as interface to HPSs. This abstraction gives us the benefit of having standardized interfaces to humans which make them *composable* together with SBS in a well defined manner without having to cope with the inherent complexity of human interfaces.

In the second step, we attach *context* related information to the service composition. We refer to this context as design time context in which the creator of the mashup makes assumptions about the expected mashup execution context. The context of the mashup provides information about the environment and introduces global constrains that must be met during the execution of the mashup. For instance, a mashup might not use external services but only internal ones.

Fine tuning of the service mashup takes place in the third step. Within the mashup, the designer is required to define parts which allow a certain *degree of flexibility*, like delegation or splitting. This activity attaches detailed context information to parts of the service mashup which are expected to change during the execution and the given context. For instance, in a mashup, there might be a critical HPS which must not be delegated to others or some services are not allowed to be split among several services.

The rest of the paper is organized as follows. In Section 2 we define our context model for social-based mashup applications. Section 3 introduces our proposed QoS model with emphasis on humans. In the following section we show how to create social service compositions with a lightweight service composition language. The initial results of our prototype are demonstrated in Section 4. We conclude the paper with related work in Section 5 and an outlook for future work in Section 6.

2. Context Model for SOM

We propose a context model that enables the weaving of context related information into social service compositions that are described in a flexible composition language. Our proposed context model aims at satisfying the following properties:

1. The model should be lightweight pertaining information regarding the most relevant entities in SOM. We do not attempt to provide a general purpose context model, but rather focus on SOM applications.
2. Context information comes from various sources inducing physical, e.g., physical location context, and logical sources, for example, a calender containing information regarding the user's location in a given time interval.

Figure 2(a) depicts our context model and some examples of its usage in SOM (see Figure 2(b)). Our model captures the well know dimensions of context (e.g., see [2]) *what* denoting the activity of an agent, *when* capturing time aspects, *who* denoting a person or service, and *how* a set of actions that were executed in the course of an activity. On the other hand (Figure 2(b)) we show examples how to use the context model in SOM (context queries).

The context model contains aspects related to the design of mashups and to their execution (i.e., runtime). At design time, concepts that need to be supported by the mashup tool are *Activity*, *Agent*, *Role*, and *Requirement*.

- **Activity**: Mashup applications comprise a set of activities required to model basic flows (processes). However, the context model does not contain details regarding the process model. Instead, context is related to information such as *Agent* and their *Roles* in *Activities*; roles may include creator of an activity or contributor.
- **Agent**: Both HPS and SBS can contribute to an activity [26].
- **Requirement**: A requirement may restrict the set of actors, which can be invoked in an activity, the roles, and most importantly the values of QoS attributes.

The other model entities address the dynamic nature of context. An *Activity* as well as *Agent* is associated with a *context change history*. Context change, or more precisely, context change events are depicted as *Object*, *Action*, and *Value* triples.

- Events are **actions taken by agents** a) in an activity that is part of a mashup (composition) or b) independent of an activity (person A moves to a new location X). In the first case, an object (person or service) triggers an action (the activity acts as the container for actions) while in the second case, objects change their state independent of any activity.
- Events are **actions acting upon agents**. Such an event may capture invocations of SBS or HPS. In other words, actions capture *functional* capabilities (e.g., interface capabilities) that could change over time. However, actions are typically driven by the context to enable flexible executions of SOM. Changing actions are important for the context-based evolution of mashup-based compositions.

Label and *Time* provide additional metadata to context change events. Labels provide additional information about the action. The value gives the result of an action, whereas labels can be regarded as ”tags” to further quantify actions.

Let us discuss two example context queries as depicted by Figure 2(b) to explain the role of context in SOM:

- Replaceability (*find HPS that can replace SBS*): Attempts to discover a set of HPS that can replace SBS that might be, for example, unavailable due to faults or other technical problems. The HPS must satisfy functional interface characteristics and capabilities in terms of QoS properties. For example, slower response time of a human is acceptable in a given SOM.
- Restriction and filtering (*Find only services in $Loc1 but not $Loc4*): This query selects and filters services based on location information (e.g., exclusive combination of input variables $Loc1 and $Loc4 expressed as *Requirement*). The *History* is used to determine whether an agent, for example, a person has moved to a certain location (changes in location context).

3. QoS Model for Service Compositions

In contrast to interface issues, quality aspects of HPS require special considerations with regard to service compositions. Context has strong impact on QoS properties due to changing availability of services. HPS exhibit fundamentally different quality than SBS. For example, SBS can handle several hundreds requests at once while humans are limited to a small number of parallel requests. On the other hand, humans are able to make complex decisions and to handle data outside the original task specification while SBS can handle these situations only to a very limited extent. Generally speaking, humans maintain a more complex notion of context and are able to maintain multiple contexts at a time. They also have the ability to resolve context when required. Thus, when combining human and software-based services, these differences must be considered in the design of compositions. We propose the extension of existing QoS models with regard to human attributes that reflect human behavior [15].

3.1 QoS Attributes

Table 1 shows a set of quality attributes, along with their definition, that can be applied to both human and machine-based services or to only one of these two types of services. The latter fact (i.e., application) is exhibited with the use of the third and fourth columns of the table. Finally, the last column shows if it is meaningful to aggregate the same quality attribute of both human and software-based services for the composition. If the answer is no, then we can have only local quality constraints on specific tasks of the composition, depending on whether the quality attribute is meaningful to model the quality of the human or machine-based service that is mapped to this task.

We are going to analyze the differences between the same quality attribute for the human and software-based services and we are going to argue why we did not model this attribute for one or the other service type in the corresponding cases that can be drawn from the last column of Table 1.

Throughput: Defines the maximum number of requests that can be completed in a given time interval. Again, there is no conceptual difference between humans and software-based services beside the scale of the respective throughput.

Availability: SBSs are usually available 99.9 percent of their time. On the other hand, the availability of humans varies as it also depends on their context and current load. However, there are usual patterns inferred from users context where availability can be approximately defined for humans (e.g. user’s usual working schedule, days off and holidays, health and mental situation, etc.).

Data quality: This attribute expresses the quality of the data produced by the service. In the way we define HPS, this attribute has the same meaning for both service types that we consider. Moreover, this is an attribute for which we cannot definitely say that its value is better for one service type

Attribute	Description	SBS	Human	Aggregation
Throughput	The number of completed service requests over a time period	yes	yes	yes
Availability	Availability of the service provided to customers	yes	yes	yes
Data Quality	The ability of a data collection to meet user requirements, defined as the proximity of a value v to a value $v^{'}$ considered as correct	yes	yes	yes
Trust	Indicates the service's trustworthiness	yes	yes	no
Delegation	Ability to delegate task to another service	no	yes	no
Soft Completion	Ability to end a task's execution untimely due to time restrictions but with a concrete result produced	no	yes	no

Table 1. Service Quality Attributes

Attribute	Metric/Type/Unit/Monotonicity	Aggregation Pattern
Throughput	Maximum/Positive Integer/(Calls/Sec)/Positive	$th_k\,(CEP) = \min_{sp_m^k \in ep_k} \sum_{\substack{t_i \in sp_m^k \\ (t_i,s_j)\in CEP}} th_j$
Availability	(Uptime/Total-Time)/Real in [0.0,1.0]/–/Positive	$av_k\,(CEP) = \prod_{\substack{t_i \in ep_k \\ (t_i,s_j)\in CEP}} av_j$
Data Quality	MAPE [17]/Real in [0.0,1.0]/–/Negative	$dq_k\,(CEP) = \min_{\substack{t_i \in ep_k \\ (t_i,s_j)\in CEP}} dq_j$
Trust	–/Real in [0.0,1.0]/–/Positive	$tr_k\,(CEP) = \prod_{\substack{t_i \in ep_k \\ (t_i,s_j)\in CEP}} tr_j$

Table 2. Service Quality Metric and Aggregation Pattern

than the other one. Depending on the application domain and the context, the situation changes so there is no winning service type. Finally, the data quality of an HPS can increase because a human can learn from the repetition of a task or by exploiting the knowledge acquired or derived from its social network.

Trust: The way this quality is measured for these entities is different because for humans it depends on both subjective and objective criteria while for SBS it depends on only objective criteria [29]. Moreover, an SBS's trust is usually more constant as compared to a human's trust. So, we believe that it is not meaningful to aggregate this quality for a composite service containing both human and software-based services.

Delegation: This attribute concerns the ability of a human to delegate a part or the whole task he is running to other humans, including this human's coordination capability in coordinating the splitted task's execution. SBS can partially support this attribute by delegating a whole task's execution to other instances of the same service. However, they cannot easily delegate parts of a task and coordinate them, if the task has already executed, as this requires special mechanisms. So it is not meaningful to model this attribute for SBS.

Soft Completion: Soft completion refers to the incomplete result of a service execution which is still useable for further activities. For example, a HPS that analyzes images for the occurrence of objects (e.g., trees) might not qualify all objects. However, it might be the case that only a yes/no decision of object occurrences is necessary in the context of the mashup/workflow. This is not true for the side of SBS, as they have to end all their activities before they can produce a specific and complete output.

3.2 Quality Aggregation Analysis

Several quality attributes can be associated with a service. Each attribute would have a short definition, a metric, a value type, a monotonicity and an aggregation pattern associated with it. Monotonicity concerns the way the values that the dimension takes can be compared. In this paper, we distinguish between positive and negative dimensions. A dimension is positive (negative) if the higher the value the higher (lower) the quality or energy level. The aggregation pattern of a dimension defines how the value of this dimension for a composite service can be determined based on the value of the component services. In this paper, we have considered only the most significant and widely-used quality attributes that appear in many research approaches. Table 2 summarizes these attributes based on the above analysis.

CEP denotes a concrete execution plan [13] of a composite service (either HPS or SBS). Each CEP can be transformed [1] to many (e.g., K) concrete execution paths symbolized with ep_k containing a subset of the tasks t_i of the CEP. In addition, each execution path ep_k has a set of *subpaths* (i.e. paths not having parallel tasks) that are indexed by m and denoted by sp_m^k. Every execution path ep_k is associated with a set of aggregated attributes denoted with $attr_k$. The set of services S_i to be executed for a task t_i are called *candidate services* and are denoted with s_j. Each service s_j has a specific set of attributes $attr_j$ derived from its service profile.

3.3 QoS and Context

We argue that QoS attributes are driven by the context in which they are measured. In this regard, we refer to context

free QoS attributes and context sensitive attributes. We consider QoS attributes like availability as context sensitive for HPS, because the location context of the human (provided service) influences the availability quality attribute. For instance, if the location of a human changes, the human might not be able to provide the service. Another example for a context sensitive attribute is delegation. Delegation might not be allowed in scenarios where a certain instance of a HPS (e.g., expert that provides an expertise service) is required. In contrast, context free quality attributes are not affected by context. An example is a valid security certificate which is required to invoke a service. In the next section we discuss the use of context and QoS in our proposed mashup model in greater detail.

4. Architecture and Implementation

The main concepts of our approach were defined in the previous sections. This section is dedicated to analyzing the architecture and implementation details of our approach. As discussed earlier, our goal is to benefit from both, human flexibility and the efficiency of SBS for tasks. Conceptually, the composition process consists of two main steps as depicted in Figure 3.

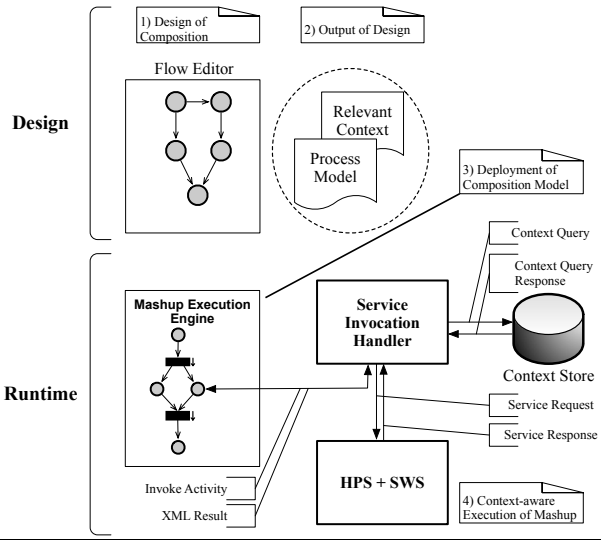


Figure 3. Architecture and Deployment

1. **Design** Firstly, one needs to define the structure of the service mashup. The structure of a service mashup defines from an abstract perspective how services and humans cooperate and which services are used by humans and vice versa. We support this activity with a online tool that enables the creation of social mashups (*Flow Editor*). It must be noted that the flow model contains additional information regarding the *relevant context* that is used during execution of the process model (e.g., context queries).
2. **Runtime** With the use of uniform WSDL interfaces we lay the foundation for a later generation of BPEL workflows. The composition model is deployed in the *Mashup Execution Engine*. An activity is usually performed by invoking a set of services. We call these invocations *actions* that can be performed in a context-aware manner. Thus, the *Service Invocation Handler* interprets context associated to activities to obtain context information via queries from the *Context Store*. Interactions with HPS and SBS (service request and service response) happen in an equivalent service oriented manner through the exchange of SOAP messages. Notice that our approach is not limited to BPEL, since the abstract definition of mashup can be transformed into other languages as well.

4.1 Flow Editor

For a proof of concept prototype implementation, and to illustrate the end user support, we used ExpressFlow [31]. We integrated concepts that are required for the creation of social service compositions in Expressflow, with regard to the limitations imposed by HPS and SBS in service mashups. In our approach, a service mashup structures activities and defines context channels which encapsulate context related information. With the help of a graphical tool (see Figure 4) we support the mashup design process. At this level, the mashup designer defines the basic structure of the mashup using different activities. The tool also provides basic service registry features which supports the designer in selecting services that are suitable in a given context (e.g., filtering of all services that operate at certain locations).

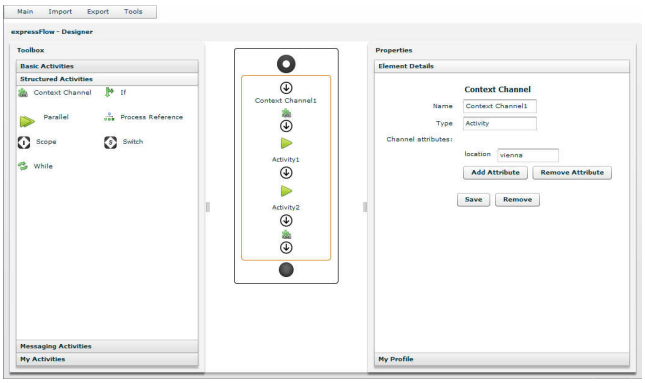


Figure 4. Screenshot of ExpressFlow online tool

We incorporate context related information directly into the code of the mashup. This approach enables context unaware services to be fully integrated into context aware mashups. By following the separation between context and services we gain the needed flexibility to integrate humans, HPS respectively, into mashups. We discuss the implementation and integration of the basic concepts and their implementation in a top down manner in the following subsections and illustrate our approach with short XML examples that

are generated by the Expressflow tool, after having specified the mashup graphically.

4.2 Modeling Context-aware Mashups with Expressflow

Mashup: In service mashups, we embed context channels to structure context information and group actions in activities. These elements can be structured with constructs like *if then else* or *parallel*. Listing 1 illustrates two parallel context channels with different context information using *Parallel* and *ParallelBranch* constructs. Each context channel specifies a context scope (e.g., location, time). During the execution these branches are executed in parallel with the context being evaluated independently.

```
<Process efid="5c21c032-091f-45a0-aaf4..."
name="OOPSLA Service Mashup Demo"
type="Service Mashup" ... >
<Parallel name="Parallel1" type="Activity" ... >
  <ParallelBranch>
    <Context name="Context Channel1"
    type="ContextChannel"
    location="Vienna" time="Today">
    ...
    </Context>
  </ParallelBranch>
  <ParallelBranch>
    <Context name="Context Channel2"
    type="ContextChannel"
    delegation="No" availability="100">
    ...
    </Context>
  </ParallelBranch>
</Parallel>
</Process>
```

Listing 1. Definition of Mashup Comprising Two Parallel Context Channels

Context Channel: Context channels act as flexible containers for service related context information which is specified during the design of the mashup. Conceptually, a context channel defines the scope and the type of context (e.g., location, time, delegation) for nested activities. During the execution of the mashup, all activities of a context channel access the predefined context information and perform the context dependend actions (e.g., data transformation) which are retrieved from a context store.

```
<Context name="Context Channel1"
  type="ContextChannel"
  location="Vienna" time="Today">
  <Activity name="Activity1"
  type="Activity" previous="null" ... />
  <Activity name="Activity2"
  type="Activity" previous="Activity1"
  next="Assignment1" ... />
</Context>
```

Listing 2. Context Channel Example

Listing 2 presents an example for a context channel, which defines the location context of all included activities to *Vienna* and the time context *Today*. It it worth noticing, that, unless specified differently, all activities are per default executed sequentially (Activity 2 follows Activity 1).

Activity: Activities structure actions which are the hooks for the actual service invocation. The example in Listing 3 shows an activity (*Activity1*) which copies three different values (*appid*, *street* and *city*) to the variable *Variable7* for a sequential invocation of two SBS (*SOAPInvoke3* and *SOAPInvoke4*) which share the same input parameters.

```
<Activity name="Activity2" type="Activity"  ... />
<Assignment name="Assignment1" type="Activity ... >
  <Copy name="Copy1" type="Activity"
  copy_from="YD–9G7bey8_JXxQP6rxl.fBFGgCdNj..."
  copy_to="$Variable7.appid" previous="null" ... />
  <Copy name="Copy2" type="Activity"
  copy_from="Argentinierstreet+8"
  copy_to="$Variable7.city"
  previous="null" next="null" ... />
   <Copy name="Copy3" type="Activity"
  copy_from="Vienna"
  copy_to="$Variable7.city"  ... />
</Assignment>
<Invoke name="SOAPInvoke3" type="SOAPInvoke"
input="Variable7" output="Variable8" ... >...
<Invoke name="SOAPInvoke4" type="SOAPInvoke"
input="Variable7" output="Variable9" ... >...
```

Listing 3. Asynchronous Activity

Action: Actions represent invocations of services that are executed in the context of an activity. Listing 4 shows how actions are modeled in ExpressFlow. Because of having specified the context on a higher level, we do not require to specify context attributes on this level. Consequently, services do not need to be aware about the context in which they are executed. We discuss how we handle the actual service invocation in Section 4.3.

```
<Invoke name="SOAPInvoke3"
type="SOAPInvoke"
input="Variable7" output="Variable8" ... >
  <Resource
   uri="http://local.yahooapis.com/MapsService...
  appid=$Variable7.appid&amp;
  street=$Variable7.city&amp;
  city=$Variable7.city"/>
</Invoke>
```

Listing 4. SOAP Invoke Example

4.3 Context Store and Service Invocation Handler

Our prototype stores context related information in a MYSQL database. Our database layout is based on the SOAF data model [30] and the context model of Figure 2(a). During runtime, we query the context store for context related actions (e.g., service request transformations) using our service invocation handler. The service invocation handler extracts context information (e.g., time, location, delegation) from SOAP message headers. In our current implementation, we use a simple keyword based search to query for context spe-

cific transformations which are represented as XSLT transformations. These transformations are retrieved as strings (streams respectively) for the use of Apache XALAN to transform the request according to the context. Notice that, in our current implementation, we support the transformation of incoming SOAP requests, but do not transform the output accordingly. This is planned for future work.

5. Related Work

Generally speaking, mashups are applications created of existing online resources. [16] categorizes mashups according to four main dimensions: a) what is mashed up, b) where to mashup, c) how to mash up, and d) for whom to mash up. Based on this categorization, there are tools that offer similar functionalities with our approach like "JackBe Presto"[1] , "Procession" [20], "Serena Mashup Suite"[2] , "Swashup" [22], "JOpera" [25] and "remash!" [4]. However, none of these tools is able to offer a context-aware and QoS-based mashup development and execution environment.

Context is any information that can be used to characterize the situation of entities [12]. Context-aware Systems (CASs) are able to adapt their behavior to the current context without explicit user intervention and thus aim at increasing usability and effectiveness by taking environmental context into account [2]. The behavior of a CAS can be adapted through context in three levels/dimensions [10]: *user interface*, *content*, and *service*. Moreover, this adaptation can be performed in a static or dynamic way by different combinations of services which are independently selected on these three dimensions.

Context has been used in discovery, composition, and adaptation of SBSs. Concerning SBS discovery, context has been used for request (e.g. location info) and input completion (e.g. missing input) so as to increase the quality of the discovery result [6]. Various approaches [9, 23, 24, 28] have been proposed for SBS composition that use local [33] and global contextual constraints for selecting among the candidate SBSs for each task of the composite SBS in a static or dynamic way.

When context changes, composite SBSs may be adapted in three different ways: a) SBS access channel is changed [3], b) an SBS is substituted with another one [3, 28], c) a new concrete execution plan is executed from scratch [14]. Unfortunately, none of the existing SBS composition and adaptation approaches is able to offer simultaneously the three different types of adaptation. Moreover, no approach is able to substitute a single SBS with a new composite one, which might be the case with cooperating HPSs.

Very few mashup approaches have been proposed to take advantage of user or environmental context for adaption purposes. In [5] a system architecture is proposed featuring a context provisioning framework for utilizing local sensors in context-aware mashups. The work described in [19] presents a services mashup system which is able to perform context-aware service composition in a semi- or fully-automatic way and to adapt the results of the composition according to the user's context. Finally, "remash!" [4] offers a framework that enables the flexible binding of services at runtime depending on the changing availability of services or the situation-specific requirement of the application.

Context and its quality can affect the QoS of a service [7]. QoS has been widely studied and researched for SBS discovery, composition [13], and adaptation [1, 8, 11]. However, no QoS-related research work has been conducted for mashups or human-based workflows.

6. Summary and Future Work

In this paper we presented a framework for the integration of humans in socially oriented service mashups. We illustrated the mechanisms to accomplish this with regard to QoS and context. We presented our initial prototype and discussed how we addressed implementation challenges concerning the interpretation of context during the runtime. In future work, we will elaborate our approach with regard to the adaptivity of mashups and extend our core context model with complex actions (e.g., reordering of service invocations in context channels). We are going to extend our QoS model with attributes like accuracy or presentation quality and study these in the context of mashups. Furthermore, we will extend our current prototype with ExpressFlow to BPEL [32] transformations to generate executable BPEL code and study alternative approaches (e.g., using scripting languages or document based approaches [27]). And finally, we will evaluate the performance of our proposed approach thoroughly and study larger examples for service mashups and context.

Acknowledgment

The research leading to these results has received funding from the European Community Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube) and 216256 (COIN). We thank Martin Vasko for help on modeling SOM with ExpressFlow.

References

[1] D. Ardagna and B. Pernici. Adaptive service composition in flexible processes. *IEEE Trans. Softw. Eng.*, 33(6):369–384, 2007.

[2] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *Journal on Ad Hoc and Ubiquitous Computing*, 2007.

[3] L. Baresi, D. Bianchini, V. D. Antonellis, M. G. Fugini, B. Pernici, and P. Plebani. Context-aware composition of e-services. In *TES 2003*, volume 2819 of *LNCS*, pages 28–41, Berlin,Germany, 2003. Springer.

[1] `http://www.jackbe.com`

[2] `http://www.serena.com`

[4] B. Blau, S. Lamparter, and S. Haak. remash! - blueprints for restful situational web applications. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW2009*, Madrid, Spain, 2009.

[5] A. Brodt, D. Nicklas, S. Sathish, and B. Mitschang. Context-aware mashups for mobile devices. In *Web Engineering (WISE '08)*, pages 280–291. Springer-Verlag, 2008.

[6] T. H. F. Broens, S. Pokraev, M. J. van Sinderen, J. Koolwaaij, and P. D. Costa. Context-aware, ontology-based, service discovery. In *Symposium on Ambient Intelligence*, volume 3295 of *LNCS*, pages 72–83. Springer, 2004.

[7] T. Buchholz, A. Küpper, and M. Schiffers. Applying web services technologies to the management of context provisioning. In *10th International Workshop of the HP OpenView University Association*, July 2003.

[8] G. Canfora, M. D. Penta, R. Esposito, and M. L. Villani. QoS-Aware Replanning of Composite Web Services. In *Proceedings of the IEEE International Conference on Web Services (ICWS'05), Orlando, FL, USA*, pages 121–129, 2005.

[9] F. Casati, M. Castellanos, U. Dayal, and M.-C. Shan. Probabilistic, context-sensitive, and goal-oriented service selection. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 316–321, New York, NY, USA, 2004. ACM.

[10] T. Chaari, F. Laforest, and A. Celentano. Design of context-aware application based on web services. Technical Report CS-2004-5, Università Ca'Foscari di Venezia, Venezia, Italy, April 2004.

[11] G. Chafle, K. Dasgupta, A. Kumar, S. Mittal, and B. Srivastava. Adaptation in web service composition and execution. *IEEE International Conference on Web Services (ICWS)*, pages 549–557, 2006.

[12] A. K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Proceedings of the Workshop on the What, Who, Where, When and How of Context-Awareness, New York*. ACM Press, 2000.

[13] A. M. Ferreira, K. Kritikos, and B. Pernici. Energy-aware design of service-based applications. In *ICSOC*, LNCS. Springer, 2009.

[14] K. Fujii and T. Suda. Semantics-based context-aware dynamic service composition. *ACM Trans. Auton. Adapt. Syst.*, 4(2):1–31, 2009.

[15] R. Kern, C. Zirpins, and S. Agarwal. Managing quality of human-based eservices. *Service-Oriented Computing – ICSOC 2008 Workshops*, pages 304–309, 2009.

[16] A. Koschmider, V. Torres, and V. Pelechano. Elucidating the mashup hype: Definition, challenges, methodical guide and tools for mashups. In *Proceedings of the 2nd Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web at WWW2009*, Madrid,Spain, April 2009.

[17] K. Kritikos. Qos-based web service description and discovery. Phd thesis, Computer Science Department, University of Crete, Heraklion, Greece, December 2008.

[18] K. Kritikos and D. Plexousakis. Mixed-Integer Programming for QoS-Based Web Service Matchmaking. *IEEE Transactions on Services Computing*, 2(2):122–139, 2009.

[19] Y. Li, J. Fang, and J. Xiong. A context-aware services mashup system. In *Seventh International Conference on Grid and Cooperative Computing (GCC '08)*, pages 702–712, Shenzhen, China, 2008. IEEE.

[20] P. S. Limited. Procession process engine data sheet. Technical report, 2008.

[21] E. M. Maximilien, A. Ranabahu, and K. Gomadam. An online platform for web apis and service mashups. *IEEE Internet Computing*, 12(5):32–43, 2008.

[22] E. M. Maximilien, A. Ranabahu, and S. Tai. Swashup: situational web applications mashups. In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 797–798, Montreal, Quebec, Canada, 2007. ACM.

[23] S. B. Mokhtar, D. Fournier, N. Georgantas, and V. Issarny. Context-aware service composition in pervasive computing environments. In *RISE*, volume 3943 of *LNCS*, pages 129–144. Springer, 2005.

[24] S. K. Mostéfaoui and B. Hirsbrunner. Towards a context-based service composition framework. In *ICWS '03*, pages 42–45, Las Vegas, Nevada, USA, June 2003. CSREA Press.

[25] C. Pautasso. Composing restful services with jopera. In A. Bergel and J. Fabry, editors, *Software Composition*, volume 5634 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2009.

[26] D. Schall, H.-L. Truong, and S. Dustdar. Unifying Human and Software Services in Web-Scale Collaborations. *IEEE Internet Computing*, 12(3):62–68, 2008.

[27] N. Schuster, C. Zirpins, S. Tai, S. Battle, and N. Heuer. A service-oriented approach to document-centric situational collaboration processes. In *WETICE '09: Proceedings of the 2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 221–226, Washington, DC, USA, 2009. IEEE Computer Society.

[28] Q. Z. Sheng, B. Benatallah, Z. Maamar, and A. H. Ngu. Configurable composition and adaptive provisioning of web services. *IEEE Transactions on Services Computing*, 2(1):34–49, 2009.

[29] F. Skopik, D. Schall, and S. Dustdar. The cycle of trust in mixed service-oriented systems. In *SEAA*, 2009.

[30] M. Treiber, H.-L. Truong, and S. Dustdar. Soaf –design and implementation of a service-enriched social network. *Web Engineering*, pages 379–393, 2009.

[31] M. Vasko and S. Dustdar. Introducing Collaborative Service Mashup Design. In *Lightweight Integration on the Web (ComposableWeb'09)*, pages 51–62. CEUR - Workshop Proceedings, June 2009.

[32] WS-BPEL. Business Process Execution Language for Web Services Version 2.0, April 2007.

[33] Y. Yamato and H. Sunaga. Context-aware service composition and component change-over using semantic web techniques. *IEEE International Conference on Web Services (ICWS)*, pages 687–694, 2007.

A.4 Adaptation of Service-Based Applications Based on Process Quality Factor Analysis

Authors:

FBK: Raman Kazhamiakin

USTUTT: Branimir Wetzstein

USTUTT: Dimka Karastoyanova

FBK: Marco Pistore

USTUTT: Frank Leymann

- Submitted to: MONA+ 2009

Adaptation of Service-Based Applications Based on Process Quality Factor Analysis

Raman Kazhamiakin[1], Branimir Wetzstein[2], Dimka Karastoyanova[2], Marco Pistore[1], and Frank Leymann[2]

[1]FBK-Irst, via Sommarive 18, 38100 Trento, Italy
{raman,pistore}@fbk.eu
[2]Institute of Architecture of Application Systems, University of Stuttgart, Germany
{karastoyanova, leymann, wetzstein}@iaas.uni-stuttgart.de

Abstract. When service-based applications implement business processes, it is important to monitor their performance in terms of Key Performance Indicators (KPIs). If monitoring results show that the KPIs do not reach target values, the influential factors have to be analyzed and corresponding adaptation actions have to be taken. In this paper we present a novel adaptation approach for service-based applications (SBAs) based on a process quality factor analysis. This approach uses decision trees for showing the dependencies of KPIs on process quality factors from different functional levels of an SBA. We extend the monitoring and analysis approach and show how the analysis results may be used to come up with an adaptation strategy leading to an SBA that satisfies KPI values. The approach includes creation of a model which associates adaptation actions to process quality metrics, extraction of adaptation requirements based on analysis results, and identification of an adaptation strategy which can consist of several adaptation actions on different functional levels of an SBA.

Keywords: Adaptation, Service-based Applications, KPI, QoS-aware SBAs

1 Introduction

In recent years extensive attention has been paid to devising and improving the concepts and infrastructures for service-based applications (SBAs) [1]. SBAs can be viewed in terms of three functional layers, namely (i) business processes, (ii) service compositions that implement these business processes, and (iii) services and service infrastructure. A major concern for enterprises is to ensure the quality of their SBA-based business processes. Thereby, process quality goals are specified in terms of Key Performance Indicators (e.g., order fulfillment time), i.e. key process metrics that contain target values which are to be achieved in a certain period. KPIs of business processes that are implemented in terms of SBAs are typically monitored using business activity monitoring technology. If monitoring results show that KPIs do not meet target values, further *process quality factor analysis* is needed to find out which of the lower level process metrics (e.g., duration of process activities, type and

amount of ordered products etc.) or QoS metrics (e.g., availability of IT infrastructure) mostly influence KPI target violations.

After influential factors of KPI violations are identified, the goal is to perform *process adaptation* in order to prevent KPI violations for future process instances or even for the running instances. Thereby, several challenges arise. Firstly, one has to choose appropriate adaptation actions for each influential factor identified (e.g., selection of a faster delivery service in order to decrease deliver time). Secondly, one has to take into account that an adaptation action can have positive effect on one metric but negative effect on others (e.g., a faster delivery time normally involves higher costs). Thus, when adapting the process one has to choose a set of adaptation actions which improve the influential factors as shown in the analysis and take into account their effects.

In order to deal with this adaptation problem, we extend previous work described in [2] which uses decision trees for process quality factor analysis. Based on this work, in this paper, we show how to extract a set of adaptation requirements from the decision tree and find an adaptation strategy consisting of a set of adaptation actions which takes into account both positive and negative effects of adaptation actions on metrics. We discuss limitations of our approach so far and show several possibilities for extending the approach in future work.

We continue the argumentation about the motivation of our work presented here in the next section and support it with a scenario. In section 3 the overview of the approach is given. The details on modeling and analysis of the influential factors are given in Section 4, while the identification and selection of adaptation actions is presented in Section 5. A discussion of the approach and possible extensions are described in Section 6. Overview of related work, as well as conclusions and plans for future work conclude the paper.

2 Scenario and Motivation

In this section we present the motivation for our approach and a scenario which we use in the following sections for explaining our concepts based on examples. This scenario has already been used in [2] for experimental evaluation of process quality factor analysis. The scenario consists of a purchase order process implemented by a reseller which offers products to its customers and interacts with its suppliers, a banking service, and a shipment service for processing the order. The customer sends a purchase order request with details about the required products and needed amounts to the reseller. The latter checks whether all products are available in the warehouse. If some products are not in stock, they are ordered from suppliers. The reseller waits, if needed, for the supplier to deliver the needed products. When all products are in place, the warehouse packages the products and hands them over to the shipment service, which delivers the order to the customer, and finally notifies the reseller about the shipment. In parallel to the packaging and shipment, the payment subprocess is performed. For measuring the performance of its business process, the reseller defines a set of Key Performance Indicators (KPIs). A typical KPI for the reseller in our scenario is order fulfillment lead time [3], which measures the number of days from order receipt to the delivery of the ordered products at the customer. A

KPI is a key metric (with either technical or business meaning) with target values which are to be achieved in a certain analysis period (e.g., order fulfillment lead time < 5 days). After specifying a set of KPIs with target values, they have to be measured based on executed process instances. If the measurement shows an unsatisfying result, i.e. the KPI targets are violated, the reseller wants to improve its process, for instance, by using process adaptation.

Due to the fact that KPIs are complex characteristics that rely upon a wide range of factors originating from different functional levels, adaptation of underlying SBAs is not a straightforward approach. In our scenario the KPI may be influenced by many factors (which have to be measured both on process level (process metrics) and service infrastructure level (QoS metrics): duration of sub-processes and activities, response time and availability of used services, ordered products and their properties such as number of ordered items, product type and size, cost of delivery service, availability of IT infrastructure etc. All those factors and a combination of those can lead to late delivery. Thus, the first step needed is to perform a process quality factor analysis and find out the *influential factors* for KPI violations.

In order to improve those factors, different *adaptation actions* may be considered, for example, replacing a service either dynamically or using a predefined set of services; renegotiating the Service Level Agreements (SLAs) with the corresponding service provider; outsourcing a subprocess or replacing it with a service from an external provider. On infrastructure level, possible adaptations are replacement of IT components with faster ones, clustering for improving availability, upgrading hardware components etc. For a particular situation, different adaptation actions and their combinations may be necessary for improving the same KPI; consistency and non-contradicting actions with respect to the KPI (and perhaps other KPIs) needs to be ensured. This is because an adaptation action can positively affect one influential factor but negatively others. Assume, for example, the selection of a new better performing service which leads to a better response time but negatively affects the cost metric. Thus, when adapting we have to take into account that the adaptation can affect other KPIs (e.g., overall process cost or customer satisfaction, etc.). We call the collection of the adaptation actions that, being enacted in combination, achieve the desired outcome an *adaptation strategy*. It is, therefore, crucial to identify and compose a strategy that improves KPIs in a coherent and integrated manner.

Finally, one can distinguish between the adaptation of the SBA model or a particular SBA instance. In the first case, the KPI violation is prevented for all the future instances. That is, if certain influential factor (e.g., service response time) is identified, the application adaptation is permanent (e.g., service will be substituted in all future instances of the SBA). Another scenario may concern adaptation of a particular running instance of the SBA (i.e., a particular customer order), where the changes are done on-the-fly in order to proactively prevent the violation of the goal for this instance (e.g., violation of an SLA with a particular customer).

In summary, the goal of our approach is to monitor and identify the reasons leading to the violations of KPIs, and then to find the coherent and complete adaptation strategy that would improve the KPI according to the desired value.

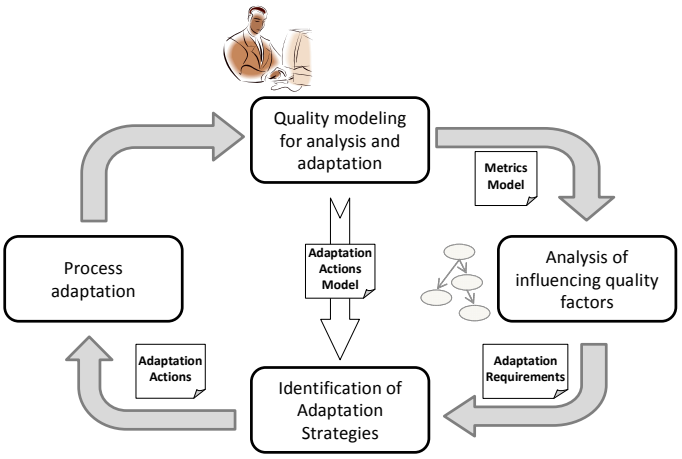


Fig. 1. Quality factor analysis and adaptation

3 Overview of the Approach

In this work we present an approach that allows for adapting service-based applications in order to prevent the violation of KPIs. The overall process is represented in Fig. 1. This approach consists of the following four phases:

- *Quality modeling for analysis and adaptation.* At design time the metrics model and the adaptation actions model are created (presented in detail in Sections 4.1 and 5.1). In the *metrics model*, the user specifies the application KPIs, and the quality metrics representing the *potential* influential factors of KPIs. Obviously, the user does not yet know the influential factors (but might suspect them), however he has to model potential metrics so that they are monitored in the first place and thus can be used during analysis. In the *adaptation actions model*, the user specifies the available adaptation actions per metric and the effect of those actions on application metrics specified in the metrics model. In particular, this model allows for defining whether an action contributes positively or negatively to a certain quality factor, i.e., whether it improves the value of a metric. Same as for metrics, the user just specifies the potential adaptation actions in isolation at this phase, without knowing yet which of those will be needed at adaptation time and in which combination.
- *Analysis of influential quality factors.* In the second phase, based on the *metrics model*, the monitoring of KPIs and potential influential quality metrics is performed across the instances of the application; the information is continuously aggregated and updated. Then the metrics related to *previous executions* of a given application are analyzed in order to identify the reasons, i.e., the *influential factors*, which lead to the undesired values of the specified KPIs. More precisely, we use machine learning techniques to construct a decision tree which shows for which value ranges of influential application metrics a KPI is satisfied or violated (see also Section 4.2, and [2]). As a result,

one identifies those tree paths of application metrics (and their value ranges) that correspond to the "bad" values of the KPI and thus fail the underlying business goal. (Note that in case of an adaptation of a process instance, the results of the analysis, i.e., the set of influential factors and their values, are compared with the values of the metrics already observed for the instance. In this way one restricts only to those metrics that are critical for that specific instance. For example, if the duration of the delivery sub-process is a critical factor only in combination with the products of large size and in the instance small product size is managed, then this factor may be excluded from the consideration.) From the identified tree paths we extract a set of (alternative) *adaptation requirements* each consisting of a conjunction of predicates over metric values (e.g., delivery time shipment < 2 days AND delivery time supplier < 3 days) which lead to KPI satisfaction. The result of the analysis characterizes thus those factors of the application that should be improved, i.e., that are subject of adaptation, and how they should be improved (their values).

- *Identification and selection of an adaptation strategy:* In the next phase the approach aims to combine, and enact concrete *adaptation actions* that address the identified requirements as part of a coherent *adaptation strategy* (Section 5.2). This phase relies on the adaptation action model, where the effect of those actions on different application metrics is described. It takes into account that an adaptation action contributes positively or negatively to a certain quality factor, i.e., whether it improves the value of a metric. After identification of a set of alternative adaptation strategies, one strategy is selected based on certain criteria.
- *Process adaptation:* The selected adaptation strategy is used for adaptation of the process model or process instance by executing all contained adaptation actions. After adaptation, the existing KPIs and metric definitions might have to be adapted thus closing the cycle.

In the next sections we will present the models and mechanisms exploited at different steps of this process in details.

4 Monitoring and Analysis of Quality Factors

The first step of our approach is to analyze the dependencies of the KPI on other quality factors. The goal of this step is to determine the main influential factors which lead to violations of KPI targets. This information will be then exploited for the identification of the adaptation requirements and strategies (Section 5). The basis for this analysis is the model of quality metrics. It contains all the properties (KPIs and potential influential factors) that should be monitored.

4.1 Model of Application Metrics

The quality properties of the application may be captured at different levels of the application stack. As we already mentioned in Section 2, at the business level this properties are represented as *key performance indicators* (KPIs). A KPI usually is a

complex metric over the executions of the application (i.e., business process instances), which is associated with the target value that should be achieved in a certain period.

At the level of service composition the relevant quality properties of the application may be captured by the *process performance metrics* (PPM), which express the characteristics of the execution of the service composition realizing the business process, and by the *quality of service* (QoS) metrics of the underlying services. The PPMs are defined based on the run-time process data and focus on a specific aspect of the process model (e.g., type of the product, duration of the shipment subprocess). The QoS properties usually express the non-functional aspects of the services exploited by the composition (e.g., average response time of the order processing service). Finally, at the lowest level of the application stack the critical parameters may be captured by the *infrastructure metrics*, which characterize the quality of the underlying middleware, containers, etc.

To capture these properties and to use them in our approach, we propose the model of a quality metric that consists of the following elements:

- *Metric measurement definition*. This information is used to characterize the way the metric is monitored and evaluated at runtime. Depending on the corresponding element of the application and on the realization of the monitoring framework, it may include the description of the measurement function or monitored property, correlation information needed to associate the measurement to the business process, the basic events used in case of composite metrics, etc. It also contains the *data range* of the metric, i.e., the possible values the metric may have. We remark that the specific ways this information is defined is out of scope of this paper; more details may be found in [2].
- *Target value* (optional, only needed for KPIs). For KPIs, which express the expectations of the business analysis about the performance of the application, it is necessary to specify the target value range of the metric (e.g., order fulfillment lead time < 5 days). This value is then used by the monitoring framework to identify performance violations and to perform the analysis of influential factors that led to those violations.
- *Potential influential factors*. For the metrics, value of which may depend on the values of other metrics (like for the KPI in the scenario presented in Section 2), it is possible also to list those metrics as potential influential factors. This list will be then used by the quality factor analysis to restrict the search during analysis [2]. Note, however, that this information is necessary only to improve the results of the analysis; in this way only a limited set of metrics is considered.

For the approach presented here the key elements of the metric model are the data range of the metric, the target value range, and the set of potential influential factors, while we omit for the simplicity the description of how to monitor the property.

Definition 1 (Quality Metric). *A* quality metric μ *is defined as tuple* $\mu = \langle r,t,M_f \rangle$ *where*

- $r \subseteq D$ is a range of metric values, and D is some data domain (e.g., real-number domain, or a set of nominal values);
- $t \subseteq r$ is the target range of metric values, which lead to KPI success;

– $M_f = (\mu_1, \ldots, \mu_n)$ is a set of potential influential factors of the metric.

We denote as M a set of all application metrics.

Example 1. Consider the KPI "Order Fulfillment Lead Time" from our scenario. It can be defined as follows in our model:

– It has the duration (time) as data domain and r =[0..inf]. We define t = [0..5].
– M_f = (Shipment delivery time, Order in Stock, Supplier Delivery Time, Supplier Service Availability, Ordered Product Types and Amounts, Order Process Availability, …).

4.2 Identification of Influential Factors

Our approach to quality factor analysis is based on machine learning techniques, more specifically decision tree algorithms [4]. The approach, its assumptions and reasons for using machine learning techniques, have been described in detail in [2]. In the following we will give only a general overview based on an example.

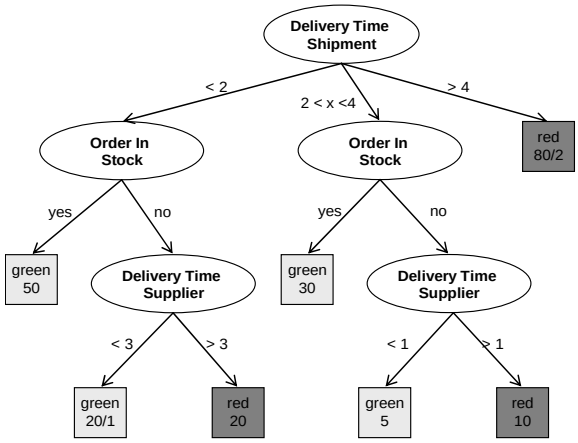


Fig. 2. An Example Dependency Tree

The result of this analysis is a decision tree as shown in Fig. 2, called a *dependency tree* as it shows the main quality factors the KPI *depends* on. The tree is generated automatically for a KPI selected by the user. The leaves of the tree show the classification of the KPI, i.e. whether it is satisfied ("green") or violated ("red") in relation to its target values, and the number of process instances which led to this path. Note that the classification (satisfied or violated) could be extended towards more than two nominal values or even numerical value ranges (regression trees); this is part of our future work. The other nodes of the tree are the main influential factors (metrics) and the branches contain conditions on those metrics.

An example tree which could have been generated for our scenario process is shown in Fig. 2. This tree shows that there are three main influential factors for the KPI "order fulfillment lead time", namely "delivery time shipment", "order in stock", and "delivery time supplier", and for which value ranges of those metrics the KPI has

been violated or satisfied. For example, when the delivery time of the shipper was > 4, the KPI target has been violated ("red") in 80 process instances and was satisfied only in two process instances in that case.

The dependency tree shows the main influential factors which lead to the violation or fulfillment of KPI target values for different value ranges of corresponding metrics. The goal of our approach is to use this dependency tree as basis for coming up with strategies for adapting those influential factors in such a way that for future or running process instances only the paths to KPI fulfillment are taken.

5 Identification of Adaptation Requirements and Strategies

Once the influential factors are highlighted, it is necessary to identify those elements of the application that should be improved, i.e., to identify *adaptation requirements*. Based on those requirements and on the model of adaptation actions associated to the quality properties of the application, possible *adaptation strategies* are identified and triggered.

5.1 Model of Adaptation Actions

In order to adapt different elements of the application at all the levels of the application stack, i.e., to enable a holistic adaptation strategy, it is necessary to provide a generalized model of possible adaptation actions. This model should relate different adaptation mechanisms to the quality properties of the application, which are subject of adaptation. In other words, the model should characterize the available adaptation actions to the model of metrics described above. More precisely, the definition of adaptation actions in our approach consists of the following elements:

- *Adaptation mechanism*. This part of the definition characterizes the machinery or a technique used to realize the specified adaptation action. For example, the adaptation action "replace a service" may be realized by a composed mechanism, which consists of service discovery and binding; while the action "re-negotiate delivery time" may refer to the automatic or semi-automatic negotiation mechanisms. It contains also the specific references and instructions (*adaptation parameters*) for the framework in order to enact the adaptation actions. For the above actions, the adaptation parameters may characterize the expected service properties (e.g., expected response time) to be used through service discovery or expected range of the quality parameter to be achieved by the re-negotiation.
- *Adaptation effect*. To relate the adaptation action to the system we characterize the former in terms of the effects the action causes on one or another application quality metric. We say that the action has a *positive effect* on the metric if the value of the latter is improved as a result of the application of the action. We say that the action has a *negative effect* on the metric if the value of the latter is worsened as a result of the application of the action. Otherwise, we say that the action does not affect the metric.

We remark that for the presented approach we abstract away the details on how the adaptation actions are realized and enacted. The core part of the model of the adaptation action is how the action relates to the application metrics.

Definition 2 (Adaptation action). Adaptation action *a is defined as a pair* $\langle M^+, M^- \rangle$ *where*

- $M^+ \subseteq M$ is a set of metrics, on which the action has a direct positive effect;
- $M^- \subseteq M$ is a set of metrics, on which the action has a direct negative effect.

We denote as A the set of all available adaptation actions.

Using the above model it is possible to annotate a wide range of adaptation actions that are available and applicable at different levels of the application functionality. These actions may include (but are not limited to)

- *Service substitution*, i.e., replacement of one of the component services with another one in order to improve certain quality characteristics. Depending on the application domain, this action may be accomplished by the mechanisms relying on dynamic service discovery and dynamic binding [5], selection and deployment of one of the services from a predefined list (e.g., from an enterprise service registry);
- *Re-negotiation of quality parameters.* The action would allow in automatic way to re-negotiate the quality parameters of the services used in the composition.
- *Re-composition of the underlying service composition or a part of the process*.
- *Replacement of a subprocess with another subprocess or with a single service* (process outsourcing, [6]).
- *Infrastructural reconfiguration.*

Example 2. In our scenario, we could define an adaptation action "renegotiate shipment delivery time" which changes the SLA with the shipment service provider considering the expected shipment delivery time. In that case we would define:

- M^+ = {Supplier Delivery Time};M^- = {Supplier Delivery Cost}

5.2 Identification of Adaptation Requirements

After the dependency tree is generated (as discussed in Section 4.2), the next step is to identify the adaptation requirements for the application in order to improve the performance of the application. This activity relies on the analysis of the dependency trees of the relevant KPIs of the application. Intuitively, this analysis may be described as follows.

- As a first step it is necessary to understand, which of the violations of the KPI (i.e., which of the "red" blocks) should be prevented. For example, it may be the case that all the possible violations should be avoided. In this case it is necessary to find an adaptation strategy (consisting of possibly several adaptation actions) preventing all of those situations. It is also possible to prevent violations only in selected situations, e.g., the most frequent ones. In this case, the other violation cases are ignored and excluded from the tree. We remark that this decision may

be done by the business analysts or even automatically, based on some predefined criteria (e.g., for the cases where number of violations exceeds 10%).

- Second, it is necessary to associate the violations with the influential factors that might help avoiding the violations. This is done by identifying all the metrics in the nodes (and their sub-ranges) on the path from the "red" node to the root of the tree. If some of these metrics is improved such that there are no violations, then the adaptation will be successful. In the above dependency tree in order to improve for the central "red" node, it is enough to improve the metric "Delivery Time Supplier" to the value of < 1. On the other hand, in order to improve for the rightmost "red" node it is not enough to improve the value of the "Delivery Time Shipper" to the value of < 4. Indeed, the other violations are still possible and the other metrics should be improved as well. If a running instance is to be adapted and some of the metric values are already known for this instance, then obviously some paths of the tree leading to "red" nodes might be irrelevant for that instance and can be excluded from further consideration.
- The final step takes into account the need to consider all the selected "red" nodes together in order to merge the appropriate actions into a complete set of adaptation requirements.

In order to realize this approach, we rely on the algorithms provide by the decision procedure for the Satisfiability Modulo Theories (SMT [7]). In this problem, the goal is to find solutions for a set of logical constraints (formulas) with respect to combinations of background theories, such as the theory of real or integer numbers, Boolean arithmetic, and even complex data structures. In other words, given a combination of formulas, the decision procedure may find assignments of the terms in the formula that satisfy the combination of constraints. Below we show how the problem of finding adaptation requirements may be expressed in terms of SMT problem, and how the adaptation requirements may be then extracted.

Our goal is to avoid all the paths in the tree that lead to the "red" leaf nodes. That is, the combination of the "metric-range" pairs on the path should not occur. This combination may be represented as a conjunction of expressions over those metrics, i.e., for a path with n nodes we built an expression $(\mu_1,r_1) \wedge \ldots \wedge (\mu_n,r_n)$, where (μ_i,r_i) represents an expression over the i^{th} metric on the path. For instance, for the central "red" path in the example tree the expression would be as follows:

(2<"Del. Time Shipment"<4) $\wedge$ ("Order in Stock"=yes) $\wedge$ ("Del.Time Supplier">1)

If in case of running instance adaptation, some of these metrics values are known, then the expression can be simplified. If, e.g., "Order in Stock" is false, then the expression is already known to be false and thus this "red" path is already avoided for this instance. If "Order in Stock" is true, then it can simply be removed from the expression, thus simplifying later analysis.

Formula 1. In order to avoid those paths, our goal is to make all those expressions false, i.e., for m paths, we have to find possible assignments of metric values such that the following formula becomes true[1]:

[1] Here the notation $\neg r_{ij}$ stands for complement of the specified range.

$$((\mu_{11},\neg r_{11}) \vee \ldots \vee (\mu_{n1},\neg r_{n1})) \wedge \ldots \wedge ((\mu_{1m},\neg r_{1m}) \vee \ldots \vee (\mu_{nm},\neg r_{nm}))$$

It is easy to see that if the formula is satisfied, then neither "red" node is reachable. The result of the analysis is represented as a set of alternatives, each of which contains the list of metrics that should be adapted and their expected ranges. In order to carry out the analysis task we use the MathSAT tool [8], which implements the SMT decision procedure.

More precisely, we define the resulting adaptation requirements as $R = \{A_1,...,A_n\}$, where $A_i = \{ (\mu_{1i},r_{1i}),\ldots,(\mu_{mi},r_{mi})\}$ is a set of metric-range pairs that should be achieved in order to address the adaptation needs.

Example 3. We present the approach using the dependency tree depicted in Fig. 2 (we assume that the goal in the example is to avoid any of possible violations). For the metrics "Delivery Time Shipment" (Sh), "Delivery Time Supplier" (Su), and "Order in Stock" (O), and for the three paths to "red" nodes we construct the following three constraints:

- Sh < 2 ∧ O=no ∧ Su > 3; Sh > 2 ∧ Sh < 4 ∧ O=no ∧ Su > 1; Sh > 4

Based on these clauses, we need to satisfy the following formula:

- (Sh > 2 ∨ O=yes ∨ Su < 3) ∧ (Sh < 2 ∨ Sh > 4 ∨ O=yes ∨ Su < 1) ∧ (Sh < 4).

The result of the analysis provided by the tool represents the following alternatives:

- (2 < Sh < 4) and (Su < 1)
- (Sh < 2) and (Su < 3)
- (O=yes) and (Sh < 4)

That is, to avoid violations of the KPI it is necessary to improve the metric "Delivery Time Supplier" to the value < 1 and the metric "Delivery Time Shipment" to the value from 2 to 4, or alternatively improve the metric "Delivery Time Supplier" to the value < 3 and the metric "Delivery Time Shipment" to the value < 2, or "Order in Stock" to become true and the "Delivery Time Shipment" to the value in range < 4. Note that as the "Order in Stock" metric is not adaptable, the third alternative is not relevant for adaptation of process models (future process instances); however it could be used for adaptation of running process instances where O=yes.

5.3 Identification of Adaptation Strategies

After the adaptation requirements are identified, the next step is to associate possible adaptation strategies which should lead to KPI fulfillment, i.e. the sets of adaptation actions that adapt the corresponding influential factors. As described in Section 5.1, for adaptable metrics a set of possible adaptation actions has been specified. The first step is thus to come up with alternative adaptation strategies, and in a second step to select one of those strategies in an optimal way.

The algorithm for identifying adaptation strategies is represented in Fig. 3. The set of strategies contains the strategies for all the alternatives. For every alternative the following procedure is applied (lines 4-10). For each of the metric to be adapted, we select the set of actions that improve it without negatively affecting other metrics to be adapted (line 9). If this set is empty for some metric, the alternative cannot be

satisfied and an empty result is returned. Otherwise, a Cartesian product of those actions with the actions for other metrics is created (lines 11-13). The resulting set of strategies is returned. For the sake of simplicity we omit here formal proofs of the algorithm correctness.

It is easy to see that any of the strategies extracted in this way will satisfy the identified adaptation requirements. However, the effect of different adaptation strategies on the SBA is not the same. This is because adaptation strategies depending on contained adaptation actions differ in their negative effects on certain applications metrics.

```
1   let S = ∅ // set of resulting strategies
2   for each A_i ∈ R
3    S = S ∪ strategies(A_i)

4   function strategies(A)
5    let S_A = {∅} // set of strategies for A, initially contains an empty set
6    for each (•,r) ∈ A
7     let S' = S_A // temporary set of partial strategies built on previous steps
8     S_A = ∅
9     let act={a | •∈M⁺(a)∧ forall(•',r')∈A,¬(•'∈ M⁻(a))}
10    if act=∅ return ∅ // the whole alternative cannot be achieved
11    for each a ∈ act // built a Cartesian product of actions
12     for each s ∈ S'
13      S_A = S_A ∪ {s ∪ {a}}
14    return S_A
```

Fig. 3. Strategy selection algorithm

Formula 2. To order the strategies we adopt a heuristic, in which the strategy with less negative effects is more preferable. To accomplish this, each strategy $s \in S$ is assigned a natural number ε^- that represents a number of negative effects the actions of the strategy have:

$$\varepsilon^- = \sum_{i=1}^{|s|} |M^-(a_i)|$$

All the strategies are then order according to this number: the lower this number is the more the adaptation strategy is preferred. Note that even if the two actions in the strategy negatively affect the same metric, the effect is counted twice as it may have stronger impact. However, other approaches and heuristics for the selection of an optimal adaptation strategy may be thought of. Some of them are discussed in the following section.

Example 4. Consider the adaptation requirements identified in Example 3. Two metrics should be improved, namely “Delivery Time Supplier” (to the value < 1) and the metric “Delivery Time Shipment” (to the value < 4). Assume also that for those two metrics the following adaptation actions contribute positively:

- For the metric "Delivery Time Shipment": a_{11} re-negotiation of the existing SLA; a_{12} outsource shipment process.
- For the metric "Delivery Time Supplier": a_{21} "find-bind" new supplier; a_{22} substitute with a predefined supplier.

It is easy to see that the possible alternative strategies in this case are:

- a_{11} and a_{21} (i.e., re-negotiate the SLA and find new supplier);
- a_{12} and a_{21} (i.e., outsource shipment and find new supplier);
- a_{11} and a_{22} (i.e., re-negotiate the SLA and substitute supplier);
- a_{12} and a_{22} (i.e., outsource shipment and substitute supplier).

6 Discussion

In the following we will discuss the limitations of our approach and identify possible extensions which are part of our future work.

One of the main limitations of our approach so far is that we do not estimate the effect of single adaptation actions on the overall result of the adaptation strategy. We so far use the constraints on metrics which are given on the branches of the dependency tree as sole adaptation requirements which can lead in some cases to a suboptimal adaptation. This is because the dependency tree is based on historical process instances and does not take possible adaptations and resulting effects into account. If we for example renegotiate the delivery time with the shipper to < 1 (instead of < 2 extracted from the tree) then it is unclear whether the delivery time of supplier has to be adapted at all or to which value. We need thus a possibility to estimate the effect of the adaptation actions on the overall result. Therefore, one could use QoS aggregation techniques based on process structure (e.g., similar to the ones used in [9] or [10]) in order to calculate based on process structure the effects of such a change. Also for estimating the effects, the use of regression trees which show numerical value ranges of KPIs (instead of a simple "green"/"red" classification) would be more helpful in that case.

Another problem which can arise is that the dependency tree does not display the quality factors which can be adapted but other more influential yet unadaptable metrics. As discussed in [2] this problem can be dealt with by either excluding some metrics from analysis or by using drill down techniques. In our case, the problem could be dealt with by using only adaptable potential influential factors during analysis for generating the dependency tree; i.e. the dependency tree in that case would only contain adaptable metrics.

Finally, the identification of adaptation strategies as well as their ordering could be optimized in several ways. First, it is possible to define global SBA constraints as a metric that should not be negatively affected by any of the adaptation actions. If some action may violate such a constraint, it should be excluded. Both the simple metrics and complex KPIs may be used for this purpose. In the latter case it is necessary to take into account also those metrics, on which KPI depends. To accomplish this, the strategy selection algorithm may be extended. Second, it is also possible to associate preferences to the adaptation actions, i.e., explicitly state that one action is more preferable than the other. The ordering of strategies should take this information into

account giving the precedence to the more preferable strategies. Third, if it is possible to capture the effect of the adaptation action onto the metric with a higher precision (e.g., instead of simple positive/negative contribution give a numerical value or even specify the effect of the action on the metric value), then the analysis should give precedence to the actions with better effect. Finally, it is possible also to exploit the relation between the metric and the number of KPI violations. This would allow also for ordering the requirements: the more violations are associated with the metric value, the more important it is. The adaptation actions, therefore, should be selected accordingly.

7 Related Work

The field of QoS-aware adaptable SBAs has only recently been given attention, which is also reflected in the scarce amount of related literature. There are no approaches, to the best of our knowledge, that enable adaptation of SBAs based on quality characteristics yet in an integrated manner across all layers, based on monitoring and analysis of KPIs and the corresponding influential factors.

There are several existing works in the context of QoS-aware service compositions [9, 11] which describe how to create service compositions which conform to global and local QoS constraints taking into account process structure when aggregating QoS values of atomic services. These approaches can be used for QoS-based adaptation by replanning the service composition during monitoring [12]. Our approach is different in that we not only take into account QoS but also quality characteristics from other SBA layers and perform analysis based on their dependencies. We do not (yet) exploit information on process structure during dependency analysis, as the approach described in [10], but use decision tree algorithms instead.

Closely related to our approach is iBOM [13] which is a platform for monitoring and analysis of business processes based on machine learning techniques. It focuses on similar analysis mechanisms as in our approach, but does not deal with adaptation of SBAs by extracting adaptation requirements from the decision trees and automatically deriving adaptation strategies, but uses simulation and what-if analysis techniques instead.

Work on service composition adaptation is available and the existing approaches that do not focus on QoS-awareness of SBAs have been classified. The available approaches are mechanisms for service composition adaptation can similarly be borrowed in the approach presented in this paper as adaptation mechanisms on the service composition level [5, 14].

8 Conclusions and Future Work

In this paper we have presented a novel adaptation approach for SBAs based on quality factor analysis. We have extended previous work on quality factor analysis by showing how the resulting dependency tree can be used for adaptation purposes. In particular we have shown how to model adaptation actions and associate them with

quality metrics, how to extract adaptation requirements from the dependency tree and come up with an adaptation strategy.

Our future work involves extending the approach as discussed in Section 6. In particular, the model of metrics and adaptation actions can be extended by adding more complex descriptions of the effects on the metrics. One could also use adaptation-specific techniques to predict and estimate those effects. We will implement the approach by extending the existing implementation of process quality factor analysis to support the adaptation phase as described in this paper and evaluate the approach experimentally based on a real-world application.

Acknowledgements The research leading to these results has received funding from the European Community's 7th Framework Programme under the Network of Excellence S-Cube Grant Agreement no. 215483.

References

1. Papazoglou, M. P., Traverso, P., Dustdar, S., Leymann, F.: Service-Oriented Computing: State of the Art and Research Challenges. IEEE Computer 11, 2007.
2. Wetzstein, B.; Leitner, P.; Rosenberg, F.; Brandic, I.; Dustdar, S.; Leymann F.: Monitoring and Analyzing Influential Factors of Business Process Performance. In Proceedings of EDOC 2009, Auckland, New Zealand, 2009.
3. Council, S.: Supply Chain Operations Reference Model Version 7.0. 2005.
4. Witten, I.H., Frank, E.: Data Mining: Practical Machine Learning Tools and Techniques. 2nd edition. Morgan Kaufmann, 2005.
5. Karastoyanova, D., Houspanossian, A., Cilia, M., Leymann, F., Buchmann, A.: Extending BPEL for Run Time Adaptability. In Proceedings of EDOC 2005, Enschede, The Netherlands, 2005.
6. Danylevych, Olha; Karastoyanova, Dimka; Leymann, Frank: Optimal Stratification of Transactions. In Proceedings of ICIW 2009, Venice, Italy, 2009.
7. Tinelli, C.: A DPLL-based Calculus for Ground Satisfiability Modulo Theories. In Proceedings of JELIA-02, volume 2424 of LNAI, pages 308{319. Springer, 2002.
8. Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Schulz, S, Sebastiani, R.: An Incremental and Layered Procedure for the Satisfiability of Linear Arithmetic Logic. In Proceedings of TACAS'05, volume 3440 of LNCS. Springer, 2005.
9. Zeng, L., Benatallah, B., Dumas, M., Kalagnamam, J., Chang, H.: QoS-aware Middleware for Web Services Composition. IEEE Trans. on Software Engineering, 30(5), May 2004.
10. Bodenstaff, L., Wombacher, A., Reichert, M., Jaeger, M. C.: Monitoring Dependencies for SLAs: The MoDe4SLA Approach. In Proceedings of SCC 2008. Washington, DC, USA, 2008.
11. Jaeger, M. C., Muhl, G., Golze, S.: QoS-aware Composition of Web Services: An evaluation of selection algorithms. In Proceedings of COOPIS 2005, Cyprus, 2005.
12. Canfora, G., di Penta, M., Esposito, R., Villani, M. L.: QoS-Aware Replanning of Composite Web Services. In Proceedings of ICWS 2005, Orlando, USA, 2005.
13. Castellanos, M., Casati, F., Shan, M.C., Dayal, U.: iBOM: A Platform for Intelligent Business Operation Management. In: Proceedings of the 21st International Conference on Data Engineering (ICDE'05). (2005) 1084-1095
14. Karastoyanova, D., Leymann, F., Buchmann, A.: An Approach to Parameterizing Web Service Flows. In Proceedings of ICSOC 2005, Amsterdam, The Netherlands, 2005.

A.5 Using Soft Constraints to Make QoS-Aware Service Selections

Authors:

UCBL: Mohamed Anis Zemni

UCBL: Salima Benbernou

UPM: Manuel Carro

- Submitted to: ICSOC 2010

A Soft Constraint-Based Approach to QoS-Aware Service Selection *

Mohamed Anis Zemni[1], Salima Benbernou[1], Manuel Carro[2]

[1] LIPADE, Université Paris Descartes, France
[2] Facultad de Informática, Universidad Politécnica de Madrid, Spain
mohamedaniszemni@gmail.com, salima.benbenrou@paridescartes.fr, mcarro@fi.upm.es

Abstract. Service-based systems should be able to dynamically seek replacements for faulty or underperforming services, thus performing self-healing. It may however be the case that available services do not match all requirements, leading the system to grind to a halt. In similar situations it would be better to choose alternative candidates which, while not fulfilling all the constraints, allow the system to proceed. *Soft constraints*, instead of the traditional *crisp* constraints, can help naturally model and solve replacement problems of this sort. In this work we apply soft constraints to model SLAs and to decide how to rebuild compositions which may not satisfy all the requirements, in order not to completely stop running systems.
Keywords: Service Level Agreement, Soft Constraints.

1 Introduction

A (web) service can be defined as a remotely accessible software implementation of a resource, identified by a URL. A set of protocols and standards, such as WSDL, facilitate invocation and information exchange in heterogeneous environments. Software services expose not only functional characteristics, but also non-functional attributes describing their Quality of Service (QoS) such as availability, reputation, etc. Due to the increasing agreement on the implementation and management of the functional aspects of services, interest is shifting towards non-functional attributes describing the QoS. Establishing QoS contracts, described in the Service Level Agreement (SLA), that can be monitored at runtime, is therefore of paramount importance. Various techniques [1] to select services fulfilling functional and non-functional requirements have been explored, some of them based on expressing these requirements as a constraint solving problem [2, 3] (CSP). Traditional CSPs can either be fully solved (when all requirements are satisfied) or not solved at all (some requirements cannot be satisfied). In real-life cases, however, over-constraining is common (e.g., because available services offer a quality below that required by the composition), and problems are likely not to have a classical, crisp solution. Solving techniques for *soft* CSPs (SCSP) [4–6] can generate solutions for overconstrained problems by allowing some constraints to remain unsatisfied.

* The research leading to these results has received funds from the European Community's Seventh Framework Programme FP7/2007-20013 under grant agreement 215483 (S-CUBE). Manuel Carro was also partially supported by Spanish MEC project 2008-05624/TIN *DOVES* and CM project P2009/TIC/1465 (*PROMETIDOS*).

A C-semiring is a tuple $\langle A, +, \times, \mathbf{0}, \mathbf{1} \rangle$ s.t.

- A is a set and $\mathbf{0} \in A$, $\mathbf{1} \in A$.
- $\sum$ (the *additive* operation)[a] is defined on subsets of A as follows:
  - $+$ is commutative ($a + b = b + a$), associative ($a + (b + c) = (a + b) + c$), with unit element $\mathbf{0}$ ($a + \mathbf{0} = a$) and absorbing element $\mathbf{1}$ ($a + \mathbf{1} = \mathbf{1}$).
  - $\sum \emptyset = \mathbf{0}$ and for all $a \in A, \sum\{a\} = a$.
  - Given any set of indices S, $\sum_{i \in S}(\bigcup A_i) = \sum(\{\sum_{i \in S} A_i\})$ (flattening).
- $\times$ (the *multiplicative* operation) is associative, commutative, $a \times \mathbf{1} = a$ and $a \times \mathbf{0} = \mathbf{0}$.
- $\times$ distributes over $+$, i.e., $a \times (b + c) = (a \times b) + (a \times c)$.

[a] Written as infix $+$ when applied to a two-element set.

Fig. 1. Definition of a C-Semiring for Soft Constraints.

Our framework takes into consideration the penalties agreed upon on the SLA by building a new (Soft) Service Level Agreement (SSLA) based on preferences where strict customer requirements are replaced by soft requirements allowing a suitable composition. This agreement has to include penalty terms to be applied while the contract terms are violated.

2 Soft Constraints in a Nutshell

A CSP defines a set of variables whose ranges we assume a finite domain (FD)[3] and a set of constraints which restrict the values these variables can take. A solution for a CSP is an assignment of a value to *every* variable s.t. all the constraints are simultaneously satisfied. Soft constraints [5, 6] generalize classical CSPs by adding a preference level to every tuple in the domain of the constraint variables. This level can be used to obtain a suitable solution which may not fulfill all constraints, which optimizes some metrics, and which in our case will be naturally applied to the requirements of the users.

The basic operations on soft constraints (building a constraint conjunctions and projecting on variables) need to handle preferences in a homogeneous way. This requires the underlying mathematical structure of classical CSPs to change from a cylindrical algebra to a semiring algebra, enriched with additional properties and termed a *C-semiring* (Figure 1). In it, A provides the levels of preference of the solutions and it can be proved that it is a lattice with partial order $a \preceq b$ iff $a + b = b$, minimum $\mathbf{0}$, and maximum $\mathbf{1}$. When solutions are combined or compared, preferences are accordingly managed using the operations $\times$ and $+$. Note that the theory makes no assumptions as to what the preferences mean, or how they are actually handled: $\times$ and $+$ are placeholders for concrete definitions which can give rise to different constraint systems, such as fuzzy constraints, traditional constraints, etc.

Figure 2 summarizes some basic definitions regarding soft constraints. A constraint takes a tuple of variables and assigns it a tuple of concrete values in the domain of the

[3] CSPs can be defined on infinite domains, but assume a FD here because it can accommodate many real-life problems, as witnessed by the relevance of FD in industrial applications, and because soft constraint theory requires finiteness.

Definition 1 (Constraint). *Given a c-semiring $\langle A, +, \times, 0, 1\rangle$, a set of variables V, and a set of domains D, one for every variable in V, a constraint is the pair $\langle def, con\rangle$ where $con \subseteq V$ and $def : D^{|con|} \rightarrow A$.*

Definition 2 (Soft Constraint Satisfaction Problem SCSP). *A SCSP is a pair $\langle C, con\rangle$ where $con \subseteq V$ and C is a set of constraints. C may contain variables which are not in con, i.e., they are not interesting for the final result. In this case the constraints in C have to be projected onto the variables in con.*

Definition 3 (Constraint combination). *Two constraints $c_1\langle def_1, con_1\rangle$ and $c_2 = \langle def_2, con_2\rangle$ can be combined in $c_1 \otimes c_2 = \langle def, con\rangle$ by taking all the variables in the original constraints ($con = con_1 \bigcup con_2$) and assigning to every tuple in the new constraint a preference value which comes from combining the values in the original constraints: $def(t) = def_1(t \downarrow_{con_1}^{con}) \times def_2(t \downarrow_{con_2}^{con})$, with $t \downarrow_Y^X$ denoting the projection of tuple t, which is defined on the set of variables X, over the set of variables $Y \subseteq X$.*

Definition 4 (Projection). *Given a soft constraint $c = \langle def, con\rangle$ and a set of variables $I \subseteq V$, the projection of c over I, denoted by $c \Downarrow_I$ is the constraint $\langle def', con'\rangle$ where $con' = con \bigcap I$ and $def'(t') = \sum_{\{t | t\downarrow_{con\cap I}^{con} = t'\}} def(t)$.*

Definition 5 (Solution). *A solution of a SCSP problem $\langle C, con\rangle$ is the constraint $(\otimes C) \Downarrow_{con}$, i.e., the combination (conjunction) of all the constraints in C projected over all the variables con of interest.*

Fig. 2. Definitions for Soft Constraints.

variables, plus a preference value (belonging to the set A). Constraints can be combined into other constraints (with $\otimes$, similar to conjunction) and projected ($\Downarrow_Y^X$) onto a tuple of variables. The preference value of every tuple in a constraint conjunction is worked out by applying $\times$ to the preference values of the tuples in the individual constraints. Projections eliminate "columns" from tuples and retain only the non-removed tuple components. Repeated rows may then appear, but only one is retained, and its preference is calculated applying $+$ to the preferences of the repeated tuples. Since a solution is a projection on some selected set of variables, the preferences of solutions are naturally calculated using the projection operation. Usually the tuple with the highest preference value is selected as the "optimal" solution.

3 Soft Service Level Agreement and SCSPs

A Service Level Agreement (SLA) [7] is a contract between provider(s) and client(s) specifying the guarantees of a service, the expected quality level, and the penalties to be applied in case of unfulfillment of duties, and it is therefore an important quality management artifact. The SLA can be used to identify the responsible of a malfunction and to decide which action (if any) has to be taken. An SLA should, therefore, be realistic, achievable, and maintainable.

An SLA has a rich structure from which we underline the properties of the services, including those measurable aimed at expressing guarantees. This part provides a set Υ of variables υ_i (whose meaning is explained in the service description) and their domains $\delta_i \in \Delta$, which can be established by the metric attribute. A Soft SLA (SSLA) is similar to a SLA but with the addition of a set of user preferences and of penalties

Definition 6 (Preference). *The set* $Pr = \{\langle \delta_i, \upsilon_i, a_i \rangle | \delta_i \in \Delta, \upsilon_i \in \Upsilon, a_i \in A\}$ *where* δ_i *is the sub-domain that the* i*-th preference belongs to,* υ_i *is the variable defining the preferences, and* a_i *is semiring value, representing the preferences in an SSLA.*

Definition 7 (Penalty). *The set* $Pn = \{pn_i \mid \exists pr_i \; s.t. \; \upsilon_i \notin \delta_i\}$ *represents the penalties.*

Definition 8 (SSLA document). *A SSLA document is a tuple* $\zeta = \langle \Upsilon, \Delta, A, Pr, Pn, T \rangle$ *where* Υ *is a set of variables* υ_i*,* Δ *is a set of variable domains* δ_i *(one for each variable), Pr is a set of preferences* Pr_i*,* Pn *is a set of penalties* Pn_i *to apply when the preferences are not satisfied and* T *is a set of pairs* $\langle pr_i, pn_i \rangle$ *which associates preferences with the penalties to apply in case of violation.*

Fig. 3. Definitions related to a soft SLA.

associated to contract breaking (respectively, Pr and Pn). The preferences are used to make a composition in the presence of unsatisfied requirements and the penalties are used to refine found solutions and to protect each party from the violation of the contract terms. These notions are depicted in Figure 3. The i-th penalty $pn_i \in Pn$ is applied when the i-th preference $pr_i \in Pr$ is not satisfied.

3.1 Extending SCSP Using Penalties

We will adapt the SCSP framework to handle explicitly penalties for service selection and to build a Soft Service Level Agreement including preferences and penalties. In this framework, service selection has three phases:

1. Model the characteristics for the selection using soft constraints.
2. Assuming a pre-selection is made using functional requirements, rank candidate services using non-functional requirements and the constraint preferences.
3. We assign penalties to unmet user preferences, and these penalties are used to rank solutions having the same constraint preferences.

Figure 4 shows the definitions for this extended SCSP framework. We extend the application of semiring operations to penalties. Variables are assumed to take values over subdomains which discretize a continuous domain, and which for brevity we represent using identifiers in $D_{\{\}}$. The constraint preference function *def* is also adapted in order to apply it both to preferences and to penalties. The projection operation is kept as in the SCSP framework.

3.2 An Example

A delivery service has an order-tracking web service. Companies wishing to hire this service want to have in the contract non-functional criteria such as availability, reputation, response time and cost.

Phase 1 Let $CS = \langle S_p, D_{\{\}}, V \rangle$ be a constraint system and $P = \langle C, con \rangle$ be the problem to be solved, where $V = con = \{\underline{\text{A}}\text{vailability}, \underline{\text{R}}\text{eputation, response } \underline{\text{T}}\text{ime, co}\underline{\text{S}}\text{t}\}$, $D_{\{\}} = \{\{a_1, a_2\}, \{r_1, r_2\}, \{t_1, t_2, t_3\}, \{s_1, s_2\}\}$, $S_p = \langle [0, 1], Pn, \max, \min, 0, 1 \rangle$, $C = \{c_1, c_2, c_3, c_4\}$. For simplicity, variables and their domains have been written in the same order.

Definition 9 (CP-semiring). *A CP-semiring is a tuple $S = \langle A, Pn, +, \times, 0, 1\rangle$, extending a C-semiring. A and Pn are two sets with lattice structure stating preference values for solutions and penalties. Operations $\times$ and $+$ are applied when constraints are combined or projected.*

Definition 10 (Constraint System). *A constraint system is a tuple $CS = \langle S, D_{\{\}}, V\rangle$, where S is c-semiring, $D_{\{\}}$ represents the set of identifiers of subdomains, and V is the ordered set of variables.*

Definition 11 (Constraint). *Given a constraint system $CS = \langle S_p, D_{\{\}}, V\rangle$ and a problem $P = \langle C, con\rangle$, a constraint is the tuple $c = \langle def_c, type\rangle$, where type represents the type of constraint and def_c is the definition function of the constraint, which returns the tuple*

$$def : D_{\{\}}^{|con|} \rightarrow \langle p_r, p_n\rangle,$$

Definition 12 (Soft Constraint Satisfaction Problem SCSP). *Given a constraint system $CS = \langle S, D_{\{\}}, V\rangle$, an SCSP over CS is a pair $P = \langle C, con\rangle$, where con, called set of variables of interest for C, is a subset of V and C is a finite set of constraints, which may contain some constraints defined on variables not in con.*

Fig. 4. CP-Semiring.

A set of penalties, ranked from the most to then less important one, has been set: $pn_i \preceq pn_j$ if $i \leq j$. The above shown values of the variable domains comes from a discretization such as $availability \in \{[0, 0.5[, [0.5, 1]\}$, $reputation \in \{[0, 0.6[, [0.6, 1]\}$, $response\ time \in \{[20, \infty[, [5, 20[, [0, 5[\}$, $cost \in \{[1000, 1500[, [1500, 3000]\}$.

Let us consider the following constraints: $c_1 = \langle def_{c1}, \{\text{availability, reputation}\}\rangle$, $c_2 = \langle def_{c2}, \{\text{response time}\}\rangle$, $c_3 = \langle def_{c3}, \{\text{availability, reputation, cost}\}\rangle$, $c_4 = \langle def_{c4}, \{\text{reputation, response time}\}\rangle$, where the preference values and corresponding penalties are in Table 1. For example, for the tuple $\langle a_2, r_1\rangle$, attributes "availability" and "reputation" are respectively assigned subdomains $[0.5, 1]$ and $[0, 0.6[$. The function $def_{c1}(\langle a_2, r_1\rangle) = \langle 0.5, pn_3\rangle$ shows that these attribute values have a preference 0.5 and company is ready to sign away this preference for a penalty pn_3.

Phase 2 Given the model, we define constraint combination to keep the minimum value of preferences (resp. for the penalties). For example $def_{c1}(\langle a_2, r_1\rangle) \otimes def_{c2}(\langle t_3\rangle) = \min(\langle 0.5, pn_3\rangle, \langle 0.25, pn_6\rangle) = \langle 0.25, pn_3\rangle$ and so on with all the tuples to obtain $c_{1,2}$. Next, we would combine $c_{1,2}$ and c_3 to get $c_{1,2,3} = c_{1,2} \otimes c_3$ and so on, until all constraints have been combined. Table 2 shows the results of combining all the constraints.

$\langle A, R\rangle$	def_{c1}	$\langle T\rangle$	def_{c2}	$\langle A, R, S\rangle$	def_{c3}	$\langle R, T\rangle$	def_{c4}
$\langle a_1, r_1\rangle$	$\langle 0, -\rangle$	$\langle t_1\rangle$	$\langle 0.25, pn_6\rangle$	$\langle a_1, r_1, s_1\rangle$	$\langle 0.25, pn_8\rangle$	$\langle r_1, t_1\rangle$	$\langle 0.5, pn_6\rangle$
$\langle a_1, r_2\rangle$	$\langle 0.25, pn_1\rangle$	$\langle t_2\rangle$	$\langle 0.5, pn_5\rangle$	$\langle a_1, r_1, s_2\rangle$	$\langle 0.25, pn_1\rangle$	$\langle r_1, t_2\rangle$	$\langle 0.5, pn_5\rangle$
$\langle a_2, r_1\rangle$	$\langle 0.5, pn_3\rangle$	$\langle t_3\rangle$	$\langle 1, pn_7\rangle$	$\langle a_1, r_2, s_1\rangle$	$\langle 0.5, pn_1\rangle$	$\langle r_1, t_3\rangle$	$\langle 0, -\rangle$
$\langle a_2, r_2\rangle$	$\langle 0.75, pn_3\rangle$			$\langle a_1, r_2, s_2\rangle$	$\langle 0.25, pn_3\rangle$	$\langle r_2, t_1\rangle$	$\langle 0.75, pn_2\rangle$
				$\langle a_2, r_1, s_1\rangle$	$\langle 0.75, pn_9\rangle$	$\langle r_2, t_2\rangle$	$\langle 0.75, pn_4\rangle$
				$\langle a_2, r_1, s_2\rangle$	$\langle 0.5, pn_8\rangle$	$\langle r_2, t_3\rangle$	$\langle 1, pn_2\rangle$
				$\langle a_2, r_2, s_1\rangle$	$\langle 0.75, pn_2\rangle$		
				$\langle a_2, r_2, s_2\rangle$	$\langle 0.25, pn_1\rangle$		

Table 1. Constraint definitions.

$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$	$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$	$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$	$\langle A,R,T,S\rangle$	$\langle pr,pn\rangle$
$\langle 2,2,3,1\rangle$	$\langle 0.75,pn_2\rangle$	$\langle 2,2,1,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,2,1,1\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,1,3,2\rangle$	$\langle 0.0,-\rangle$
$\langle 2,2,2,1\rangle$	$\langle 0.50,pn_2\rangle$	$\langle 1,2,3,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,2,1,1\rangle$	$\langle 0.25,pn_2\rangle$	$\langle 1,1,3,1\rangle$	$\langle 0.0,-\rangle$
$\langle 2,1,2,2\rangle$	$\langle 0.50,pn_3\rangle$	$\langle 1,2,3,1\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,1,2\rangle$	$\langle 0.25,pn_3\rangle$	$\langle 1,1,2,2\rangle$	$\langle 0.0,-\rangle$
$\langle 2,1,2,1\rangle$	$\langle 0.50,pn_3\rangle$	$\langle 1,2,2,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,1,1\rangle$	$\langle 0.25,pn_3\rangle$	$\langle 1,1,2,1\rangle$	$\langle 0.0,-\rangle$
$\langle 2,2,2,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,2,2,1\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,3,2\rangle$	$\langle 0.0,-\rangle$	$\langle 1,1,1,2\rangle$	$\langle 0.0,-\rangle$
$\langle 2,2,3,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 1,2,1,2\rangle$	$\langle 0.25,pn_1\rangle$	$\langle 2,1,3,1\rangle$	$\langle 0.0,-\rangle$	$\langle 1,1,1,1\rangle$	$\langle 0.0,-\rangle$

Table 2. Ordered constraint combinations with preferences and penalties.

Phase 3 The set of solutions is ranked by preferences and then by penalties (already in Table 2). The solution with highest rank is chosen first. If it turns out not to be feasible, the associated penalty is applied and the next solution is chosen, and so on.

3.3 Mapping SSLA onto SCSP Solvers

Given the our design of an SSLA, mapping it into a SCSP is very easy: variables v_i in the SSLA are mapped onto the corresponding v_i in the SCSP; SSLA domains δ_i are discretized and every discrete identifier is a domain for a SCSP variable; and preferences and penalties (both lattices) are handled together by the def function, so they can be mapped to the A set in a C-semiring with an adequate definition of the def function.

4 Conclusion

We have presented a soft constraint-based framework to seamlessly express QoS properties reflecting both customer preferences and penalties applied to unfitting situations. The application of soft constraints makes it possible to work around overconstrained problems and offer a feasible solution. Our approach makes easier this activity thanks to ranked choices. Introducing the concept of penalty in the Classical SCSP can also be useful during the finding and matching process. We plan to extend this framework to also deal with behavioral penalties.

References

1. Carlos Müller, Antonio Ruiz-Cortés, and Manuel Resinas. An Initial Approach to Explaining SLA Inconsistencies. In Athman Bouguettaya, Ingolf Krueger, and Tiziana Margaria, editors, *Service-Oriented Computing (ICSOC 2008)*, volume 5364 of *LNCS*, pages 394–406, 2008.
2. Ugo Montanari. Networks of Constraints: Fundamental Properties and Application to Picture Processing. *Information Sciences 7*, pages 95 – 132, 1974.
3. Rina Dechter. *Constraint Processing*. Morgan Kaufman, 2003.
4. S. Bistarelli. *Semirings for Soft Constraint Solving and Programming*. Springer, 2004.
5. Stefano Bistarelli, Ugo Montanari, and Francesca Rossi. Semiring-based constraint satisfaction and optimization. *J. ACM*, 44(2):201–236, 1997.
6. S. Bistarelli, U. Montanari, and F. Rossi. Constraint Solving over Semirings. *In Proc. IJCAI95*, 1995.
7. Philip Bianco, Grace A.Lewis, and Paulo Merson. Service Level Agreements in Service-Oriented Architecture Environment. Technical Report CMU/SEI-2008-TN-021, Carnegie Mellon, September 2008.

A.6 Ad-hoc Management Capabilities for Distributed Business Processes

Authors:

UNIHH: Sonja Zaplata

UNIHH: Dirk Bade

UNIHH: Kristof Hamann

UNIHH: Winfried Lamersdorf

CoreMedia AG: Daniel Strassenburg

Geoflags GmbH: Benjamin Wunderlich

- Submitted to: BPSC 2010

Ad-hoc Management Capabilities for Distributed Business Processes *

Sonja Zaplata, Dirk Bade, Kristof Hamann, Winfried Lamersdorf
Distributed Systems and Information Systems
Computer Science Department, University of Hamburg, Germany
{zaplata|bade|hamann|lamersdorf}@informatik.uni-hamburg.de

Daniel Straßenburg[1], Benjamin Wunderlich[2]
[1]CoreMedia AG, [2]Geoflags GmbH, Hamburg, Germany
[1]Daniel.Strassenburg@coremedia.com, [2]benjamin@geoflags.de

Abstract: Advanced business processes are mostly distributed and require highly flexible management capabilities. In such scenarios, process parts often leave their initiator's direct sphere of influence – while management requires both *monitoring* as well as *instant reaction capabilities* anytime during the overall execution of the process. However, realizing such functions is often difficult, e.g. due to the heterogeneity and temporal disconnectivity of some participating execution systems.

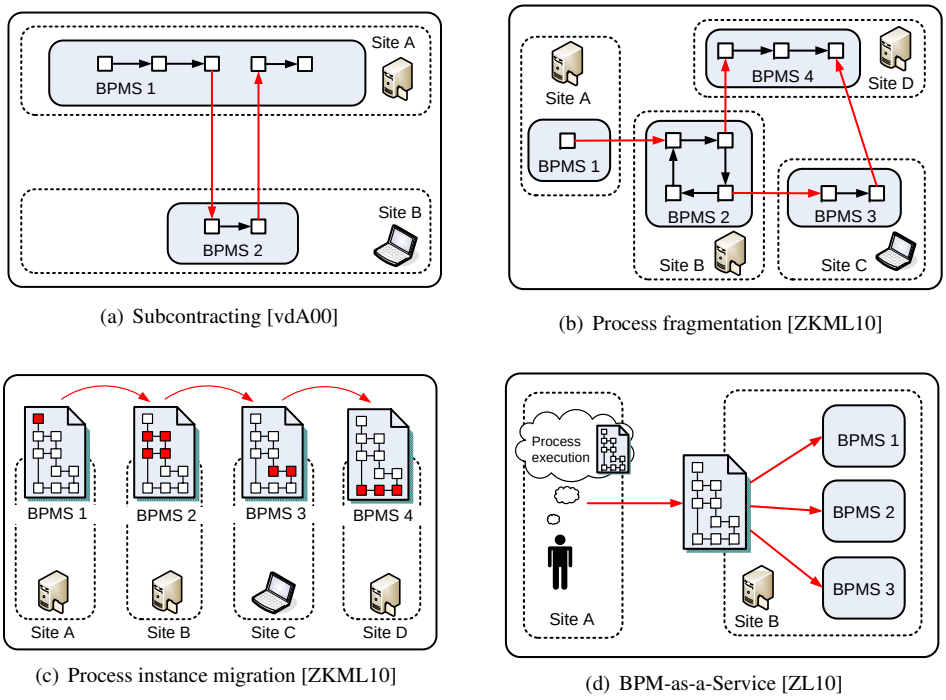
Therefore, this contribution proposes a two-tier concept for monitoring and controlling distributed processes by representing a process management system as a *manageable resource* according to the Web Service Distributed Management (WSDM) standard. Based on a minimal shared model of management capabilities it allows to define customized events and processing rules for influencing business processes executed on a remote (and even on a temporarily disconnected) process management system. Applicability is demonstrated by a scenario-based evaluation on distributed WS-BPEL and XPDL processes and is also tested in the specific context of mobile process management.

1 Motivation

Today's competitive business collaborations highly benefit from transparency and visibility of the status of their private business process networks because these most often relate best to the key performance indicators (KPIs) of the participating organizations. Within a single organization, *business activity monitoring (BAM)* technologies support real-time analytics about running business transactions and allow for the correlation of events for causalities, aggregates, thresholds, and alerts based on user-defined preferences. The analyzed information is delivered in (near) real time and provides an important basis to detect failures and non-compliances, to react to them accordingly and in sufficient time and, thus, to optimize the execution of such intra-organizational business processes in whole or in part.

However, to stay competitive and provide new value-added products and services, often also *cross-organizational collaborations* become necessary which span business processes

*The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).



(a) Subcontracting [vdA00]

(b) Process fragmentation [ZKML10]

(c) Process instance migration [ZKML10]

(d) BPM-as-a-Service [ZL10]

Figure 1: Examples for the distribution of process execution

between several organizations and different process management systems. Thereby, not only atomic resources such as employees, machines and services, but also the execution of a process instance itself can be distributed. Figure 1 shows examples for such distribution, realized as *subcontracting* of single process parts (e.g. to execute a subset of the process in a different location), *fragmentation* of processes (e.g. due to non-functional aspects such as execution time, performance, navigation cost and capacity utilization), *process instance migration* from one workplace to another (e.g. if required services and resources cannot be accessed from a centralized system because of technological differences or due to security policies) and *BPM-as-a-Service*, where the execution of a business process is fully operated by an external provider [vdA00, ZKML10, ZL10].

Resulting *cross-organizational processes* are often highly distributed and dynamic. However, respective process owners are still interested in monitoring and controlling the entire process – even when it is partly executed on another system. Beyond that, it is even more relevant to gather information about the execution on the remote system, because such participants may be dynamically selected or exchanged on the basis of their workload, context or QoS parameters. Cross-organizational monitoring and controlling capabilities can thus support the controllability of active process parts and – based on the collected information and experiences – also optimize the distribution and execution of upcoming processes.

As a current drawback, today's BPM systems mostly consider monitoring and controlling of single centralized process executions, are often heterogeneous and do not provide standardized runtime monitoring or management APIs [vLLM$^+$08]. Therefore, an integration of runtime monitoring information from different source systems is hardly possible yet. Required possibilities to also take influence on a remote process execution and

to react to the observed behavior of the process (preferably in real time) are still challenging. This paper therefore aims at a concept and supporting infrastructure to flexibly collect information about the execution of process parts running on a remote system, to automatically process this information and to predefine and execute timely reactions to detected complex situations where ever necessary. The approach presented here proposes a *service-based common management interface* and uses *complex event processing* in order to specify user-defined management rules and actions.

The rest of this paper is organized as follows: Section 2 motivates the need for a cross-organizational management approach for distributed business processes based on existing and related work. Section 3 presents a two-tier middleware extension for process management systems in order to support the provision and utilization of ad-hoc management capabilities. Section 4 analyzes applicability of the proposed concepts based on a use case and Section 5 concludes the paper.

2 Background and Related Work

Distributed and decentralized process execution becomes increasingly important and, consequently, many such approaches demonstrate the relevance of this research (cp. [J$^+$01] for a brief overview). Research in the area of monitoring constitutes an important part of the management of such distributed processes. Relevant previous approaches use the idea of extending existing business process models by weaving in additional activities which call-back to a central monitoring system (e.g. [BGG04, BG05]). Thereby, the user initiating the process is free to build up his own monitoring system according to his individual preferences, e.g. accessing the status of the process instances, the duration of activities or the actual navigation of control flow. As another advantage, neither agreements nor the adaptation of the partner system are required. Nevertheless, many information interesting for distribution, such as current system properties (e.g. the location where the process is executed, or the current workload of the engine), the number and type of deployed (but uninitiated) process models or the occurrence of internal process instance events (e.g. errors) are not visible. Furthermore, for a consistent overall monitoring additional activities have to be inserted after each functional task. This may result in a huge monitoring overhead, in the worst case expanding the original process description up to its double size, potentially decreasing the performance of process execution considerably and mixing business logic and technical management logic in an undesired way (cp. also [MWL08]). Finally, appropriate actions depending on the results of the gathered information are limited, because running process activities cannot be influenced, e.g. canceled.

In order to preserve efficiency of process execution while at the same time allow quick and adequate reactions to predefined situations, the subscription to *process-related events* and their corresponding processing are attached a high importance [vA09]. Such events can be divided into primitive and complex events. While a primitive event simply represents some relevant change of a certain property (e.g. change of a process's status, a workload shift, etc.) complex events represent some arbitrarily complex inference of information from one or more primitive or other complex events [Luc02]. This is achieved by so called *Complex Event Processing (CEP)* and *Event Stream Processing* techniques. As an example, the ESPER project [1] addresses business process management and automation,

[1] http://esper.codehaus.org/

i.e. process monitoring, BAM, reporting exceptions and operational intelligence. It uses an SQL-based query language to express rules and provides a rule engine for complex event processing. In order to address the heterogeneity of possible event sources, common agreements and standards for the representation of events (e.g. IBM's *Common Base Event* [IBM04]) as well as for the specification of complex event inference statements are required. Regarding the latter aspect, several *Event Query Languages* have already been proposed (cp. [Bui08]) and most of them are based on proprietary extensions of SQL. But events and query languages are only one part of standardization requirements. The specification of reactions to (complex) events is equally important. Rule-based approaches, especially *Event-Condition-Action* rules, are widely used but due to manifold application domains neither the event, nor the condition or action representations are commonly agreed on. Wetzstein et al. [WKK+10] therefore present an approach to support monitoring in service choreographies based on agreements about events to be shared with other partners and using complex event processing to derive key performance indicators for the overall process execution. However, this approach still only focuses on the subscription to events, but neither offers the possibility for requesting monitoring information on demand nor for initiating ad-hoc management actions.

As a foundation for such interoperability of heterogeneous workflow management systems, the Workflow Management Coalition (WfMC) has issued the *Workflow Reference Model*. Accordingly, the reference model also contains administration and monitoring tools for the management of users, resources and processes [WfM98]. A short overview of management operations is proposed here, especially for user and role management (e.g. changing privileges of users), audit management (e.g. querying logs and audit trails), resource control (e.g. concurrency levels, thresholds), process supervisory functions (e.g. termination of process instances) and process status functions (e.g. fetching information about process instances). Associated specifications for achieving workflow interoperability (e.g. Wf-XML [SPG04]) are more detailed, but still focus on sending, installing and retrieving process definitions to/from a remote process engine. However, the Wf-XML idea of exchanging process management related information based on a common model by using standard web services is, in general, very interesting.

The *Web Services Distributed Management (WSDM)* standard develops this idea a bit further. It allows to specify an arbitrary resource (e.g. a printer) as a so-called *manageable resource* which offers a set of resource-dependent properties accessible by a self-describing service interface [OAS06b]. Providing such resource properties requires to specify a model as a mutual understanding of the resource to be managed. However, only a model supporting the management of web services *(MOWS)* [OAS06a] themselves has been developed. The first part of the work presented here therefore proposes a model to exchange basic information and control options for business process management systems involved in cross-organizational collaborations. A similar basis has been proposed by van Lessen et al. [vLLM+08] for WS-BPEL process instances. In this paper, however, we are extending this idea by also including relevant process model and process engine properties as well as related events, and, as the second part of this work, presenting a loosely coupled management component in order to analyze and process the received information either on-site or remotely. As collaboration already implies a common ground, a certain amount of trust between the participating partners can be assumed. It is therefore assumed that collaboration partners do not intentionally provide false information, as such malicious behavior would be detected later anyhow.

3 A Two-Tier Process Management Middleware

Prerequisite of the approach presented here is to consider a business process management system as a *manageable resource* according to the understanding of WSDM. Defining the elements and properties of this manageable resource, relevant functionalities such as data retrieval, event subscription and control options can be exposed as services and can be integrated in a standard registry and thus in existing and future applications. Therefore, a minimal shared understanding of business process management, especially of process models, process instances and process engines is required. The next section (Section 3.1) proposes such an abstract model of a business process management system in the special context of distributed processes. Based on that, Section 3.2 presents a component using the resulting management services and events in order to specify user-defined monitoring and management (re-)actions. Implementation within the framework of WSDM is shown in Section 3.3.

3.1 Tier 1: Process Management System as a Manageable Resource

In order to find an adequate basis for a common understanding of the elements and attributes relevant for distributed process management, an analysis of several current practical and theoretical approaches and systems as well as abstract models and concrete products for traditional and distributed business process management has been carried out. The analysis included the *Workflow Reference Model* [WfM98], *Workflow-Petrinets* [vdA98] and general *Workflow Patterns* [VDATHKB03] as abstract concepts; the *Business Process Modelling Notation (BPMN)* [OMG09], *XML Process Definition Language (XPDL)* and *Business Process Execution Language for Web Services (WS-BPEL)*[OAS07] as ways to describe a process; and *ActiveBPEL* [2] and *Apache ODE* [3] as traditional and *DEMAC* [ZKL09] as mobile process execution and management systems. The analysis lead to the identification of most relevant management entities and a resulting basic model which is shown in Figure 2. It holds the process management system as the manageable resource which can be accessed by a service-based management interface either by pulling read-only information about its entities (*information interface*), by asking for manipulation of entity values (*modification interface*) or for receiving events emitted by the entities (*event interface*). To provide information not only about the functional manageable resource, but also about its management, a *meta interface* allows to access information about management capabilities as well as operations for the configuration of the management interface, e.g. to configure subscriptions for events.

In the context of distributed process management, the proposed entities of a process management system include (but are not limited to) the *process models* which are deployed to the process engine, the *process instances* which are instantiations of these models (representing the processes which are currently running), and the *process histories* which contain information about processes which have already been finished (cp. Figure 2). Furthermore, to consider the special characteristics of distributed process management (such as mobility, cooperation and dynamic assignment) the process management system has a relevant *context* comprised of the *intrinsic context* of the process engine (e.g. system properties such as workload or service availability), and the *extrinsic context* (e.g. location or weather).

[2] http://www.activevos.com/community-open-source.php
[3] http://ode.apache.org/

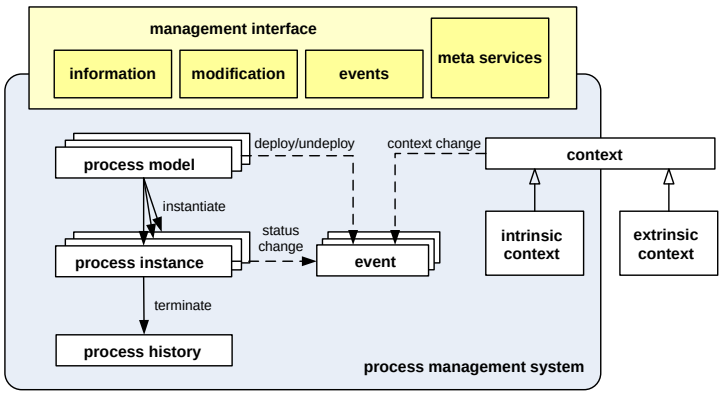


Figure 2: Process management system as manageable resource

Following existing research works in the area of context models (e.g. [SP04, HA08]), both types of context can either be static (e.g. the identity of the owner of the system) or dynamic (e.g. the current workload). Generic context models which can be customized for such application are e.g. proposed by [KZTL08].

Figures 3 and 4 show the refinement of the entities *process model* and *process instance* as manageable resources. In particular based on the Workflow Reference Model [WfM98] and XPDL [NM02], a process consists of *activities* which are connected by *transitions* to define a control flow (potentially restricted by a *condition*), and a set of *data fields* using a certain *data type* also defined within the process model. Furthermore, participants can be predefined as required *performers* for specific activities (such as the *swimlane* concept in BPMN [OMG09]) which is especially important in the context of distributed process management as this construct contributes to the selection of partner systems. Although, theoretically, all these entities can be reused in other processes (e.g. a data type), it is important to notice that they belong to only one process here, i.e. changing them will only affect this single process model. Besides, in order to enable a distributed execution and a respective management, both model and instance need to have a unique identifier for the correlation of requests.

A process instance (cp. Figure 4) extends its process model by implementing the associated runtime information. Most importantly, this involves the *status* of the running process (e.g. executing, suspended, in error [SPG04]), the specific *values of the data fields*, the *status* of the activities (e.g. running, skipped [SPG04]), the *evaluation* of transition conditions (true/false) and the *actual performers* who are finally executing the activities. The latter is interesting in case no performer is specified within the model, or for the case of deviances, e.g. if the specified participant has again subcontracted a part of the assigned control flow to another process management system. Finally, *process histories* reflect the entities of the terminated process instances in a static way (not depicted).

According to existing approaches such as Wf-XML [SPG04], all entities contain a number of sub entities and individual atomic properties, e.g. a process engine has a current workload expressed as the number of running process instances and CPU load, or a process instance activity has a start time, a duration and an end time. Creation of entity instances and changes of their properties' values are effecting the associated *events*. In order to allow manageability, exchanging information about a resource property requires an uniform and unambiguous representation and interpretation of values, e.g. represented as standard

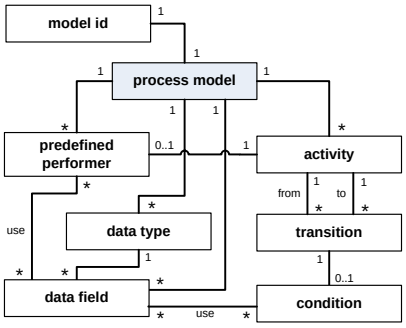


Figure 3: Process model as manageable resource

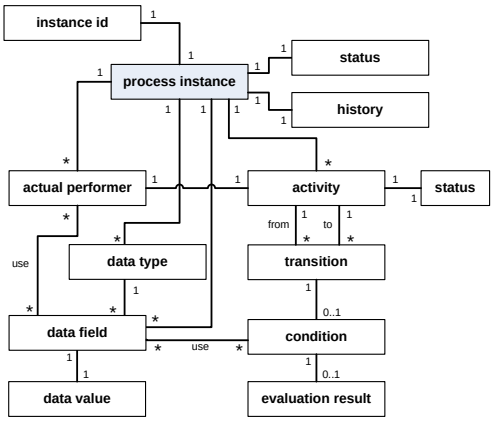


Figure 4: Process instance as manageable resource

or complex data types, and a metric. Furthermore, the *modifiability* (e.g. read or read-write), the *availability* of the property (e.g. before, during or after execution of the process or the activity) and the *mutability* and *frequency of updates* should be specified. Due to space limitations and similarity to existing meta models, the enumeration of relevant entity properties and events should, however, not be part of this paper.

3.2 Tier 2: Management Component for Complex Event Processing

Providing informational and manipulative services and the possibility to subscribe to events based on a common understanding such as established in Section 3.1, arbitrary management applications can be composed in order to collect information and react to even complex situations in a user-defined way. In this section, we present a loosely-coupled management component to support such operations.

The general methodology of the approach is depicted in Figure 5: The user who is initiating a controlled distributed execution of a process (in the following called the *customer*) takes the original process description to be executed and creates an additional document (*management document*) which holds the user's requirements for the management of this process (*management rules*). Here, the term *management* subsumes all objects, situations and operations which are, from the customer's perspective, relevant for the correct execution and administration of the distributed process and are not covered by the functional business process description. Relevant objects are the entities of the model presented in Section 3.1, e.g. process models, instances and data objects. Situations and operations are described within the management document as complex situations and actions. An example for such a situation-action pair is monitoring the duration of executing a specific activity (*object*) and, in case a specified amount of time has passed and no progress becomes visible (*situation*), to restart the activity (*action*). However, also monitoring rules that do not influence the execution of the process are possible (e.g. after each activity, its performer, duration and current location should be logged) or distribution decisions and actions can be supported (e.g. if the workload exceeds a specified threshold, the process should be transferred to a process engine with a better capacity).

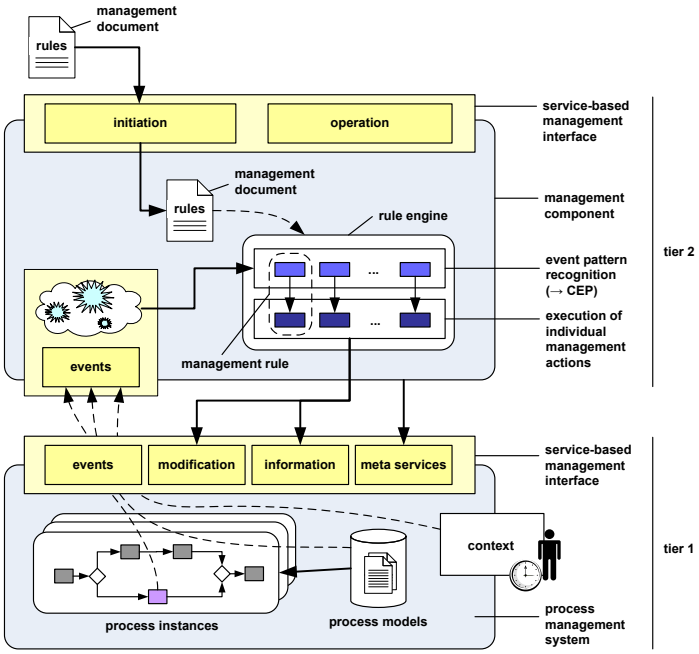


Figure 5: Management component to support customized management actions

Listing 1 shows the general syntax of a management rule as a part of the management document. For administration purposes, each management rule has a *name* and, optionally, a *description*. The *rule pattern* holds an event pattern to determine if a complex situation has occurred or not. The *rule action* specifies the service to be executed, including parameters for the service call if necessary. Encapsulated as a composite service, even complex actions can be defined. Furthermore, arbitrary system-external services, such as sending an email, can be referenced.

```
1 MANAGEMENT-RULES
2   MANAGEMENT-RULE
3     NAME          : <String>
4     [DESCRIPTION  : <String>]
5     RULE-PATTERN  : <Event-Pattern>
6     RULE-ACTION   : <Service-Invocation>
7   END MANAGEMENT-RULE
8   ...
9 END MANAGEMENT-RULES
```

Listing 1: Structure of management rules

The management document is passed to the management component and the system returns a management identifier as a reference to the management document. Thus, the customer can adapt the management rules later if necessary, e.g. if management requirements are changing or if a long-running process encounters unexpected problems to be solved. Interpretation and processing of the management rules is executed by a rule engine (cp. Section 3.3). If required, the relevant events are subscribed and event notifications are passed to the rule engine in order to perform the pattern matching. If a specified pattern is recognized, the rule engine initiates the execution of the corresponding actions.

Besides the management rules (line 18 of Listing 2), the management document holds additional information required for correlation, assignment and execution of processes and

rules. Therefore, *general information* (lines 3-7 of Listing 2) contains a *management endpoint* which is the unique identifier of the process management system to be supervised, and a *management mode* which defines when the management should end. As the management starts with passing the management document (i.e. the rule engine starts listening to the specified events) it can either terminate automatically when all process instances are finished (management mode = "system") or it can explicitly be finished by the customer (management mode ="user"). The latter is relevant if the management should also observe the process engine as a candidate for further distributions, if process history data is required later (i.e. for evaluation) or if the process model can be instantiated again from the outside (which is e.g. the case for WS-BPEL processes which deploy their own service interface for the initiation of new process instances). In case of an automatic termination, the customer can optionally specify a notification endpoint in order to be informed once the management has been finished. Termination of the management plays an important role, because here all the rule patterns have to be removed from the rule engine, and the associated event subscriptions have to be canceled.

```
1  MANAGEMENT-DOCUMENT
2
3    GENERAL-INFORMATION
4        MANAGEMENT-ENDPOINT       : <URL>
5        MANAGEMENT-MODE           : "system"|"user"
6        [NOTIFICATION-ENDPOINT    : <URL>]
7    END GENERAL-INFORMATION
8
9    INSTANTIATION-INFORMATION
10       PROCESS-MODEL-REFERENCE   : <STRING>
11       LOCAL-INSTANCE-REFERENCE  : <STRING>
12       [INSTANTIATION-TIME       : <DATE>]
13       [INSTANTIATION-DELAY      : <INTEGER>]
14       [INSTANTIATION-PARAMETERS]
15       [BLOCKING-EVENT-TYPES]
16   END INSTANTIATION-INFORMATION
17
18   MANAGEMENT-RULES
19
20 END MANAGEMENT-DOCUMENT
```

Listing 2: Structure of the management document

The part of the *instantiation information* (cp. lines 9-16 in Listing 2) contains relevant data about the process instances. It holds the reference to the associated process model (*process model reference*) and a placeholder for the process instances (*local instance reference*) which do not exist at the time of deployment, but which need to be referenced within the rule patterns and actions for instance management. In case only one instance of the process has to be assigned and executed by the remote system, the process can optionally be started immediately, at a specified point of *time* or after a specified *delay*. In this case, also the *parameters* for process instantiation have to be passed. Finally, the management document specifies which events should be able to block the execution of a process instance (cp. line 15 in Listing 2). Such *blocking events* are required to enable immediate reactions without having the problem that the process engine continues execution of the process although a prompt management action should be performed on the basis of the determined situation. Blocking events are only relevant for the monitoring of process instances and are thus also part of the instantiation information.

In order to allow customizing the location of decision-making, the processing of events and derivation of management reactions can optionally be executed on the remote system or at the customer's site. In the first case, the management document has to be transferred to the remote system, all events are caught locally and the management actions are carried out by the partner system. Thus, in general, process management data does not have to

be transferred over the network and consequently, the delay resulting from management is minimized. This option is well suited in case that mobile participants are involved and network connection is temporarily unavailable. In the second case, all information and/or events are transferred to the customer and management reactions are determined here. This is required if e.g. management decisions are confidential or need approval by a human operator. However, this strategy may decrease performance of process execution and has only few advantages over the existing approach of activity weaving as presented in Section 2. Furthermore, the customer has to run and maintain the tier 2 management component in order to make the decisions – which may not be desired e.g. in the case of BPM-as-a-Service scenarios (cp. Figure 1(d)). An attractive alternative is, however, the combination of both strategies, e.g. having general monitoring data collected and processed by the remote system and calling-back to the customer only in case of infrequent severe problems by, e.g., sending an email.

3.3 Implementation

In order to use the WSDM framework, the model of relevant characteristics and relationships of process management system, process models, instances and context has been represented as a *WSDM resource properties document* which is specified in XML. An example for the representation of the property "WorkloadInfo" is depicted in Listing 3 and 4. As stated in Section 3.1, a meta description determines each property's name, the modifiability, the mutability, the expected frequency of mutation and the availability of the property. Furthermore, XML-Schema (XSD) is used to specify structure and data type of each property. The resource properties can be accessed using the WSDM web service operations *GetResourceProperty* and *UpdateResourceProperty* as well as a set of additional operations, e.g. for cancellation of process instances, which are altogether included in the associated web service description (WSDL). The WSDL file also contains the location where the service can be accessed, e.g. a URL. Finally, WS-Notification (WSN) topics for subscription of the event interface have been specified and included, and for each event it is specified whether it should be allowed to block the process execution in order to enable a direct reaction.

The left side of Figure 6 shows an overview of the implementation with WSDM. In order to interact with the manageable resource, the management consumer now only requires

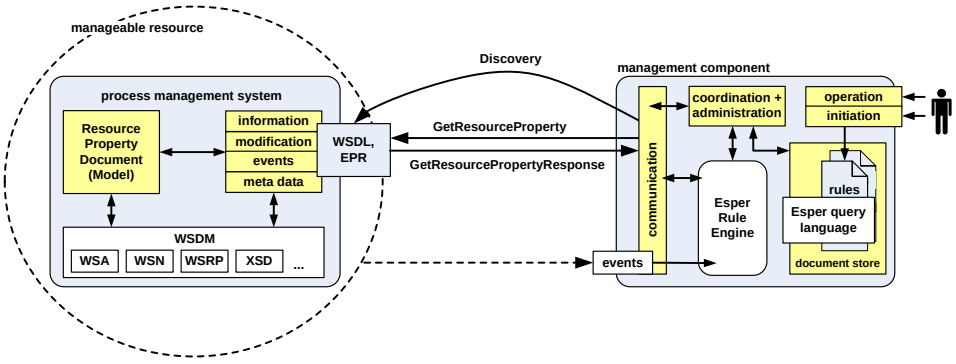


Figure 6: Prototype implementation using WSDM and Esper

the *Endpoint Reference (EPR)* of the process management system, i.e. the information from the WSDL file. After that, read-only or write requests as well as subscriptions for events can be carried out by requesting a property of the resource properties document. The requested resource respectively replies with a *GetResourcePropertyResponse* resp. *UpdateResourcePropertyResponse* message in case of a successful operation or with an exception in case of an error. In case the consumer wants to subscribe for an event, besides the topic also the type of the event can be specified, i.e. if the event should block the execution of the process or not.

```
<Property name="tns:WorkloadInfo"
          modifiability="read-only"
          mutability="mutable"
          meta:mutfreq="meta:minutes">
  <meta:availability>meta:always</meta:availability>
</Property>
```

Listing 3: Example for the meta-description of a BPM resource property within WSDM

```
<xsd:complexType name="WorkloadInfo">
  <xsd:sequence>
      <xsd:element name="RunningInstances" type="xsd:int">
      <xsd:element name="CPULoad" type="xsd:float">
  </xsd:sequence>
</xsd:complexType>
```

Listing 4: Example for the schema definition of a BPM resource property within WSDM

On side of the management component (cp. right side of Figure 6), the management document is interpreted and stored for the time of its validity. The *coordination and administration module* is now responsible for the discovery of the specified management endpoint, to subscribe for the required events and to manage time constraints and deadlines. The received event stream is processed and tested against the user-defined event patterns. In the prototype implementation, the complex event processing is done by the existing Esper rule engine (cp. Section 2). As Esper expects expressions in the SQL-based Esper query language, the abstract *event pattern* part of the management rules (cp. line 5 of Listing 1) is represented by the respective terms of this language. A simplified example for a complete management rule is presented in Listing 5. In this case, the process variable named *deadline* within all instances of the process model with id='1' is updated if the execution of an activity with id=4 takes longer than 60 seconds.

```
<Rule>
  <Name>ExtendDeadline</Name>
  <Trigger>
        SELECT * FROM PATTERN
        [EVERY (e1=ActivityStarted(activityId="4",modelId="1")
                -> TIMER:INTERVAL(60 SEC)
                -> NOT e2=ActivityFinished(activityId="4",modelId="1"))]
        WHERE e1.instanceId=e2.instanceId
  </Trigger>
  <Action>
    <Service  epr="http://vsis.informatik.uni-hamburg.de/bpms.wsdl"
              operation="UpdateResourceProperty">
      <Param type="PropertyName"><Value>DataField</Value></Param>
      <Param type="ProcessInstanceID"><Value>%{e1.instanceId}</Value></Param>
      <Param type="DataFieldName"><Value>deadline</Value></Param>
      <Param type="DataValue"><Value>60</Value></Param>
    </Service>
  </Action>
</Rule>
```

Listing 5: Example for a management rule with Esper event pattern and WSDM service invocation

4 Evaluation

So far, the prototype implementation as sketched above has been applied to two existing distributed process management systems: first to the DEMAC [ZKL09] process engine which uses XPDL processes and supports the runtime migration of process instances (cp. Figure 1(c)) and, second, to the Sliver [HHGR06] process engine which uses a subset of WS-BPEL processes. Both process engines can be applied for mobile process management and had to be modified in order to implement the proposed management API. The following example scenario is used to show the most important observations and results also in comparison to two previous approaches.

4.1 Example Scenario

Figure 7 shows an example from the *eErasmus eHigher Education (eEH)* project [JL06], which is is an international exchange program of higher education institutes among EU countries. In order to facilitate a uniform exchange of students joining this program, allow them to take courses at foreign universities and have the selected courses acknowledged by the home university, a standardized process is proposed for all participating universities. The simplified functional process used here involves subcontracting the host university for *approving the credentials* necessary for taking courses there, allowing *courses and exams* until a specified deadline and *preparing the credentials* achieved at the host university in order to acknowledge them at the home university.

The distributed execution involves several management requirements which are *expected in advance*, i.e. before execution of the process starts: (R_1) The host university is paid a certain amount of money for each student and for the associated administration effort. Therefore, the duration of each activity executed by the host university has to be logged. (R_2) In order to handle potential errors in time, the home university wants to be sure that the foreign university has received the subprocess and is able to execute it, and, (R_3) if duration of an activity expected as critical (here *preparation of credentials*) exceeds the average time for executing a task, (R_4) the activity should be skipped in order to at least allow the control flow of the process to return to the calling system. (R_5) As it sometimes happens that the deadline for taking courses is adapted by the host university, e.g. because the student gets ill, the home university wants to know about such events in order to avoid automatic removal from the home register of students.

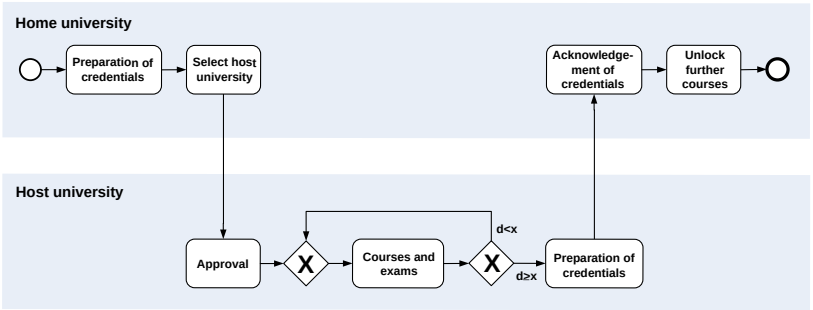


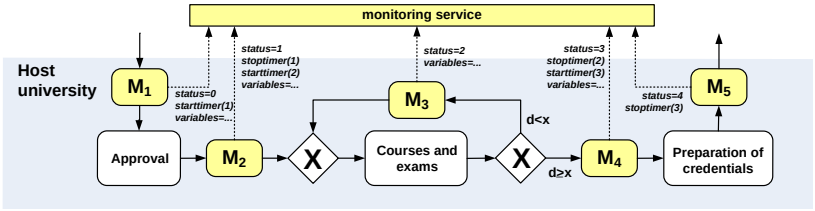
Figure 7: eErasmus example process

In addition, there are a number of *unexpected occurrences* during the runtime of this rather long-running (i.e. several months) process: First, a financial aid program asks about the status of the student's overall study (R_6). Second, the student has married and his/her name has to be adapted (R_7). The next section shows a comparison on how the presented approach and previous approaches support these management requirements.

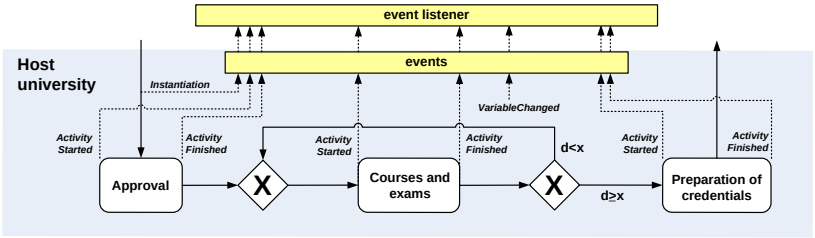
4.2 Comparison and Results

Figure 8 shows the realization of the monitored process instance with weaving of monitoring activities (such as in [BG05]), event-based monitoring only (such as in [vA09]) and the ad-hoc management approach proposed here, each realized resp. simulated with the prototype implementation presented above. Results are summarized in Table 1.

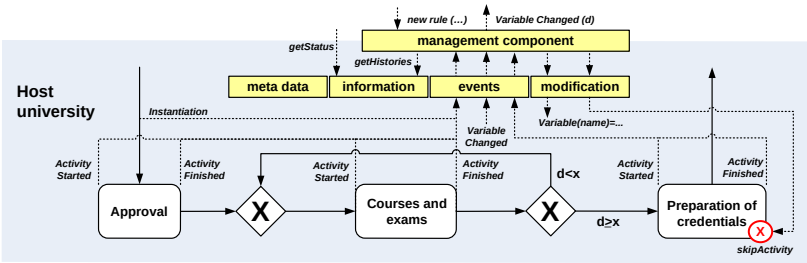
It shows that monitoring aspects which are known in advance, such as measuring of the duration of predefined activities, the start of instance execution and the observation of variable value modifications can be realized by the designtime insertion of respective monitoring activities (timer activities and passing of variables values to the central monitoring



(a) Approach using activity weaving



(b) Approach using event-based monitoring only



(c) Approach using ad-hoc management

Figure 8: Different realizations of the scenario-based management requirements

Management requirement	Ad-hoc management	Event-based monitoring	Activity weaving
(R_1) Duration of activities	+	+	+
(R_2) Instance started	+	+	+
(R_3) Detect critical activity duration	+	o	o
(R_4) Skip critical activity if necessary	+	-	-
(R_5) Observe variable value	+	+	+
(R_6) Ad-hoc status retrieval	+	o	o
(R_7) Ad-hoc variable value modification	+	-	-

Table 1: Applicability of management requirements during execution of the example process

service) and by the event-based monitoring and the ad-hoc management approach (by subscription of the respective events). The detection of abnormal activity duration can be realized by the ad-hoc management as a complex rule involving also additional information about previous process instances executed on this system and calculating their average time of execution. This is neither possible by a system which makes use of events only (e.g. the events of other process instances have not been captured before) nor by activity weaving (histories of other process instances are not visible in the monitored process instance). A simple user-defined deadline could be applied for these approaches, but this would not reach the quality of estimating whether or not the assumed duration is normal or abnormal for a specific remote system. Skipping critical activities is also a problem for previous approaches, because event-based monitoring does not offer control functionalities at all, and activity weaving cannot skip crashed activities by weaving an "end activity" because the control flow does not go on in this case and therefore will not reach this activity.

Considering the occurrence of unexpected behavior, the ad-hoc management shows its biggest advantage: The status retrieval can be made by calling the process's resource property *process status* and interesting *data values* directly. Both activity weaving and event-based monitoring can provide this data only in case a monitoring activity is inserted after each functional activity resp. all available events have been subscribed. Therefore, it is more or less a coincidence if such requests can be fulfilled as they cannot be determined in advance and relevant properties have to be weaved/subscribed before runtime. The ad-hoc variable modification is also not possible because of missing runtime modification operations. However, even by using the ad-hoc management approach, the process manager has to be careful not to violate the integrity of the process. Therefore, in case of the modification of the student's name, the process manager should abstain from calling the modification interface directly, but better update the management document by inserting a new rule. An example for such a rule would be to wait until the current activity is finished (subscribe the *activityFinished* event as a blocking event), perform the modification, and then resume execution. Furthermore, in case of ad-hoc requests and multiple instances, the appropriate process instance has to be found by an appropriate correlation set. In this case, for instance, the (previous) name of the student can be used within the management rule in order to pick the right instance.

Considering the non-functional characteristics of the approaches, it shows that desired separation between business logic and management logic can be achieved by event-based and ad-hoc management approaches (as shown in Figures 8(b) and 8(c) the original business process does not have to be changed), but not by activity weaving (cp. Figure 8(a)). Especially in the context of mobile process management, the approach of activity weaving furthermore proves to be very instable (i.e. if the monitoring service is not available, the

process execution is delayed or even fails). In case of the manageable resource, this can only happen if events are defined to be blocking and the event processing and/or reaction is performed on a remote system. For non-blocking events the management overhead is not relevant, because the execution of the process goes on and thus it is not influenced by the management. However, the time consumption for identifying complex situations and thus the time consumed before emitting the complex event is dependent on the complexity and the number of processing rules. Having three processing rules with basic to intermediate complexity in this example, the processing time is still insignificant even on resource-constraint mobile devices (i.e. less than 1 second on a netbook with Intel Atom N270, 1,6GHz, 1GB RAM). Further benchmarks also on more complex processing rules can be derived from official experiments with Esper [Esp10]. The time for invoking management services and thus for initiating reactions shows the behavior of ordinary service invocation and can be described as the sum of requesting the WSDM resource property (T_R), the time for internal computation (T_C) (e.g. for calculating a prognosis for activity duration) and the time for network transfer (T_N). In case of a local management component and without calling external services, T_R is again insignificant and $T_N = 0$. Of course, T_C and T_N are dependent on the complexity of computation and service availability and respectively of the bandwidth and speed of the network connection. Finally, for the solely event-based approach, no delays effected by the management are visible at all – however no reactions are possible and thus events can be emitted in parallel to an ongoing process execution without delay. Finally, compared with both event-based and ad-hoc management approaches, activity weaving has the important advantage that no system modifications, security mechanisms or agreements are necessary.

5 Conclusion

In today's highly dynamic business networks, customized monitoring and controlling options for distributed business processes gain increasing importance. This paper advances existing approaches for the management of such processes by presenting a concept to not only passively observe the behavior of business processes running on a remote process management system but also to enable quick automatic and spontaneous reactions on the basis of a service-based management interface. Thereby, the presented approach allows for increased flexibility during process execution – taking into account also the requirements of modern distributed process management variants such as BPM-as-a-Service or integration of mobile (sub-) systems – and the integration of valuable functionalities of remote process management systems which have not been exploited before. The price for such increased flexibility is, however, the necessity to integrate and configure a corresponding add-on infrastructure. Process managers also have to be aware of their respective new potential, e.g. by influencing process execution during runtime which may lead to undesired side effects, and in the worst case, to inconsistent process execution. Furthermore, the presented approach has to be secured so that both the provider of management functionality as well as the consumer of distributed process management are protected in a sufficient way, i.e. a customer should only be able to access his own process models and instances, and information about general internal data such as engine properties or workload should be subject to negotiation. Therefore, the conceptualization and application of relevant protective measures and specific customizable security and privacy mechanisms must be an important part of future work.

References

[BG05] Luciano Baresi and Sam Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Third International Conference of Service-Oriented Computing (ICSOC)*, volume 3826. Springer, 2005.

[BGG04] Luciano Baresi, Carlo Ghezzi, and Sam Guinea. Smart Monitors for Composed Services. In *ICSOC '04: Proceedings of the 2nd international conference on Service oriented computing*, pages 193–202, New York, NY, USA, 2004. ACM Press.

[Bui08] Hai-Lam Bui. Survey and Comparison of Event Query Languages Using Practical Examples. Master's thesis, Ludwig-Maximilians-Universität Munich, Nov 2008.

[Esp10] EsperTech. Esper - Performance. http://docs.codehaus.org/display/ESPER/Esper+performance, May 2010.

[HA08] Melanie Hartmann and Gerhard Austaller. *Ubiquitous Computing Technology for Real Time Enterprises*, chapter Context Models and Context Awareness, pages 235–256. IGI Publishing, 2008.

[HHGR06] Gregory Hackmann, Mart Haitjema, Christopher D. Gill, and Gruia-Catalin Roman. Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices. In Asit Dan and Winfried Lamersdorf, editors, *International Conference on Service-Oriented Computing (ICSOC 2006)*, volume 4294, pages 503–508. Springer, 2006.

[IBM04] IBM. Common Base Event. http://www.ibm.com/developerworks/library/specification/ws-cbe/, August 2004.

[J+01] Stefan Jablonski et al. A Comprehensive Investigation of Distribution in the Context of Workflow Management. In *ICPADS 2001*, pages 187–192, 2001.

[JL06] R. Vermer Juliet Lodge. Case Study e Erasmus eHigher Education (eEH). Technical report, SIXTH FRAMEWORK PROGRAMME, Information Society Technologies, R4eGov, Deliverable WP3 D1-D4, 2006.

[KZTL08] Christian P. Kunze, Sonja Zaplata, Mirwais Turjalei, and Winfried Lamersdorf. Enabling Context-based Cooperation: A Generic Context Model and Management System. In *Business Information Systems (BIS 2008)*. Springer, 5 2008.

[Luc02] David Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, May 2002.

[MWL08] Daniel Martin, Daniel Wutke, and Frank Leymann. A Novel Approach to Decentralized Workflow Enactment. In *Enterprise Distributed Object Computing*, pages 127–136. IEEE, 2008.

[NM02] Roberta Norin and Mike Marin. Workflow Process Definition Interface – XML Process Definition Language. Specification WFMC-TC-1025, Workflow Management Coalition, 2002.

[OAS06a] OASIS. Web Services Distributed Management: Management of Web Services (WSDM-MOWS) 1.1. Standard Specification, 2006.

[OAS06b] OASIS. Web Services Distributed Management: Management Using Web Services (WSDM-MUWS) 1.1. Standard Specification, 2006.

[OAS07] OASIS. Web Services Business Process Execution Language Version 2.0. Technical report, OASIS, 2007.

[OMG09] OMG. Business Process Model and Notation (BPMN), Version 1.2. Technical report, OMG, 2009.

[SP04] Thomas Strang and Claudia L. Popien. A Context Modeling Survey. In *UbiComp 1st International Workshop on Advanced Context Modelling, Reasoning and Management*, pages 31–41, Nottingham, September 2004.

[SPG04] Keith D. Swenson, Sameer Pradhan, and Mike D. Gilger. Wf-XML 2.0 XML Based Protocol for Run-Time Integration of Process Engines . Technical report, WfMC, 2004.

[vA09] Rainer von Ammon. Event-Driven Business Process Management. In *Encyclopedia of Database Systems*, pages 1068–1071. Springer, 2009.

[vdA98] Wil M. P. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[vdA00] Wil van der Aalst. Loosely coupled interorganizational workflows: modeling and analyzing workflows crossing organizational boundaries. *Inf. Manage.*, 37(2), 2000.

[VDATHKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.

[vLLM+08] Tammo van Lessen, Frank Leymann, Ralph Mietzner, Jorg Nitzsche, and Daniel Schleicher. A Management Framework for WS-BPEL. *Web Services, European Conference on*, 0:187–196, 2008.

[WfM98] WfMC. Workflow Management Coalition Audit Data Specification. Specification WFMC-TC-1015, Workflow Management Coalition, 1998.

[WKK+10] Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, and Daniel Zwink. Cross-Organizational Process Monitoring based on Service Choreographies. In *Proceedings of the 25th Annual ACM Symposium on Applied Computing (SAC 2010)*, pages 2485–2490. ACM, 2010.

[ZKL09] Sonja Zaplata, Christian P. Kunze, and Winfried Lamersdorf. Context-based Cooperation in Mobile Business Environments: Managing the Distributed Execution of Mobile Processes. *Business and Information Systems Engineering (BISE)*, 2009(4):301–314, 10 2009.

[ZKML10] Sonja Zaplata, Kristian Kottke, Matthias Meiners, and Winfried Lamersdorf. Towards Runtime Migration of WS-BPEL Processes. In *Fifth International Workshop on Engineering Service-Oriented Applications (WESOA'09)*. Springer, 4 2010.

[ZL10] Sonja Zaplata and Winfried Lamersdorf. Towards Mobile Process as a Service. In *25th ACM Symposium On Applied Computing (SAC 2010)*, pages 372–379. ACM, 3 2010.

A.7 Cross-Organizational Process Monitoring based on Service Choreographies

Authors:

USTUTT: Branimir Wetzstein

USTUTT: Dimka Karastoyanova

USTUTT: Oliver Kopp

USTUTT: Frank Leymann

USTUTT: Daniel Zwink

- Submitted to: SAC 2010

Cross-Organizational Process Monitoring based on Service Choreographies

Branimir Wetzstein, Dimka Karastoyanova, Oliver Kopp, Frank Leymann, Daniel Zwink
Institute of Architecture of Application Systems
Universitaetsstr. 38
Stuttgart, Germany
lastname@iaas.uni-stuttgart.de

ABSTRACT

Business process monitoring in the area of service oriented computing is typically performed using business activity monitoring technology in an intra-organizational setting. Due to outsourcing and the increasing need for companies to work together to meet their joint customer demands, there is a need for monitoring of business processes across organizational boundaries. Thereby, partners in a choreography have to exchange monitoring data, in order to enable process tracking and evaluation of process metrics. In this paper, we describe an event-based monitoring approach based on BPEL4Chor service choreography descriptions. We show how to define monitoring agreements specifying events each partner in the choreography has to provide. We distinguish between resource events and complex events for calculation of process metrics using complex event processing technology. We present our implementation and evaluate the concepts based on a scenario.

Keywords

Business Activity Monitoring, Cross-Organizational Monitoring, Service Choreography

1. INTRODUCTION

Business Process Management (BPM) encompasses methods, techniques, and tools that allow organizing, executing, and measuring the processes of an organization [12]. When BPM is layered over a Service Oriented Architecture (SOA) [9], services are used for implementing activities of business processes. In the context of SOA, business processes are modeled and executed using the (WS-)BPEL language, which is a workflow language for orchestration of Web services. While a service orchestration implements an executable private process model implemented by a single participant, a service choreography models the publicly visible processes and message exchanges between participants from a global viewpoint [10]. BPEL4Chor is a BPEL extension for modeling service choreographies [3].

For controlling the achievement of business goals especially in business processes and measuring process performance, business activity monitoring (BAM) technology enables continuous, near real-time event-based monitoring of business processes based on key business metrics, also known as key performance indicators (KPI) [11]. Business process monitoring has been traditionally focused on intra-enterprise processes. Today, companies are forced to collaborate in a more open manner in order to meet joint customers' needs. There is also more and more outsourcing of parts of business processes to external companies. Thereby, an intra-enterprise business process is fragmented into a cross-organizational process and the source company is often still interested in monitoring of the outsourced process fragment. A well-known example is shipment tracking whereby the shipper opens its process to some extent to the customer. Thus, there is a need for companies to interchange monitoring data of their business processes with other companies.

In this paper we present a solution to this problem by describing an event-based approach to cross-organizational monitoring based on service choreography descriptions. We use BPEL4Chor choreographies as basis for specification of so called monitoring agreements. A monitoring agreement specifies which events need to be provided by each partner in the choreography for building a monitoring solution. In particular, we distinguish between resource events which are defined based on the abstract processes in the choreography description and complex events needed for calculating higher-level process metrics using a complex event processing (CEP) language [8]. In order to support event correlation across partners in a choreography, we show the need for a choreography instance identifier and describe how it can be used in SOAP-based communication. We have implemented the approach in the Web services setting by extending an existing BPEL engine and using a CEP framework.

The rest of the paper is organized as follows. In Section 2 we present the motivation for our work based on a scenario which we use in the rest of the paper to present examples for our concepts. In Section 3 we depict the overall approach. Section 4 describes in detail how monitoring agreements are modeled. Section 5 deals with the monitoring infrastructure and event correlation in choreographies. In Section 6 we present related work, and finally, in Section 7 we conclude the paper and outline our future work.

2. SCENARIO AND MOTIVATION

For explaining the motivation and concepts of our work we have chosen a purchase order scenario as illustrated in the BPMN diagram shown in Figure 1. The diagram shows a chorography between a customer, a reseller, and a shipper

SAC'10 March 22-26, 2010, Sierre, Switzerland.

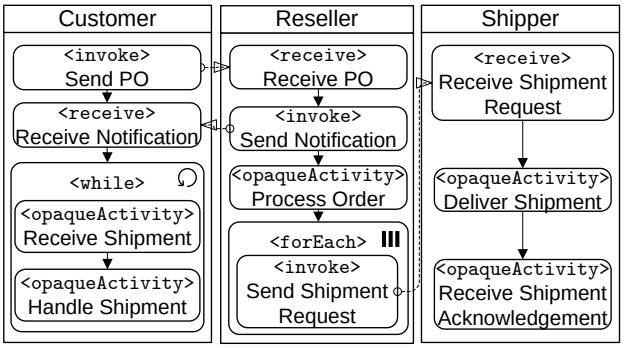


Figure 1: Purchase Order Scenario

(other involved participants such as suppliers have been omitted for space reasons). The customer sends an order request with details about the required products and needed amounts to the reseller. The reseller confirms the order by sending a notification to the customer. The reseller processes the order by ordering products from suppliers if needed, packages them and notifies the shipper. The order can be split in several parts if some of its parts take much longer to deliver. In that case the order parts are shipped separately. The shipper delivers the products to the customer.

Firstly, note that the diagram shows only public processes of the participants and their interactions. Private parts of the process are denoted by *opaque* activities and their implementation is not exposed to other participants. Note also that for this paper we assume that the partners have agreed on this choreography using the BPEL4Chor language [3]. A BPEL4Chor choreography description consists of a set of abstract BPEL process models (in our scenario three of them), one for each participant type (customer, reseller, and shipper), and a topology document which specifies how these abstract processes are connected together by message links. The choreography description also specifies concrete participants for participant types (e.g., two concrete shipping companies or concrete customers involved in the choreography; BPEL4Chor also supports dynamic sets of participants; however their monitoring is out of scope of this paper and part of our future work). Each partner implements its abstract business process as defined in the choreography locally typically using WS-BPEL, but not necessarily so. A partner could also use, for example, Java as long as it behaves according to the specified abstract BPEL process [4].

In the following we will motivate the need for our approach based on scenario examples. Considering the monitored objects, i.e., what is to be monitored, we can distinguish between process tracking and evaluation of process metrics. In process tracking, partners want to track the state of the choreography beyond their own process. In our scenario, for example, the customer is interested in tracking how far the order processing is. Obviously, this information can be provided by the reseller and shipper when they publish events as their process is executed, such as `Order received`, `Order processed`, `Shipment request received` and so on. We assume in this paper, that partners are willing to provide this information (or a subset of it) as long as it is part of its "public" process as modeled in the choreography. Note that we assume here that there is no special Web service operation provided by the reseller and shipper for inquiring this information; this would be a special case, and we concentrate on event-based monitoring in this paper. For process tracking, one has to agree for *which* process resource and which state change of that resource the event is to be published, *what* data (process data and IDs for correlation) is transmitted in the event, and *where* the event can be retrieved (on which messaging queue or pub/sub topic). Obviously, for unambiguous specification of the *which* and *what* question, an underlying choreography model is needed as basis. This is in our case the BPEL4Chor description. Process tracking relies only on state changes of process resources (in case of event-based monitoring also known as resource events). Besides process tracking it is often needed to evaluate metrics based on complex events. These metrics are then used e.g. as basis for definition of Service Level Agreements (SLAs) or Key Performance Indicators (KPIs). Consider, for example, the metric *order fulfillment lead time* which could be measured in our scenario from the start of the activity `Receive PO` in the reseller process until the `While Loop` completes in the customer process. Therefore, corresponding events have to be gathered, correlated and their timestamps subtracted. In general, thus we have to be able to specify CEP-like complex events based on events of different partners. The problem which arises here is that of event correlation in the choreography. In particular in this scenario we have to be able to correlate process instances within a choreography instance execution. In Section 5 we will explain in more detail, why in the general case, a special technical choreography instance ID is needed which has to be transported on protocol level, e.g., as part of SOAP headers.

In cross-organizational monitoring, obviously there are privacy issues. Firstly, the assumption of our approach is that partners are willing to provide only monitoring information on their public processes, but not private processes. This is why we have chosen to take a choreography description as basis for monitoring specification. It should also be possible to specify events selectively even for the public process. One could think of different monitoring levels dependent on how much the customer wants to pay for that information (if we assume that monitorability is part of service levels and is sold as a feature). Another issue is to be able to selectively restrict which partners can see which events.

3. OVERVIEW OF THE APPROACH

Figure 2 sketches the main concepts of our approach. The service choreography description can be seen as an agreement between partners on their public processes and message exchanges. We base our approach on a BPEL4Chor service description in which each partner exposes an abstract BPEL process [3]. We introduce a *monitoring agreement* which is an XML-based document specifying monitoring aspects between partners based on the choreography description. A monitoring agreement consists of a set of *resource event* definitions and *complex event* definitions. Resource events are defined based on abstract BPEL processes in the choreography by specifying at which BPEL resource and for which state of that resource an event is to published, which data it should contain, and where it should be published (at which message queue or pub/sub topic). Complex events are defined based on resource events and other complex events using a Complex Event Processing (CEP) language. They are needed for calculating process metrics. Both resource events and complex events are exchanged between partners over *message queues* or alternatively *pub/sub topics*.

Considering the methodology in creating corresponding monitoring agreements, there are two possible approaches. In a top-down approach the parties agree on what is to be monitored

and create together a monitoring agreement document, possibly during creation of the choreography document itself. The document is then deployed to each party's infrastructure. The infrastructure is configured considering which events it has to publish to other partners and which events it retrieves from others. A more dynamic, bottom-up approach would imply that each partner creates the corresponding XML document (which is not yet an "agreement") independently of other partners and specifies which events it provides (and optionally also which events it requests) to partners in the choreography, possibly exposing different monitoring levels based on e.g. service levels and price the requester wants to pay. This monitoring document could then be published to a service registry together with the WSDL and choreography document. Obviously, in such a scenario a matchmaking phase is needed which checks whether requested and provided monitoring events match finally creating a *monitoring agreement* as a result. In this paper we focus on the top-down approach and leave the bottom-up one for future work.

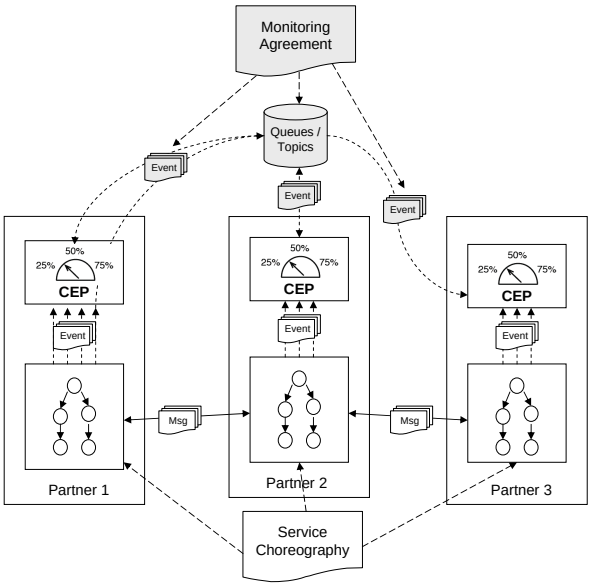


Figure 2: Overview of the Approach

Considering the lifecycle in creation of corresponding cross-organizational monitoring solutions we can distinguish between three phases: creation of monitoring agreements, deployment, and the concrete monitoring. After creation of the monitoring agreement document, it is used by each partner to configure its middleware for monitoring in the deployment phase. That involves configuring its own process middleware eventing infrastructure for publishing events to the specified destinations and subscribing to event queues or topics for getting events from other partners. Note that partners have to support managing the choreography instance identification in SOAP based message communication (Section 5).

4. MODELING OF MONITORING AGREEMENTS

The monitoring agreement is an XML document consisting of two types of definitions: resource event definitions and complex event definitions. Resource event definitions are specified based on the choreography descriptions and complex events are defined based on other events. Complex events can serve as basis for specification of SLAs and BAM solutions. One could for example specify service level parameters and service level objectives based on complex events which contain corresponding metric values. That is however out of scope of this paper.

4.1 Definition of Resource Events

Resource events are defined based on the abstract BPEL process models in the BPEL4Chor choreography. A resource event definition specifies the following three elements:

- *Monitored Resource:* Firstly, we have to specify which process resource should be monitored and for which state of the resource the event should be published. The resource is identified by pointing to the corresponding BPEL4Chor elements. Monitored resources we are interested in are the instances of the BPEL process, activity, scope, and variable. The state models (e.g., *started*, *completed*, *terminated*, *compensated*, and corresponding transitions) for these resources are not standardized. We use the state models defined in [5]. The resource identification will result at process runtime in corresponding resource identifiers which are transported in the event and are needed for event correlation, as discussed in Section 5.

- *Process Data:* Optionally, one can specify which process data (defined as BPEL variable) is to be part of the event. The data is read at the moment of event publishing.

- *Target message queue or pub/sub topic:* Finally, one has to specify a message queue or a pub/sub topic to which the event is to be published. If the resource event is to be published to a partner only under a certain filtering condition, e.g., some attribute value contained in the event or some other events, an additional complex event has to be created which is created based on this condition (discussed further below). The access to the queue or topic can be restricted to certain participants by specifying their names; the concrete realization mechanism for access control, needed credentials etc. have to be specified separately.

```
<monitoringAgreement
  xmlns:chor="http://purchaseOrder/choreography"
  xmlns:reseller="http://purchaseOrder/reseller">
 <resorceEventDefinitions>
  <resourceEventDefinition name="OrderReceivedEvent">
   <monitoredResource
     choreography="chor:orderChoreography"
     process="reseller:ResellerProcess"
     scope="process"
     activity="reseller:ReceivePO"
     state="completed"/>
   <data>
    <processVariable name="order"
      variable="purchaseOrder"/>
   </data>
   <publish>
    <queue name="purchaseOrder.reseller"
      access="reseller"/>
   </publish>
  </resourceEventDefinition>
  ...
 <resorceEventDefinitions>
 ...
<monitoringAgreement>
```

Listing 1: Resource Event Definition

Listing 1 shows a *resource event definition* for the `OrderReceived` resource event. It is specified by pointing to the `Receive PO` activity in the reseller process model. The event is to be published when the corresponding activity is `completed`. In addition, the event should contain the data from the `purchaseOrder` variable. It is published to the queue which can only be accessed by the reseller. As there are several customers and shippers as potential participants in this choreography, we cannot simply give access to this queue to all participants. Further below, in Section 4.2, we will define a complex event which sends this event to the customer who actually requested this order.

4.2 Definition of Complex Events

Complex events are specified by correlating and aggregating existing events. Event correlation and aggregation is a well-known topic in the area of complex event processing (CEP) and there are different languages available for the specification of complex events [8]. In our case, we have decided to use the language of ESPER[1], which is the CEP implementation we have used in our prototype (Section 5.2). But alternatively any other language could be used instead, the choice being dependent on aspects such as language expressivity needed. Note that we use the term *complex event* for an event which results from using a CEP statement over one or more events; we do not further distinguish between more fine-grained meanings of complex, composite, and derived events as in some other works.

```
<monitoringAgreement
  xmlns:chor="http://purchaseOrder/choreography"
  xmlns:reseller="http://purchaseOrder/reseller">
 ...
 <complexEventDefinitions>
  <complexEventDefinition providedBy="reseller"
    name="CustomerAOrderReceivedEvent"
    choreography="chor:orderChoreography">
   <consume>
    <queue name="purchaseOrder.reseller"/>
   </consume>
   <eventAggregation resultType="FILTER">
    <statement><![CDATA[
     SELECT a
     FROM PATTERN [
      EVERY a=ResourceEvent(name="OrderReceivedEvent"
      AND variables('order').customer="customerA")]
    ]]></statement>
   </eventAggregation>
   <publish>
    <queue name="orderChoreography.customerA"
      access="customerA"/>
   </publish>
  </complexEventDefinition>
  ...
 </complexEventDefinitions>
<monitoringAgreement>
```

Listing 2: Complex Event for Event Filtering

The complex event definition consists of an event aggregation statement and the target topic definition. In addition, we have to specify by whom the aggregation is performed and published on the target queue or topic (`providedBy` attribute). The reason is that we have to avoid that several partners perform this aggregation, as this would lead to a duplication of events. The event aggregation statement uses the CEP language to construct a new complex event out of already defined resource events and complex events. Therefore, we first specify from which queues or topics these existing events are `consumed`. Later, when referencing those events (correctly speaking: event streams) in the `eventAggregation` statement, we use the names from the corresponding event definitions. Considering monitored resource identifiers needed for correlation of events (see Section 5.1 for more details) we use the following naming scheme: cid stands for choreography ID and ciid for choreography instance ID, pid for process ID and piid for process instance ID, sid for scope ID and siid for scope instance ID, aid for activity ID and aiid for activity instance ID.

Complex events definitions are specified recursively based on resource events to achieve two purposes: (i) event filtering and (ii) event aggregation in order to evaluate complex process metrics. In some cases, event filters have to be defined for resource events in order to ensure that the events are delivered to the right participants. Consider in our example the `OrderReceivedEvent` (Listing 1). For privacy reasons, it should only be visible to the customer which placed the order and not to other potential customers which are also defined as participants in the choreography (but that do not participate in this particular choreography instance). If we assume that the `customerID` is part of the `purchaseOrder` variable then we can define an event filter as shown in Listing 2. Only those `OrderReceivedEvents` which contain the correct customerID are placed into the queue `orderChoreography.customerA` accessible only by customerA.

```
<complexEventDefinition providedBy="reseller"
  name="CustomerAOrderFulfillmentTime"
  choreography="chor:orderChoreography">
 <consume>...</consume>
 <eventAggregation resultType="COMPLEX">
  <statement><![CDATA[
   SELECT
    abs(b.timestamp - a.timestamp) AS metricValue,
    "ms" AS unit,
    a.resource.ciid AS ciid
   FROM PATTERN [ EVERY
    a = ResourceEvent(
      name="CustomerAOrderReceivedEvent")
    -> b = ResourceEvent(
       name="CustomerAShipmentReceivedEvent"
       AND resource.ciid = a.resource.ciid) ]
  ]]><statement>
 </eventAggregation>
 <publish>
  <queue name="orderChoreography.customerA"
    access="customerA"/>
 </publish>
</complexEventDefinition>
```

Listing 3: Complex Event for Metric Computation

Besides event filtering, another important use case for complex events is evaluation of process metrics. In Listing 3 we define a complex event `CustomerAOrderFulfillmentTime` which contains the corresponding metric value in the attribute `metricValue`. In addition it contains the attribute `unit` and the choreography instance identifier. The metric value is calculated by correlating two events already defined, namely `CustomerAOrderReceivedEvent` and `CustomerAShipmentReceivedEvent`. These events are correlated based on choreography instance IDs and then their timestamps are subtracted. The result event is published to the corresponding queue by the reseller who also performs this event aggregation. Note that obviously such a definition results in one result event per choreography instance, i.e. an event stream.

5. MONITORING OF CHOREOGRAPHIES

After the monitoring agreement is created, it is deployed to

[1]http://esper.codehaus.org

each partner's infrastructure. The partner thereby extracts from the agreement the events it has to provide and configures its middleware, e.g., the BPEL engine using a deployment descriptor to provide resource events, and the CEP engine to provide complex events. It also subscribes to topics or queues where he receives events from other partners. A possible realization is described in Section 5.2.

5.1 Event Correlation in Choreographies

In order to be able to perform event correlation for monitored resources, corresponding resource identifiers have to be included in events. Consider, for example, the calculation of the order processing duration between the activity `Receive PO` and `Receive Shipment Acknowledgment` in our scenario. For each of those activities an event stream is created. Obviously, we need to correlate events belonging to the same purchase order, i.e. the same choreography instance. Thus, events have to contain identifiers of the corresponding monitored resource. In this case, each event belongs to a certain activity instance (note that in general there can be several activity instances per activity if that activity is contained in a loop). An activity instance belongs again to a process instance. However, in this case those two identifiers (piid and aiid) are not enough as those two activities are part of different process models. In our case, in order to be able to correlate the corresponding two events, we have to correlate on the choreography instance level. Thus, the events have to contain an identifier which identifies the choreography instance.

For identifying monitored resources, either technical or business IDs are needed. In the case of process models which are realized as executable BPEL processes, for example, the BPEL engine assigns technical process instance IDs to process instances, scopes and activities (however not to choreographies). For a choreography instance, in the general case also an identifier is needed. In some special cases, correlation could be done based on business identifiers transported in BPEL messages. For example, if the orderID is known to the shipper, then it can be sent in the `Order Shipped` event and thus correlated with the `Order Received` event. However, in the general case this cannot be ensured. In synchronous invocations (BPEL invoke with input and output) the correlation between the sent and replied message is done on protocol level, e.g., SOAP/HTTP and not based on message payload (which does not necessarily contain needed identifiers). Assume for example, the synchronous invocation of the reseller process to a warehouse process to check whether all products are in stock. In that case the orderId is not necessarily part of that message. If now the reseller subscribes to the events of the warehouse those events have to contain a technical identifier.

```
<soap:header>
 <chor:choreography
   xmlns:chor="http://iaas/monitoring/choreography">
  <chor:cid>
    {http://.../choreography}orderChoreography
  </chor:cid>
  <chor:ciid>
    {...}orderChoreography/2009-11-02-12:05:21:005
  </chor:ciid>
 </chor:choreography>
 ...
</soap:header>
```

Listing 4: Choreography-ID in SOAP Header

This is why we need in the general case a technical identifier of the choreography instance which is transported on protocol level. For SOAP-based communication this identifier can be transported in the SOAP header. Obviously, the corresponding middleware, e.g., BPEL engine and service bus, has to be adapted to include and read the identifier during message exchanges. It also has to be transported in events. Listing 4 shows an example SOAP header with the choreography instance identifier. It is created when the first process instance of the choreography instance is created, in our case when `Send PO` in the customer process has started. It contains the choreography ID + timestamp of creation. Note that this approach also works in case of multiple *alternative* start activities; it does however not support multiple start activities triggered in parallel at different participants for the same choreography instance. The semantics of the latter case is not described in the BPEL4Chor specification and we have not found any use case for it.

5.2 Implementation

We have implemented the approach as shown in Figure 3. It shows the implementation from the point of view of one partner in the choreography. The prototype implementation is based on the Apache ODE BPEL engine[2] and the ESPER event processing framework.

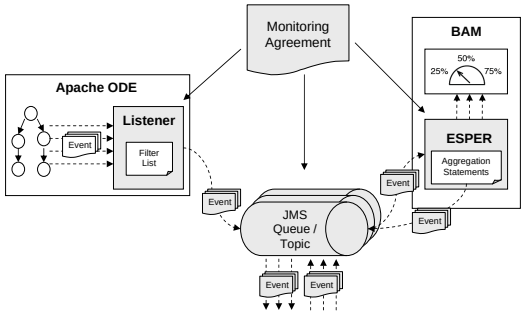


Figure 3: Tool Support

The monitoring agreement XML document which we create by hand is deployed to both an event listener in the ODE engine and the ESPER engine which is part of a BAM tool with a GUI for displaying metrics. The resource event generation is performed by an event listener which we have implemented as part of the ODE engine. It is configured by reading the monitoring agreement document and extracting the resource event definitions relevant for this participant. As the process instances are executed, it receives all internal ODE events resulting from the process execution and filters them according to the resource event definitions from the monitoring agreement. Information from internal ODE process events is taken and augmented with needed process data and choreography instance identifier read from the process instance context and the corresponding (external) resource event is created. It is sent to a JMS queue or topic as defined in the monitoring agreement. We have used Apache ActiveMQ as our JMS implementation[3]. The complex event generation and receipt of events from other participants is implemented by the BAM tool in our architecture. It uses ESPER as the underlying CEP framework and contains a GUI for displaying received events and has some dashboards for showing calculated metrics. Event aggregation statements from the monitoring agreements are registered as

[2]http://ode.apache.org
[3]http://activemq.apache.org

ESPER statements and produce complex events which are published on specified queues or topics. Finally, the last part we had to implement is dealing with the choreography instance ID (ciid). Therefore, the ciid is read from and written into SOAP headers in message interactions with other choreography participants. It is saved in the process instance context and is also propagated to the event listener which then writes the ciid into events. A new ciid is created if there is no ciid in the received message which is the case during instantiation of the first process instance of the choreography.

6. RELATED WORK

As already explained in the introduction, state of the art event-based process monitoring solutions are based on BAM technology and focus on intra-organizational processes. There exist several research approaches [1, 2] and products [11] which deal with evaluation of process metrics in near real time and their presentation in dashboards. They all have in common that events are emitted as the process is executed, collected by a process monitor and evaluated in near real time. Some solutions focus on monitoring of BPEL processes [1, 2], while others are more general and support an extensible architecture via event adapters [11]. These approaches are similar to ours in that they also use an event-based approach based on BPEL processes. However, they focus on single BPEL orchestrations and do not deal with monitoring of choreographies in a cross-organizational setting. The only approach we are aware of which deals with monitoring of BPEL processes in a cross-organizational setting is presented in [7]. Thereby, a common audit format is presented which allows processing and correlating events across different BPEL engines. In our approach, we also assume that the participants have agreed on state models of BPEL resources and resulting resource event definitions. In addition, we deal with complex events specification and event correlation in choreographies using a choreography instance identifier which is not supported in [7].

Service Level Agreements (SLA) are similar to our problem in that they involve monitoring in a cross-organizational setting. Thereby mostly two partners, the service consumer and the service provider, agree on certain service QoS, typically technical characteristics such as availability and response time [6]. The commonalities with monitoring in our context are that in an SLA partners also agree on metrics and how they are to be monitored. However, in our case the focus is on event-based monitoring of process metrics across participants in a choreography which is not being dealt with in frameworks such as WSLA focusing on QoS measurements. Our approach, however, could be extended towards specification of SLAs based on the monitoring agreement.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented an approach to event-based process monitoring based on service choreographies. A monitoring agreement is created which defines events each partner in the choreography has to provide. We have distinguished between resource events which are defined based on a BPEL4Chor choreography description and complex events using a CEP language. We have also shown the need for a choreography instance identifier for event correlation and how it can be included in a SOAP based communication.

Throughout the paper we have already discussed several possible extensions for our future work. In this paper we focused on an event-based monitoring approach (a.k.a. push model). We will extend this approach by enabling also partners requesting monitoring information on demand (a.k.a. pull model). Furthermore, we will explore dealing with a dynamic set of unknown participants at design time. At the moment, in order to ensure privacy, static queues and topics with corresponding access rights are defined in the agreement; we plan to extend this towards creating dynamic queues and topics for participants which are not known before runtime. Finally, we will deal with a bottom-up approach to specification of monitoring agreements as discussed in Section 3 and we will explore the usage of Web service Distributed Management (WSDM) set of specifications as underlying monitoring infrastructure.

Acknowledgments The research leading to these results has received funding from the European Community's 7th Framework Programme under the Network of Excellence S-Cube - Grant Agreement no. 215483.

8. REFERENCES

[1] F. Barbon, P. Traverso, M. Pistore, and M. Trainotti. Run-Time Monitoring of Instances and Classes of Web Service Compositions. In *Proceedings of the IEEE International Conference on Web Services(ICWS'06)*, pages 63–71, 2006.

[2] L. Baresi and S. Guinea. Towards Dynamic Monitoring of WS-BPEL Processes. In *Proceedings of the 3rd International Conference of Service-Oriented Computing (ICSOC'05)*, pages 269–282. Springer, 2005.

[3] G. Decker, O. Kopp, F. Leymann, and M. Weske. BPEL4Chor: Extending BPEL for Modeling Choreographies. In *ICWS*, Salt Lake City, USA, July 2007.

[4] G. Decker, O. Kopp, F. Leymann, and M. Weske. Interacting Services: From Specification to Execution. *Data & Knowledge Engineering*, 68(10):946 – 972, 2009.

[5] D. Karastoyanova, R. Khalaf, R. Schroth, M. Paluszek, and F. Leymann. BPEL Event Model. Technical Report 2006/10, University of Stuttgart, Germany, November 2006.

[6] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *J. Netw. Syst. Manage.*, 11(1):57–81, 2003.

[7] S. Kikuchi, H. Shimamura, and Y. Kanna. Monitoring Method of Cross-Sites' Processes Executed by Multiple WS-BPEL Processors. In *CEC/EEE*, pages 55–64, 2007.

[8] D. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Professional, May 2002.

[9] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer*, 11, 2007.

[10] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.

[11] U. Wahli, V. Avula, H. Macleod, M. Saeed, and A. Vinther. *Business Process Management: Modeling Through Monitoring Using WebSphere V6.0.2 Products*. IBM, International Technical Support Organization, 2007.

[12] M. Weske. *Business Process Management: Concepts, Languages, Architectures*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.