Institute of Architecture of Application Systems

A Model-Driven Approach to Implementing Coordination Protocols in BPEL

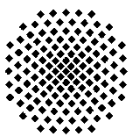
Oliver Kopp[1], Branimir Wetzstein[1], Ralph Mietzner[1], Stefan Pottinger[2], Dimka Karastoyanova[1], Frank Leymann[1]

[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.

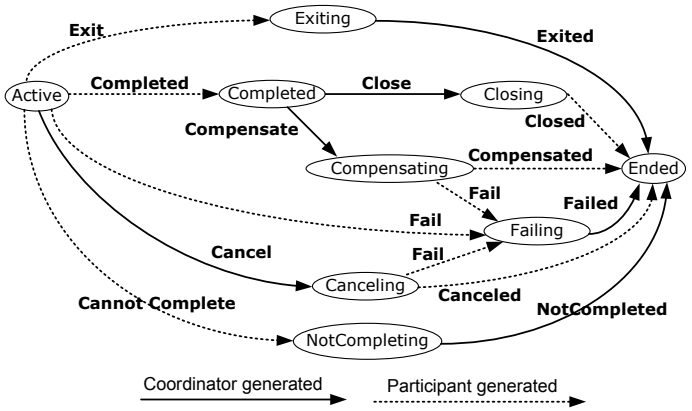


Fig. 1. WS-BA with Participant Completion Protocol [6]

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

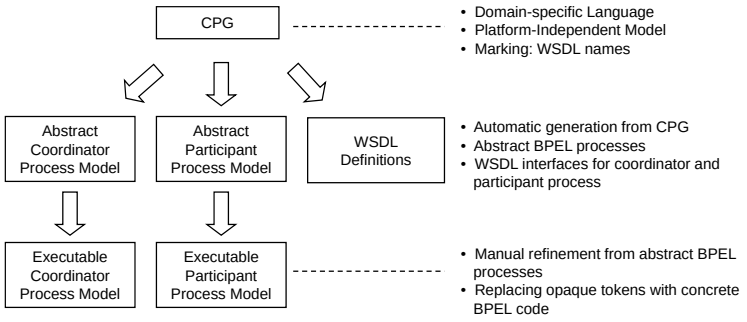


Fig. 2. Model-Driven Implementation Approach

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

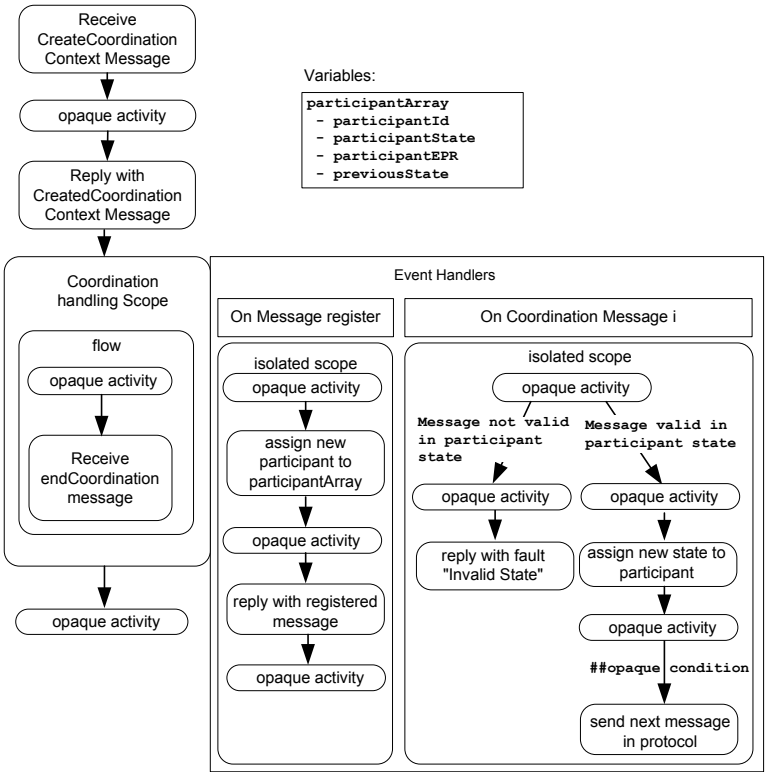


Fig. 3. Pattern for the generation of coordinator scopes

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)