Using Event-Driven Process Chains for Model-Driven Development of Business Applications

Daniel Lübke[(1)], Tim Lüecke[(1)], Kurt Schneider[(1)], Jorge Marx Gómez [(2)]

daniel.luebke@inf.uni-hannover.de, tlueecke@acm.org,
kurt.schneider@inf.uni-hannover.de, marx-gomez@informatik.uni-oldenburg.de

[(1)]University of Hannover, FG Software Engineering
[(2)]University of Oldenburg, Department of Computer Science

Abstract: Web services provide a standardized way of accessing functionality over networks. Most beneficial is their use if many Web services are composed in order to develop an application. Due to their nature, Web services can be used to support businesses if their composition matches the underlying business processes. However, the activities related to composition as well as design of a corresponding user interface are still time consuming. This is especially true in small and medium sized enterprises (SMEs) due to their available resources. Therefore, we propose a light-weight concept for model-driven composition by attaching additional attributes to EPCs only. This allows to model the Web service composition as well as the user interaction. In this context model-driven means that developers create models instead of source code. These models are then used to create executable code.

In contrast to established approaches complete applications can be modeled with less effort. Therefore, even SMEs who cannot invest heavily into information technology can profit from the advantages of Web service technology.

1 Introduction

With the advent of Web services during the last years, software components can be remotely accessed via local networks as well as the Internet. Small services provide clients with specific functions. These can be invoked using standardized protocols, like Simple Object Access Protocol (SOAP) [GHM$^+$03]. The goal is to create an infrastructure allowing business applications to transparently discover and use Web services. Thereby, the integration of different applications and the development of distributed applications will be made easier. This architecture, called Service Oriented Architecture (SOA), provides a transparent environment in which applications are composed out of services.

Some hopes and visions are associated with SOAs. For example, Enterprise Application Integration (EAI), i.e. the seamless connection and data exchange between different systems in an enterprise, is mainly based on using Web service standards. While EAI has become an objective for larger enterprises due to the huge number of deployed systems, small and medium sized enterprises (SME) still have problems supporting their business processes with integrated IT systems.

SMEs compete against larger corporations utilizing their flexibility and their ability to innovate. In order to further increase their opportunities, these SMEs need to deploy ERP systems to support their business processes. But to stay as flexible and competitive as today, SMEs would have to customize their ERP system each time the business processes change. However, ERP systems are complex and their customization as well as maintenance are costly. Therefore, SMEs often do not have the financial resources to deploy and maintain powerful and large ERP systems.

To fill this gap, cheaper ERP systems with less functionality have been offered and the concept of Application Service Providing (ASP) has emerged. However, both solutions have their drawbacks: ERP systems offering less functionality do not realize all possible opportunities and do not address maintenance costs. Even worse, ASP, i.e. the operation of systems by a third-party in an external data-center, has been rejected because enterprises are not willing to store their valuable data externally and the distribution of responsibilities creates management problems [Wal03]. Therefore, a solution is needed which combines local data management with reduced costs and flexible support for changing and optimizing business processes [LGS05].

We address this problem with an ERP system whose logic is completely composed of Web services. These are dynamically arranged to support the company's business processes [KGRL04]. Such an ERP system has the advantage of storing all relevant data in-house as well as being extensible by integrating as many Web services as required for realizing the desired functionality.

The main focus of this paper is the light-weight composition of a set of Web services to a fully functional application by directly using business process descriptions. For a fully functional application not only the logic but also the user interface has to be composed as well. For modeling the application's processes Event-Driven Process Chains (EPCs) are used.

The presented ideas are subject of our current research. The user interface part has already been implemented, demonstrating the technical merits of our approach and serving as a proof of concept. Composition of Web services and data management will be implemented next.

This paper is structured as follows: After presenting the related work on this subject, the paper discusses possible advantages of using EPCs in the given problem domain. In the main part, the concepts for composing Web services and user interfaces are explained. Afterwards, a small example is given. The paper closes by describing future research problems and challenges before a conclusion is given.

2 Related Work

As stated in the introduction, the goal of our approach is the light-weight design of business applications as a whole using a model-driven concept. This includes the generation of the user interface needed to support human interaction with the IT system. However, EPCs are only used for the modeling and visualization of business processes. Thus they must be extended with additional information, allowing for the modeling of the Web services' flow and the user interface.

From the field of *Business Process Modeling*, research into EPCs [KNS92] is valuable to our approach: Since the main purpose of EPCs is to model business processes, some properties are missing to directly use EPCs as a workflow language. However, there has been research on how EPCs can be used as a workflow language [Deh02] and which extensions are needed to pass all workflow patterns [MNN05]. Other research has been done to precisely define the semantics of EPC models [Kin06]. Furthermore, proprietary standards have been developed, e.g. by IBM [LSZ01] to combine business process management and Web service development.

In the field of *Web Service Composition* the predominant standard is the Business Process Execution Language for Web services (BPEL4WS, WS-BPEL, BPEL) [ACD$^+$05] which allows the composition of Web services but does not incorporate any user interaction. Furthermore, there has been research on the architectural and management issues of Web service-based applications, for example by Ardissono et. al. [ACPS04] and Anzboeck [ADG02]. On the subject on how to synchronize business processes and Web service composition there has been much research as well, e.g. repository replication by Terai et. al [TIY03]. Furthermore, there are possibilities to map BPEL descriptions back to EPCs [MZ05].

Related to *user interface Design and Generation* is the attachment of user interface information to business processes. The idea of capturing the core design of a user interface in an abstract model is not new and has been researched actively in the Model-Based Design of user interfaces (MB-UI) for more than a decade [Pat99]. Numerous design environments have been proposed, each differing in the number and type of models used (for a thorough overview the reader is referred to [dS00]). The *task model*, commonly found in all approaches, is tightly connected to our project: The business process model is in fact a task model on a very high abstract level. This is elaborated in [Træ99], where the author shows that both models share the same basic components.

Criteria which are required for successful acceptance of model-based techniques by practitioners and problems MB-UI techniques have faced in the past are listed in [TMN04]. Especially their complexity hinders their application. Therefore, our approach particularly strives to reduce the inherent complexity.

3 Using Business Processes as a Modeling Tool for ERP Systems

The main users of our envisioned ERP system are SMEs as they often optimize their business processes. They should themselves be able to customize the software as easily and cheaply as possible.

We assume SME personnel to be able to understand and edit simple business process notations. Business Processes are often modeled in Business Process Languages, i.e. special notations suited to be comprehensible by business process designers, IS specialists and many economists. These notations are a very good foundation to build a common understanding between all involved parties.

In SOA-based applications services are composed. In particular applications need to know when and how a service should be executed.

Such compositions can be implemented by using workflow systems. They allow to describe an executable process whose activities can be service calls. Special languages like BPEL have been developed in order to make Web service composition easier. Composition models are often refinements of business processes. After special composition languages have been used, the extraction of a pure business process view is difficult. The synchronization between business process models on the one hand and the dependent composition models on the other hand is a real challenge. Research has been done on how to make the transition back to business process models easier [MZ05] or how to replicate between both repositories [TIY03]. But much effort would be saved if it was not necessary to synchronize at all. Instead a unified repository in which changes to the underlying business processes would directly change the composition model as well would be a better alternative.

For our design of an ERP system suited for SMEs the aim is such a unified repository. Event-Driven Process Chains (EPCs) are the foundation of our approach: They are extended with necessary attributes for generating a working software system. The system shall support the corresponding business processes and use Web services for embedding the application logic into the system.

In order to be useful, an application needs to be operated by end users. In contrast to BPEL we decided to model the user interaction directly in the business process as well and generate screen masks out of the business process repository.

The necessary extensions can be organized in different views on the EPC model: There can be views for traditional EPCs, for composition properties, for requirements and so on. EPCs are well-suited for this task because many views are just hierarchical refinements of traditional business functions. For example, steps in the user interface correspond to steps within a business function.

Consequently, our approach tries to attach necessary information for generating applications as properties to EPC models. Therefore, we divide the application into three layers:

- *Presentation Layer*: In this layer the user interface is generated from the extended EPC attributes.
- *Process Layer*: The process layer is responsible for the composition of the Web services, to organize the application's workflow and data management.
- *Web Service Layer*: The application logic is composed out of Web services which are offered on the Internet and in the local network.

In the next two sections we present the additional properties needed for Web service composition and UI generation.

4 Attaching Web Services to Business Functions

When composing Web services within the given scenario of SMEs, the composition should be easily understandable and changeable. Since EPCs can foster a common understanding, it would be helpful to use EPC models in order to do the composition.

For composition we differentiate Web services depending on their granularity:

- *Business Service*: This kind of Web service can be directly invoked to execute the program logic covering a whole business function in an EPC. This means one business function can be related directly to one Web service call.
- *Sub-Business Service*: All other services which provide a corresponding business service interface, such as technical support services like data querying etc. and business logic services which contain business logic but not on an abstraction level high enough to directly support a business function without requiring additional help mechanisms like transaction management.

In the following the composition options of each Web service type will be discussed.

4.1 Composition of Business Services

If a Web service contains the required logic of a business function, there is a 1:1 mapping between the service and the function. Therefore, the Web service description, for example as a WSDL document, can be attached to the business function.

Furthermore, the Web services' input and output need to be managed and saved. In BPEL this is realized by transformations which clutter the diagrams. Furthermore, these transformations are technical details and therefore only interesting for IT experts but neither for economists nor business process designers. Because of this we decided to attach input and output XSL transformations to the business function as attributes. These transformations can access the business objects attached to the business function according to the eEPC notation. Because we require each business object to have an XML Schema, XSL can convert between XML objects and the XML document containing the Web service parameters and vice versa. The XSL input and output transformations as well as the WSDL document would be the functional equivalent to BPEL's partner links and port types.

This concept achieves to store all additional information needed for Web service composition and execution as properties of business functions. No additional symbols have to be introduced in the EPCs although we recommend to mark business functions. Such markers are intended for people interested in the compositional and IT-related aspects. We propose to use three markers for a business function: (1) for business functions which are automated by Web services, (2) another one for those that cannot be automated by software and a third for those (3) that are unknown. The last marker is like a to-do-marker which signals that the Web service composer needs to inspect the business function and

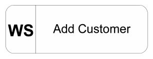
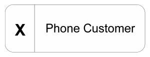
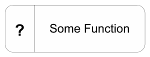
Name	**Description**	**Notation**
Web Service	A given Web service can be used to achieve the business function, e.g. ”Add customer” or ”Cancel Order”.	
Manual Execution	The business function cannot be executed but need to be done manually, e.g. ”Send product to customer” or ”Phone Customer”	
Unknown	The business function has not yet been inspected and it is not known whether or not it can be automated by a Web service.	

Table 1: Marked Business Functions for Web service composition

decide whether to use a Web service or to keep the manual execution. The marker types are shown in table 1.

We believe that Business Services will be the predominant form of Web services for the following reasons:

- *Semantic*: Web services are normally developed to support a business process. Therefore the software designers will mimic the semantic of the business process in their design. This leads to top-level interfaces reflecting the business functions. Good examples of accepted design practices leading to such a design are the Facade Pattern, Command Pattern [GHJV95] and Use Case/Front Controller Pattern [ASP02, AMC03].
- *Performance*: Web services are remote components. Each remote call is very costly in terms of performance. Therefore, design focuses on minimizing remote calls leading to coarse-grained interfaces of remote components [Fow02].
- *Ease of Composition*: Composition is supposed to be easier with coarse-grained Web services. Since this practice has been propagated for years (e.g. by Hanson [Han03]) it will influence designers' decisions.

Because of these reasons we believe EPC-only composition will be possible most of the time.

4.2 Composition of other Services

Sometimes Web services, that are too fine-grained to be composed with EPCs, need to be included in the composition model. In this case, more advanced composition options are needed. Introducing all these capabilities into EPCs would destroy their main advantage: The easy notation with three main elements (function, event, connector) would vanish. Furthermore, it would duplicate the effort already undertaken, because BPEL was developed with exactly that support in mind.

Therefore, we propose a two-layered approach for composing finer grained services by combining EPC's and BPEL's strengths:

1. Fine grained Web services are composed using BPEL. This allows maximum flexibility. The result is a Business Service.
2. The new Business Service is integrated into the global Web service composition modeled with EPCs. This still allows a good overview and a reference for discussion with all responsible people in an enterprise.

Smaller services are therefore composed outside the EPC model thus hiding complex composition logic from the top-most view. The business process model can still be changed by business process designers. Independently, experts can fine-tune details in the Web service composition.

4.3 Composition Architecture Summary

Our general Composition Architecture as shown in figure 1 is based on EPCs for composition of the Web services. The EPCs and their execution engine form the Process Layer. The Process Layer calls Web services located in the Web Services Layer. This layer consists of Web services divided into Business Services and other services. Non-Business Services will be composed using BPEL in order to form a Business Service which can be composed using the Process Layer.

The clear distinction between Process Layer and Web service layer allows different roles to fulfill process-related and technical tasks. The EPC notation is understandable for non-IT experts; Web service development and more demanding composition are separated clearly. These tasks can be done by technically skilled people.

In the context of SMEs this means that the general workflow can be changed by the enterprise itself. In contrast the business services and more complex compositions can be developed and maintained by a third party like ISVs etc.

In chapter 6 we present a short example on how to compose a business process using our concept.

5 Generating and Controlling the user interface

In the sections above the lower two layers of the envisioned ERP system were presented. If a process change occurs on the middle layer, only the presentation layer is rendered

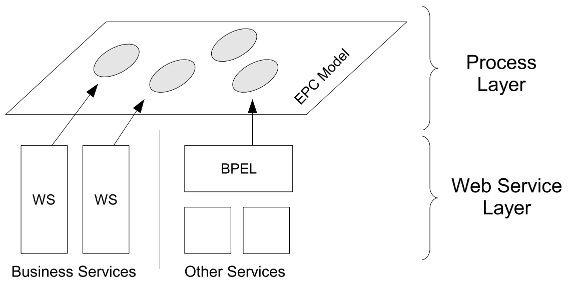


Figure 1: Overview of the Composition Architecture

useless: This is due to the occurring inconsistencies between the process model and the control flow implicitly defined by the user interface. It is crucial that the presentation layer is kept up-to-date according to the business process changes to preserve the flexibility of our approach.

Our goal is to semi-automatically generate the user interfaces by adding additional information to the process model. The following requirements of the user interface can be identified and mapped to the correspondent level, where they should be dealt with:

1. The cooperation between different users of the ERP system has to be handled because many users can be involved in the same business process. This is a distinguishing feature of this kind of application, as normal standalone applications are only controlled by a single user. The cooperation is already modeled by the business process and merely has to be supported by our system.
2. The activities in the business processes have to be modeled in such detail that a screen mask for a single user interface can be inferred from the extended information. The business functions requiring interaction with a user can be seen as generalized descriptions of these activities, for which a more detailed task model has to be designed. As we will see, our proposed task model remains on a high abstraction level in comparison to others [dS00], which reduces the inherent complexity.

5.1 Business Process Layer

The separation of concerns to different layers of the model as described above can be made explicit by the use of hierarchic functions in the EPC model. Events form the implicit interface between both model layers as they are shared by both the business process and the task model. This interface is used to implement the cooperative aspect of the system. It easily translates into an exchange of events by the ERP system among the different clients: If an event has occurred in the process layer, all affected users have to be notified. To this end, roles are assigned to each user and to the steps in the business processes. If an event is triggered, all affected users can be notified and the event is forwarded to their client computers. Given a well documented EPC, these events can be displayed appropriately to the user in the user interface. If all necessary preconditions are fulfilled, the user can afterwards start the subprocess (i.e. the task model) and control it using his client application. Once the user finishes the subprocess, the server is notified of the evoked end events. This in turn may lead to the notification of other clients and eventually to the execution of following processes and task models. EPCs are well-suited to model the cooperative aspect of ERP systems.

5.2 Task Model Layer

Modeling the cooperation on the business process layer, the designer can focus on modeling the activities of a single user on the task model layer.

Tasks are activities required to reach a certain goal. A single task is always assigned to a single goal. A task model is a composition of tasks, defining their temporal and conditional relationships. For instance the task model specifies in what order tasks are to be performed, i.e. if one task necessitates another, or if tasks can be executed independently of each other. A task model defines a so called *structured* task, which subsumes the goals of its subtasks to a global goal. Structured tasks can then again be used in other task models. Therefore, a task hierarchy is established.

EPCs are a task model, with the notion of a task translated to the notion of a function. The hierarchical relationship mentioned above is easily modeled by the use of EPC's hierarchical functions. However, EPCs reside on a very high abstract level with functions denoting quite complex activities. In order to generate a user interface from an EPC, its level has to be lowered by decomposing each abstract business function into more detailed tasks.

The detailed information is given by assigning each function or task to a certain type. The user interface generator uses this type in order to construct a component capable of supporting it. Comparing the various proposed task models, a common set of tasks can be identified, which seems to be of elementary importance. We have adopted these tasks in our approach and transferred them to an EPC notation as listed in table 2.

Tasks are connected to objects which need to be manipulated or retrieved. These are the domain of the task. In our case, the domain is well-defined by the use of Web Services and the representation of data by an XML Schema. In other models the objects handled by the various tasks are sometimes as fine grained as a String object. The trade-off is between flexibility to model the user interface on the one hand side and the complexity and size of the model on the other hand side. We decided to define the tasks on a higher abstraction level, because the complexity greatly hinders the acceptance of the approach. Each task is assigned an information object, which is part of the whole process's XML data and is defined by a complex XML Schema type. Due to the well-structured nature of XML Schema, this information can be used to generate basic user interfaces supporting the execution of the task types. This approach is similar to [LLK04].

XML Schema is recursively built from primitive types, like strings or integers, which are defined in its specification. These types can either be used to derive other simple types by

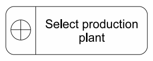
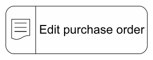
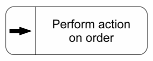
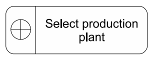
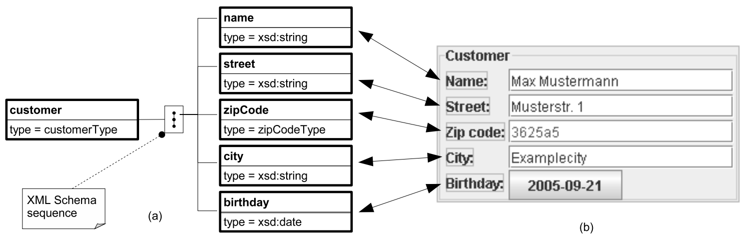
Name	Description	Notation
Selection	The user selects data from a collection of possible choices.	
Edit	The user edits some information object from the data model.	
Control	The user explicitly invokes some action. This is used to model navigational decisions.	
User	The user has to do something by himself, e.g. planning, comparing, etc.	

Table 2: Task types

constraining the range of possible values (like only positive numbers) or they can be used to compose them in a content model, which is either one of `all`, `sequence` or `choice` (like address containing name, street, city, ...).

In order to construct a user interface, each primitive type is mapped to a well-suited editor. The XML Schema's constraints can either be used to limit the editor itself, or be used as a validation rule to check the user input. If the entered value is out of range, this can be reported to the user. A short example of the whole transformation is given in figure 2. The assignment of the editors to the different primitive types can be supplemented with a template system, which allows the user or a whole organization to choose the representation most fitting for a certain type.

If editors are available for all simple types, they can be composed in the same way as specified in the XML Schema. In case of a `sequence`, the translation is the straightforward chaining of the editors. The XML Schema content models `all` and `choice` are represented by the use of check boxes, drop down lists or tabbed panes. All editors can be recursively composed with each other in a bottom-up approach.



Name: Max Mustermann
Street: Musterstr. 1
Zip code: 3625a5
City: Examplecity
Birthday: 2005-09-21

Figure 2: Generation of a simple mask (b) for a customer data type (a).

The result is a basic user interface component for each task in the task model. The events in the process represent the possible results of a task. Each event can hold a condition evaluating the task's output to determine the path taken at splits. Though this may seem to contradict the passive nature of events, the data itself is produced only in the functions. Raising the tasks on the high abstraction level as we have done, results in a task model which resembles very much the controller in the MVC framework [KP88] (more specifically the Model 2 architecture used in Web applications). Hence, the task model descriptions can be used as a back-end of a generic controller, which is responsible for (a) selecting the following view, (b) managing the data model , and (c) invoking the underlying services.

A minor problem arises with (a) when we consider AND/OR splits in the EPC model. In the traditional interpretation such a split means that the following paths can be executed independently of each other. Thus, the generic controller can arbitrarily handle the paths one after another, which results in a sequence thereby removing the split. However, in the context of task models the designers might intend to model synchronized tasks, which depend on each others information. In this case the controller has to merge both associated user interface components.

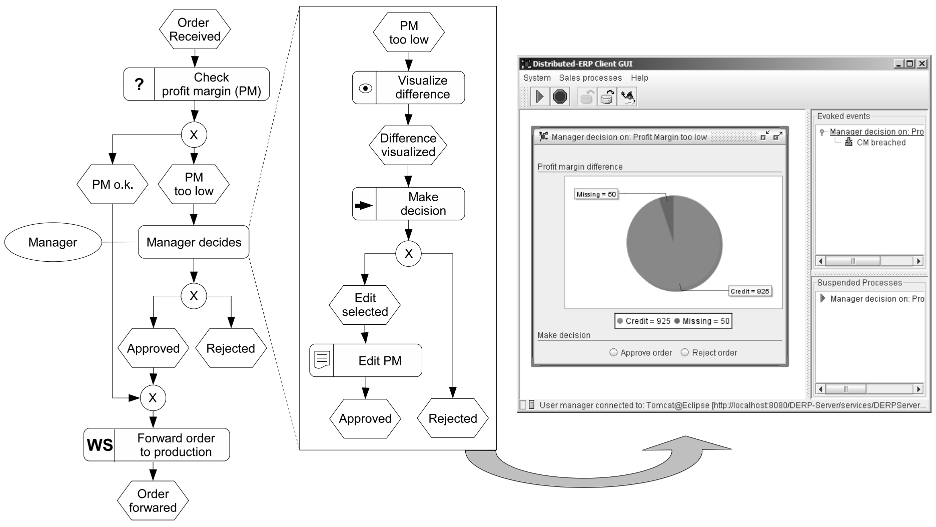
The user interfaces generated above are constrained in two dimensions: Firstly, a task model is a decomposition of an abstract business function. Thus, the User interface is goal-oriented in its nature, because all tasks have a common overall goal, namely that of the abstract business function. This corresponds to a wizard-style interface with a linear control flow, guiding the user towards that goal. Secondly, the basic user interface components are constrained to the expressiveness of XML Schema. Consequently, structures which cannot be expressed by XML Schema cannot be manipulated by the user interface.

The generated interfaces seem to be simple in nature. However, they are still sufficient for the domain of ERP systems, where most screen masks resemble simple forms for textual or numerical data. Their simplicity furthermore reduces the complexity of modeling the user interface. The latter might prove to be crucial for the adoption by practitioners, as the learning curve is not as steep as in other approaches.

6 Example: Supporting a Order Reception Application

To illustrate the concept a small example is given in this chapter: A small company takes orders from its customers, who can request a special price they are willing to pay. Thus, the profit margin of each order has to be checked. If it is not in accordance with the company's strategy, a manager has to decide whether the order will be accepted or not. If it is accepted, the profit margin has to be adjusted. We define the process using our extended EPC notation as shown in figure 3.

Starting with the incoming order, an employee chooses an ”Receive Order” item within its user interface. This issues the EPC's start event and the margin is checked. This action is automated by a Web service as can be seen by the business function's symbol. The result will be some change to the underlying data model. The following events specify mutual



User manager connected to: Tomcat@Eclipse [http://localhost:8080/DERP-Server/services/DERPServer...

Figure 3: Extended EPCs for an Order Reception Application

excluding conditions on the model, so the path taken at execution time can be determined. If the profit margin is too low, the manager has to handle the order.

The correspondent business function is decomposed into a task model, from which a user interface can be generated. If a *"Margin too low"* event occurs, the manager's client application receives an event and the task model is executed. First, the difference between the expected margin and the one from the order is visualized. This can either be a simple textual display in the user interface, or the company could specify a special editor like a chart for this task. Based on the difference, the manager must make a decision, which is modeled by a Control task. If the order is not rejected, the manager must update the profit margin in an Edit task. If the order is approved, it is forwarded to the production facility, which can be automated by a Web service call.

7 Future Work

This paper presented a concept on how to compose Web services to a fully functional business application including the user interface. The user interface part has already been implemented in an application server. The Web service composition is implemented rudimentarily. Our system is able to call Business Services. We will extend the application server with additional functionality for dealing with Web services: Fail-over support, load balancing, selection of appropriate Web services are the next logical steps.

Other extensions will be security and transaction management. However, these extensions will require their own properties, e.g. which user roles may access various business functions or where transaction boundaries are located. The more properties get attached to EPCs, the more an EPC editor is needed, which can show views suited for different stakeholders in the underlying EPC model. For example, business process designers could be interested in an EPC notion, while IT department managers would like to see which software supports certain processes and by which users it is accessed.

On the management and requirements side, there are open issues on how to transfer the business requirements as smoothly as possible into a business process model. Our goal is to integrate known and proofed requirements engineering concepts of the Software Engineering world, like Use Cases [Coc00], into business process models. As with traditional Requirements Engineering this is mainly an organizational aspect.

On the theoretical side, the EPC model with the new properties for Web service and user interface composition needs to be formalized. This includes extensions to EPML (EPC Markup Language) [MN05]. We have preliminary support for these extensions since we use EPML to store the business process models on the application server.

8 Conclusions

In this paper we described a concept for using EPC models in order to design the service composition and user interface of business software. EPCs have the advantage of offering a simple notation which can be easily extended. If combined with Web services the step from business processes to the service composition model can be made easily, thus offering a unified modeling environment for business software. While several other technologies exist for composing services, we added user interface generation. For accomplishing this task we describe the task model of the user interface with the EPC notation of functions and events. This notation is sufficient to describe ERP user interfaces having a mainly linear control flow.

Combined, these technologies provide a solid foundation for ERP systems well suited for SMEs. They allow flexible and rich functionality by easy integration of Web services while providing local data storage and easy customization. These strengths will hopefully be further improved by the results of the open research questions outlined in the future work section.

References

[ACD$^+$05] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *Business Process Execution Language for Web Services version 1.1*, February 2005.

[ACPS04] Liliana Ardissono, Davide Cardinio, Giovanna Petrone, and Marino Segnan. A framework for the server-side management of conversations with web services. In *Proceed-*

ings of the 13th international World Wide Web conference on Alternate track papers & posters, pages 124–133. ACM Press, 2004.

[ADG02] Rainer Anzböck, Schahram Dustdar, and Harald Gall. Software configuration, distribution, and deployment of web-services. In *SEKE '02: Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 649–656, New York, NY, USA, 2002. ACM Press.

[AMC03] Deepak Alur, Dan Malks, and John Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall PTR, 2nd edition edition, June 2003.

[ASP02] Ademar Aguiar, Alexandre Sousa, and Alexandre Pinto. Use-Case Controller. WWW: http://se2c.uni.lu/tiki/se2c-bib_download.php?id=554, 2002.

[Coc00] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, 2000.

[Deh02] Juliane Dehnert. Making EPCs fit for Workflow management. In Markus Nüttgens and Frank J. Rump, editors, *EPK 2002 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten*, pages 51–69. Gesellschaft für Informatik e.V. (GI), November 2002.

[dS00] Paulo Pinheiro da Silva. User Interface Declarative Models and Development Environments: A Survey. In Philippe A. Palanque and Fabio Paternò, editors, *DSV-IS*, volume 1946 of *Lecture Notes in Computer Science*, pages 207–226. Springer, 2000.

[Fow02] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 1st edition, November 2002.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1st edition, January 1995.

[GHM$^+$03] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, and Henrik Frystyk Nielsen. SOAP Version 1.2 Part 1: Messaging Framework. Technical report, World Wide Web Consortium, 2003.

[Han03] Jeff Hanson. Coarse-grained interfaces enable service composition in SOA. WWW: http://builder.com.com/5100-6386-5064520.html, August 2003.

[KGRL04] Oliver Krüger, Jorge Marx Gómez, Claus Rautenstrauch, and Daniel Lübke. Developing a distributed ERP system based on Peer-to-Peer-Networks and Webservices. In Jorge Marx Gómez, editor, *Proceedings of the Workshop for Intelligent Mobile Agents in Peer -to- Peer Networks, EIS 2004*, 2004.

[Kin06] Ekkart Kindler. On the semantics of EPCs: Resolving the vicious circle. *Data & Knowledge Engineering*, 56(1):23–40, January 2006 2006.

[KNS92] G. Keller, M. Nüttgens, and A.-W. Scheer. Semantische Prozeßmodellierung auf der Grundlage "Ereignisgesteuerter Prozeßketten (EPK). Number 89 in Veröffentlichungen des Instituts für Wirtschaftsinformatik. Scheer, A.-W., 1992.

[KP88] Glenn E. Krasner and Stephen T. Pope. A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. *J. Object Oriented Program.*, 1(3):26–49, 1988.

[LGS05] Daniel Lübke, Jorge Marx Gómez, and Kurt Schneider. Serviceorientierte Architekturen und Prozessmanagement Chancen durch die Verbindung zweier Welten. *ERP Management*, (3), September 2005.

[LLK04] Patrick Lay and Stefan Lüttringhaus-Kappel. Transforming XML Schemas into Java Swing GUIs. In Peter Dadam and Manfred Reichert, editors, *GI Jahrestagung (1)*, volume 50 of *LNI*, pages 271–276. GI, 2004.

[LSZ01] Peter Lambros, Marc-Thomas Schmidt, and Claudia Zentner. Combine business process management technology and business services to implement complex Web services. Whitepaper, IBM, May 2001.

[MN05] Jan Mendling and Markus Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). Technical report, Vienna University of Economics and Business Administration, March 2005.

[MNN05] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. Towards Workflow Pattern Support of Event-Driven Process Chains (EPC). In Markus Nüttgens and Jan Mendling, editors, *XML4BPM 2005 - XML Interchange Formats for Business Process Management*, pages 23–37, 2005.

[MZ05] Jan Mendling and Jörg Ziemann. EPK-Visualisierung von BPEL4WS Prozessdefinitionen. In *Proceedings of 7th Workshop Software-Reengineering*, May 2005.

[Pat99] Fabio Paterno. *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag, London, UK, 1999.

[TIY03] Koichi Terai, Noriaki Izumi, and Takahira Yamaguchi. Coordinating Web Services based on business models. In *ACM International Conference Proceeding Series: Proceedings of the 5th international conference on Electronic commerce*, volume 50, pages 473–478. ACM Press, 2003.

[TMN04] Hallvard Trætteberg, Pedro J. Molina, and Nuno Jardim Nunes. Making model-based UI design practical: usable and open methods and tools. In Jean Vanderdonckt, Nuno Jardim Nunes, and Charles Rich, editors, *Intelligent User Interfaces*, pages 376–377. ACM, 2004.

[Træ99] Hallvard Trætteberg. Modelling Work: Workflow and Task Modelling. In Jean Vanderdonckt and Angel R. Puerta, editors, *CADUI*, pages 275–280. Kluwer, 1999.

[Wal03] Kenneth R. Walsh. Analyzing the application ASP concept: technologies, economies, and strategies. *Commun. ACM*, 46(8):103–107, 2003.

Structuring Business Nested Processes Using UML 2.0 Activity Diagrams and Translating into XPDL

Barbara Gallina and Nicolas Guelfi and Amel Mammar
Software Engineering Competence Center
Faculty of Sciences, Technology and Communication
University of Luxembourg, L-1359 Luxembourg-Kirchberg, Luxembourg
{barbara.gallina, nicolas.guelfi, amel.mammar}@uni.lu

Abstract: Engineering complex distributed business processes necessitates an integrated use of modeling, verification and validation techniques. This paper presents a pragmatic approach for such purpose. In particular we present complex business processes modeled using transactions described with UML 2.0 activity diagrams, in which we introduce hierarchy as a structuring primitive and exception handling as a fault tolerance mechanism. The structuring capabilities of our approach allow designers to tackle the complexity of large business transactions, the exception handling mechanism introduces a novel way to design transactions and to cope with exceptional behaviors inherent to BPM. Since our transaction models need to be validated during the development process, we have chosen to integrate the modeling tool together with a workflow management system that will allow animation of the business transactions. This is done by model transformation from activity diagrams to XPDL descriptions. One of the other interesting aspects of our approach is that the UML 2.0 activity diagrams models are given in such a way that different execution environments can be targeted for deployment. These environments (MTS, CORBA, BTP,) differ mainly on the way they implement the ACID properties and on their underlying exception handling mechanism. A first prototype supporting this approach, but limited to XPDL 1, has been developed and we also present its possible extension to the new XPDL 2 standard.

Keywords UML 2.0 Activity Diagrams, XPDL, sub-flows, exception handling, ACID properties.

1 Introduction

This work is part of the Efficient research project of Henri Tudor research center (Luxembourg) in cooperation with our university. The project presents a three layered approach: a business, a specification and technical layer. At the business layer, transactions, modeling business processes, are identified; at the specification layer, these transactions are specified using UML; at the technical layer, the execution of the transactions, trough an adequate tool called Animator, takes place. Animator is a toolset for designing, animating, validating and verifying trusted business transactions [DG01, Tud05]. At the specification layer, UML 2.0 Activity Diagrams (AD) have been chosen since we think they could represent a reasonable communication trade off between business logic experts and IT experts in order to describe trusted business processes. Business logic experts simply need to express functional requirements of the system which they are thinking of. For IT experts, however, every requirement has to be verified and consequently the use of formal methods is important.

XPDL is a standardized language allowing process definitions interchange between a variety of tools ranging from workflow management systems to modeling and simulation tools. In the Efficient context, XPDL is the language used at workflow engine level in order to execute and animate processes to be able to validate them.

Several dimensions have to be considered in order to think about a complete notation for describing business processes. As deeply argued in [RtHEvdA04, vdAtHKB00], several typical design patterns characterise business processes. In this work however we are going to consider basic elements and patterns necessary in order to reach our goal: structuring complex distributed business processes, modeling them through UML 2.0 activity diagrams and translating them into XPDL in order to be able to validate their design through animation.

The rest of this paper is structured as follows. In section 2, some relevant workflow process peculiarities are presented; in section 3, a typical business workflow scenario is described using UML 2.0 activity diagrams. XPDL is briefly introduced in section 4 while in section 5, the translation rules are informally presented . Finally, in section 6, conclusions and future work are presented.

2 Basic workflow elements

A workflow process is defined as the automatic routing of documents to the users responsible for working on them. Workflows are concerned with providing the information required to support each step of the business cycle. These documents may be physically moved over the network or maintained in a single database with the appropriate users given access to the data at the required times. Triggers can be implemented in the system to alert managers when operations are overdue. Similarly, in Efficient project context, a business transaction is defined as the description of an exchange of a set of information to achieve a business goal, mostly the delivery of an output to a particular customer or market. The basic elements that constitute our business transactions are described in the following list.

- Tasks/activities may be elementary or composed of a set of activities. In case of composed activities hierarchical levels among them may be recognised. Processes are composed of activities that have to be executed in order to reach a goal.
- Documents/objects represent administrative documents, credit cards and so on. Process tasks are based on operations that are executed on objects.
- Decisions and parallelisms are quite crucial points in business processes since they allow business expert to express alternative (but still normal) paths and parallel paths.
- Exceptions are nowadays deeply investigated at both levels: modelling and workflow engines. Since it is hard to foresee in an exhaustive way the entire spectrum of

situations that could happen during the process execution, it is better to avoid freezing a system by trying to define all possible situations. Better is to define particular circumstances that diverge from the normal ones and try to define a behaviour that is useful to follow in those cases.

- Roles/participants represent the entity that is going to execute the process. Generally a role may be a person, a machine or a particular software. Sometimes the process execution may need the participation of more roles of different typologies at the same time. Roles (business partners) may cooperate in order to reach the transaction goal in normal and exceptional cases.

As far as we know, not all these elements have been already taken into consideration. In particular untill now, among possible tasks, only elementary ones have been considered; moreover only normal process behaviors have been modeled. The contributions of this work are the consideration of composed activities and a first theoretical step to investigate how to represent transactions and exception handling mechanism in UML 2.0 activity diagrams and also in both XPDL 1.0 and XPDL 2.0.

3 UML 2.0 activity diagrams for nested business processes: a typical case-study

Activity diagrams, as means to describe business processes, have been chosen also in other previous works [GCR04]. We still choose activity diagrams since we think that they can be an efficient communication trade-off between business experts and IT experts but we focus our attention on UML 2.0 activity diagrams because of their more powerful expressiveness compared with those of UML 1.5.

In UML 2.0 substantial changes have been made [Obj03] if compared with UML 1.5, in particular the activity diagram metamodel subset has been redesigned from scratch. The main concept underlying activity diagrams is now called *Activity* and replaces *Activity-Graph* in UML 1.5. *Activity* is not a subclass of *StateMachine* any more. The metamodel defines six levels of increasing expressiveness. Since our aim is to express nesting business transactions in a distributed context and exception handling and since business transactions imply the exchange of massages, *IntermediateActivities* level is involved. In fact this level supports modeling of activity diagrams that include concurrent control and data flow. Moreover in order to use partitions we also use *CompleteActivities*.

SubactivityStates have vanished, and nesting is now accomplished by calling subordinate (enclosed) Activities from Actions defined in the enclosing context. These enclosed activities in UML 2.0 terminology are called *CallBehaviorAction*. An AD is a directed graph, consisting of nodes that are connected via directed edges. Nodes comprise action nodes, object nodes and control nodes.

Action nodes, as already mentioned, may invoke other behaviours and the related behaviour is still an activity diagram. All actions may receive parameters as input and return parameters as output that are subclasses of object nodes.

In this work, instead of extending UML 2.0 activity diagrams by adding a stereotype for each XPDL activity type, as done in [JMN03], the effort is that to exploit as much as possible the standard AD elements and adding stereotypes only where they are necessary. Currently, however, UML doesn't support transactional concept. We mean that if we want to underline ACID properties at the moment we can not. In order to do that we have to stereotype an existing symbol by changing its semantics.

We are going to describe our extension by briefly re-introducing a typical business case-study (already illustrated in [GCR04]) involving a customer, a wholesaler and a manufacturer, detailing how ACID properties and exception handling mechanism (both forward and backward error recovery) could be designed. A business process in order to be modeled as a transaction should ensure ACID properties. As already mentioned, in UML 2.0 there is no way to express the transaction concept and in this work we are interested in doing a first step towards it because in order to talk about efficient capture of transactions, first of all, we have to ensure transactions quality. We then propose to model a nested process through a CallBehaviorAction Metaclass with stereotype Transaction in cases in which the underlying protocol assures (relaxed or not) ACID properties. In all other cases a nested process will be modeled through a CallBehaviorAction without any stereotype. A transaction may terminate normally (according to the expectations), exceptionally or aborting the effects. In cases in which an hazard happens we propose a way to also model this circumstance.

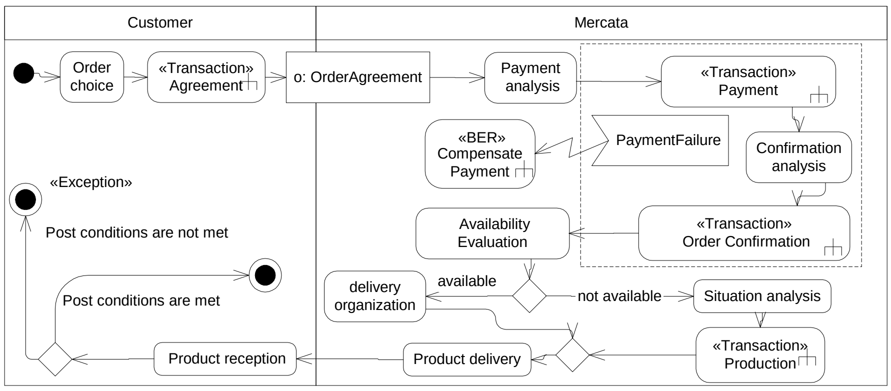


Figure 1: Highest level diagram of Mercata Case-Study.

In Figure 1 a business process, depicted through a UML 2.0 activity diagram, is presented. There are two roles Customer and Mercata. A customer after having chosen the order and agreed (successfully executed the sub-process Agreement) about the order details, sends the order to Mercata. The Payment process, a new sub-process, takes place. During this sub-process an exception may occur. This possibility is depicted by drawing an interruptible region around the part in which the exception is supposed to happen. If during Payment sub-process a Failure Exception occurs and we must roll-back the situation, apply Backward Error Recovery (BER), because we have no means to go on in another way, a special sub-process, a UML 2.0 CallBehaviorAction with stereotype BER, has to be called.

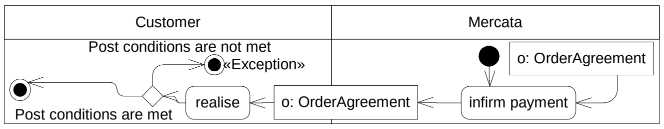


Figure 2: Compensation sub-process

In figure 2 the roll-back is shown: the payment is infirmed and the customer receives the notification. If Payment terminates successfully and the order is also successfully

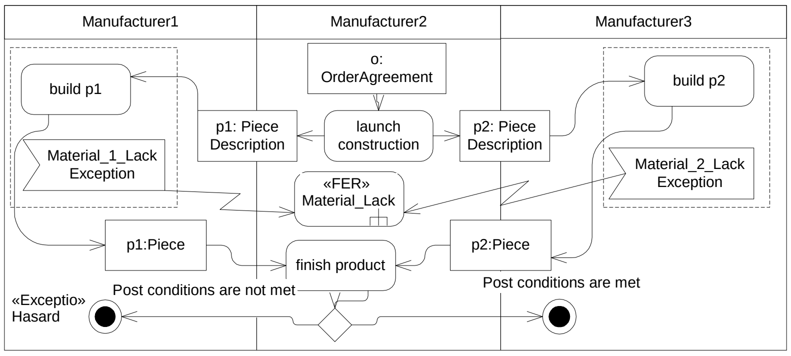


Figure 3: Production sub-process

confirmed, Mercata will deliver the product to the customer in case of availability or will ask for production. In case of production call, figure 3 shows how the manufacturers are going to work in order to satisfy Mercata request. Manufacturer2 receives the order and establishes who has to do what. Manufacturer1 and Manufacturer3 have to produce a part of the final product. In case of lack of material exception for both parts, the two exceptions are handled by consulting the exception tree, depicted in figure 5. Exceptions are represented trough a class diagram. Among exceptions a hierarchical relationship is underlined. Following this relationship it can be established which exception has to be handled in case of concurrency. Class diagrams may be mapped into XMLSchema and passed as parameters to processes. A general Material Lack exception has then to

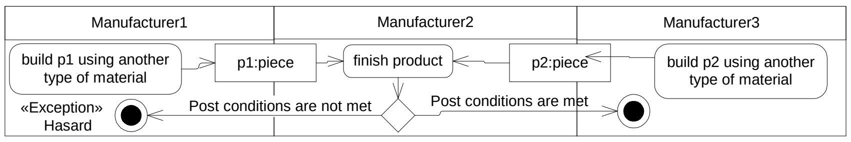


Figure 4: FER-Lack of Material.

be handled and a CallBehaviorAction stereotyped FER (Forward Exception Handling) is called. In Figure 4 the same roles of Production process cooperate in order to face the exception. If at the end of FER conditions are met, a consistent state will be reached.

This UML 2.0 extension seeks to be a first answer to the lack of ACID properties specification already mentioned in [SD05]. Its purpose is to provide basic building blocks that may be used in order to model advanced transactional protocols.

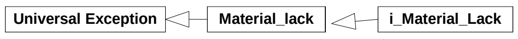


Figure 5: Class Diagram describing Excepion Tree.

4 XPDL: a standard workflow process definition language

The Workflow Management Coalition (WfMC) [WfM05] was founded in August 1993 as a international non-profit organization. The goal of the WfMC is to promote and develop the use of workflow through the establishment of standards to define workflow terminology, to satisfy interoperability and connectivity between workflow products. One of the main activities since 1993 has been the development of standards for these interfaces. The WfMC's reference model identifies five interfaces. Interface 1 is the link between the so-called "ProcessDefinition Tools" and the "Enactment Service". The Process Definition Tools are used to design workflows while the Enactment Service can execute workflows. The primary goal of Interface 1 is the import and export of process definitions. To support the interchange of workflow process definitions, XPDL has been proposed. XPDL [XPD02] uses an XML-based syntax, specified by an XML schema. The main elements of the language are: *Package*, *Application*, *WorkflowProcess*, *Activity*, *Transition*, *Participant*, *DataField*, and *DataType*. *Package* element is the container holding the other elements. *Application* is used to specify the applications/tools invoked by the workflow processes defined in a package. *WorkflowProcess* is used to define workflow processes or parts of workflow processes and it also consists of an oriented graph composed of nodes and edges. Nodes are activities while edges are transitions.

Activity is the basic building block of a workflow process definition. There are three types of activities: *Route*, *Implementation*, and *BlockActivity*. Activities of type *Route* are dummy activities just used for routing purposes. Activities of type *BlockActivity* are used to execute sets of smaller activities. Element *ActivitySet* refers to a self contained set of activities and transitions. A *BlockActivity* executes such an *ActivitySet*. Activities of type *Implementation* are steps in the process which are implemented by manual procedures (No implementation), implemented by one of more applications (called *Tool*), or implemented by another workflow process (*Subflow*). Activities may have *restrictions* on the incoming and outcoming transitions. A restriction on the incoming transitions is called "join", while a restriction on the outcoming transitions is called "split". Restrictions may have a type specification: AND or XOR. If an activity node has a join restriction type AND, then all incoming edges need to be present in order to start the activity. If an activity node has a join restriction type XOR, then the activity node is initiated when at least one of the incoming edges is taken.

Participant is used to specify the participants in the workflow, i.e., the entities that can execute work. There are 6 types of participant: ResourceSet, Resource, Role, OrganizationalUnit, Human, and System.

Elements of type *DataField* and *DataType* are used to specify workflow relevant data.

Data is used to make decisions or to refer to data outside of the workflow, and is passed between activities and subflows.

The XPDL 2.0 version has just been released [XPD05]. Relevant changes have been introduced in the new version. For example the possibility to represent messages exchanged among participants directly using *MessageFlow* entity instead of an indirect representation. Moreover the concept of transaction is finally introduced that means that an activity which is specified as transactional may terminate in three ways: normal, compensation or hazard. This is a step further towards the introduction of transaction and exception handling concepts and encourages our work in the same direction. In XPDL 2.0, in fact, a subprocess activity (implemented by Subflow) can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol (such as Business Transaction Protocol or Microsoft Transaction Protocol). In BTP protocol, for example, participants may use recorded (before or after) images, or compensation operations to provide the roll-forward, roll-back capacity which enables their subordination to the overall outcome of an atomic business transaction. It is possible that one of the participants may end up with a problem that causes a Cancel or a Hazard. In this case, the flow will then move to the appropriate Intermediate Event, even though it had apparently finished successfully. Since our research began before the final release of XPDL 2.0 and transaction model is still is an open issue (see 7.6.13 in [XPD05]), we focused on XPDL 1.0.

5 From UML 2.0 AD to XPDL

A transformation is the generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language [MG05]. In order to provide transformation rules between UML 2.0 AD and XPDL a deep knowledge on their expressiveness seems to be necessary. Their expressiveness may be evaluated by using the workflow patterns proposed in [vdAtHKB00, WvdAD+04, vdA03, vdA04]. In those papers a more intuitive notion of expressiveness is considered. The modelling effort is the criterion in order to evaluate the expressiveness: a great effort in modeling implies a less powerful expressiveness. We use some of the workflow patterns that result to be relevant in our project context in order to cover the basic elements presented in section 2. In cases in which there is no direct support for the workflow pattern and a workaround solution can not be sketched easily, the language under discussion results to be not powerful enough in terms of easy-modelling. In XPDL specification we can read: “using the basic transition entity plus dummy activities, routing structures of arbitrary complexity can be specified.“ In order to supply to the lack of direct support to some important patterns, this suggestion has to be followed. We present in the following, in a declarative way, the translation rules. Some of these rules are already in use in the context of Efficient project (see [EBD+03] and [Esh03] for more details). In this work, however, a deep study about the possibility to nest processes has been done. Moreover a first step towards the inclusion of exception handling mechanism and transaction concept has been done.

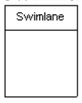
AD Construct	XPDL Construct
Swimlane. Each role is modeled through a swimlane. 	a default-value inside the Data Fields in XPDL or further investigation about the use of Participant type ROLE + participant element specification inside the activity. `</DataType><InitialValue>Swimlane </InitialValue></DataField>`

Table 1: Role representation and translation.

AD Construct	XPDL Construct
B2B Action node. ActivityName	Activity with No Implementation. `<Activity Id="ActivityID" Name="ActivityName"><Implementation> <No/></Implementation>...</Activity>`

Table 2: Simple task representation and translation.

Using the toolset, called animator, provided by the Efficient project, users model processes through activity diagrams and data through class diagrams. There are some constrains that have to be followed. A project has obviously to be defined and inside it a package for each activity diagram has also to be defined. The activity diagram is translated into an XPDL *WorkflowProcess* element in [GGM05] a complete example is presented, in particular the translation of the highest level activity diagram of Mercata case-study. The package in which an AD is defined has to be translated into the XPDL package. The business domain described through a class diagram could be part of the XPDL repository. In fact in the repository it is possible to introduce all the relevant data that the workflow processes may need.

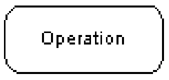
5.1 Roles

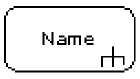
Roles may represent human beings, a piece of software, a machine or something else. A role is the entity that is responsible of the execution of a task. Playing the different roles involved in the process, designers may, exploiting Efficient toolset, validate the process model. In table 1 role concept translation is shown. In particular we see that in UML 2.0 AD roles are represented through swimlanes while in XPDL we propose to use a default-value inside Data Fields.

5.2 Simple and complex activities

Simple activities represent executable activity nodes that constitute fundamental units of work in both XPDL and UML 2.0. In the Efficient project context, simple activities have no implementation and they serve only to show the documents processing, done manually in most cases. In Table 2 the translation rule is represented.

AD Construct	**XPDL Construct**

CallOperationAction. 	Activity implementation type Tool.
	`<Activity Id="ID"Name="Operation"`
	`<Implementation>`
	`<Tool Id="IDTool" Type="APPLICATION">`
	`<ActualParameters>...</ActualParameters>`
	`</Tool></Implementation>...</Activity>`

Activity node (CallBehaviourAction). 	Activity implementation type SubFlow (synch).
	`<Activity Id="ID" Name="Name">`
	`<Implementation>`
	`<SubFlow Execution="SYNCHR"`
	`Id="RelatedWPID">...</SubFlow>`
	`</Implementation></Activity>`

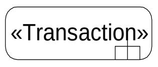
Activity node (CallBehaviourAction) stereotyped *transaction*. 	Activity implementation type SubFlow (synch).The description element could be used in order to indicate the stereotype nature. `<Description>Transaction</Description>` In XPDL 2 the isTransaction element could be used.

Table 3: Complex tasks representation and translation.

Complex activities, represented in XPDL by Tool activities, have also operations associated. In this work we provide a UML representation for them because designers could also be interested in representing Web Services and Web Service invocation. We propose to represent Tool Activities with UML 2.0 CallOperationAction model element. Composed activities are very important because they allow designers to encapsulate specific functionalities together. These activities may accept/return parameters. Moreover by splitting a process into sub-processes we increase the possibility to reuse some of them somewhere. When a subprocess represent a real transaction concept, we propose to explicit it at design time by adding a stereotype *Transaction* in UML and exploiting the description element in XPDL. Workflow engines at the moment are not aware about transactions concepts, however, since XPDL 2.0 will rapidly replace the previous specification, the transaction concept will be supported and our work will find a concrete application. Table 3 summarizes how complex tasks may be represented and translated.

5.3 Pseudo state nodes (or Control nodes) into Route activities

Initial and final pseudo nodes could also be omitted in XPDL since they result deducible. In fact an initial node has usually no ingoing arrows, as well as a final node has no outgoing arrows. But, since in some cases, in which, for example, we are modelling loops, we could have ingoing arrows on the entry point of the initial node, we prefer translating it

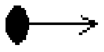
AD Construct	XPDL Construct
-initial. 	Route Activity without transition restrictions or nothing (when optimization applyable). `<Activity Id="initialID" Name="initial"><Route/>...`

Table 4: Initial and final nodes representation and translation.

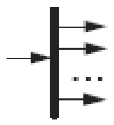
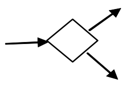
AD Construct	XPDL Construct
- fork. 	Route Activity with transition restriction AND split. `<Activity Id="ID" Name="fork"><Route/> ...<TransitionRestrictions> <TransitionRestriction> <Split Type="AND"/></TransitionRestriction> </TransitionRestrictions></Activity>`
- decision. 	Route Activity with transition restriction XOR split. WfMC defines a decision as an OR split. `<TransitionRestriction> <Split Type="XOR"/></TransitionRestriction>`

Table 5: Fork and decision representation and translation.

into XPDL by using a route activity. Since final node is dealt with similarly, the reader is referred to [GGM05].

Remaining control nodes, in some cases, may also be removed, allowing an optimization in the XPDL specification generation. A *fork* may be removed in all the cases in which it is not preceded by a decision node. The removal is possible because its semantics may be synthesized in the antecedent node. Obviously this synthesis has sense only when the semantics of the two nodes do not come into conflict. Dually a *decision* may be removed in all cases in which its antecedent is not a fork node. In cases in which no optimization rule in order to remove fork/decision nodes is applicable, we propose to represent and translate them as illustrated in Table 5. Concerning *join* and *merge* nodes, the reader is referred to [GGM05].

Deferred choice differs from the “normal“ choice, in that the choice is not made explicitly (based on existing data) but several alternatives are offered to the environment, and the choice between them is delayed until an external signal is received. Using the WFMS terminology, this means that the alternative activities are placed in the worklist, but as soon as one of them starts its execution, the others are withdrawn. This pattern is called implicit XOR-split. In UML 2.0 AD in order to represent this pattern we use the same solution proposed in [WvdAD$^+$04, Whi04].

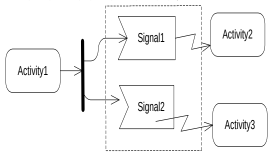
AD Construct	XPDL Construct
Deferred choice. 	not directly supported'a workaround solution has to be used. The solution already in use inside the project context has been kept.

Table 6: Deferred choice representation and translation.

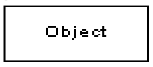
AD Construct	XPDL Construct
Object node. The details of the object and the relationships among all the relevant objects inside the process are presented into a class diagram. 	Data Field (represented by an XMLSchema)+formal and actual parameter definition in case an object constitutes input/output parameter of a sub-flow or of an application. The actual parameter must be the identifier of the corresponding workflow relevant data. `<DataField Id="o_schema" IsArray="FALSE" Name="object"> <DataType>`

Table 7: Documents/Objects representation and translation.

5.4 Documents and Objects

Documents and objects constitute the principal resource that has to be processed. This resource appears as input to activities, it is transformed and reappears as output. Workflow processes orchestrate these documents processing in the best way. In our proposition UML 2.0 Object nodes are translated into XPDL data fields, as Table 7 shows, which structure may be detailed in a class diagram and translated into an XML Schema and finally referenced into the final XPDL generation.

5.5 Transitions or Arcs

Activity diagrams and also the corresponding XPDL Workflow processes are oriented graphs made of nodes and arcs. After having described the nodes representation, we now show in table 8 how to represent transitions.

5.6 Exception Representation and Handling

Exceptions represent undesirable events that happen in an unpredictable instant. In order to face these events, at design time, we should think about a possible behavior to call in those circumstances. Designing exceptional behavior beside the normal one is one way in order to improve reliability. In the Efficient project context, the goal is that of efficiently capturing business transactions. By efficiently capturing, the quality is really important. Reliability is a non functional requirement that belongs to QoS requirements.

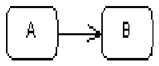
AD Construct	XPDL Construct
Edge. 	Transition. `<Transition From="A-ID" Id=transID" To="B-ID">... </Transition>`

Table 8: Transition representation and translation.

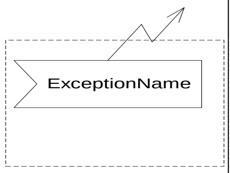
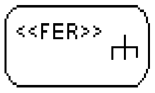
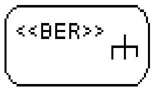
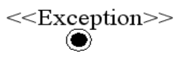
AD Construct	XPDL Construct
Accept Event Action Object Node with an outgoing Interrupting Edge inside an Interruptible region. 	Deadline specification inside each activity belonging to a specific context. `<Deadline Execution="ASYNCHR"> <DeadlineCondition>date </DeadlineCondition> <ExceptionName>ExName </ExceptionName></Deadline>` Transition condition of type Exception `<Transition From="Ax" Id="Trans" Name="XFailure" To="Ay"> <Condition Type="EXCEPTION"> </Condition>...</Transition>`
FER (call behaviour action) 	Sub-Process Definition (synchronous). The description element could be used in order to indicate the stereotype nature. `<Description>FER</Description>` In XPDL 2.0, however, an *Intermediate Event Activity* of type *Target* could be used.
BER (call behaviour action) 	Sub-Process Definition (synchronous). The description element could be used in order to indicate the stereotype nature. `<Description>BER</Description>` In XPDL 2.0, however, an *Intermediate Event Activity* of type *TriggerCancel* could be used.
Exceptional Outcome 	Route Activity with description element Exception. `<Description>Exception</Description>` In XPDL 2.0, *End Event Activity* are available in order to underline the different possible outcomes of a transaction.

Table 9: Exceptions and exceptions handling representation.

As Table 9 summarizes, we propose to use an *InterruptibleRegion* in UML 2.0 around the part of the process that may be subject to exceptions. In XPDL, this concept may be represented by declaring a deadline related to each activity that could be subject to exceptions. Following an *InterruptibleEdge* (a transition condition of type *exception* in XPDL) we land to the exception handling mechanism that is represented by a stereotyped *CallBehaviorAction* called *FER*, in case in which we are able to bring the process to a consistent state different from the initial one, or *BER* when we only are able to roll back the situation.To describe an exceptional termination of a process we propose to stereotype the final node with *Exception*. The process will terminate with an exception in all cases in which the postconditions are not verified. This proposition has been integrated in a research project, called CORRECT [CGGP05], funded by the Luxembourg Ministry of Higher Education and Research under the project number MEN/IST/04/04. In CORRECT the transaction protocol is an advanced one, particularly adapt for facing exceptions in distributed and concurrent complex transactions.

In this work, since the protocol can not be known, because of the recent introduction of real transaction concepts inside workflow definition languages, we only propose a first step towards the inclusion of exceptions description. In XPDL 2.0 not only transaction concept will be supported but also the *Event* concept. In particular Intermediate and End Event Activities seem to be useful in order to catch exceptions and in order to express the different possible outcomes of a transaction. These improvements of expressiveness inside the language make us to keep on thinking that we are moving in the right direction by working on them.

6 Conclusion and future work

In this work we have presented a way to describe nested business processes and also exception and exception handling mechanisms in UML 2.0 activity diagrams and the corresponding transformation rules in order to generate the XPDL specification from them. In fact in order to automatically generate an XPDL specification from activity diagrams, currently, we have improved the plugin developed in the project context. This plugin has been implemented inside the commercial tool MagicDraw [NM05] and is part of the toolset Animator. Since however, at the time of writing, the stable release of this tool does not support UML 2.0 metamodel. We have used UML 1.5 metamodel (Subactivities instead of CallBehaviorAction, for example) and we have generated automatically the specification of nested processes without covering the exception handling part because of the lack of tools. Exploiting MagicDraw open API, the algorithm has been implemented in the Java language.

In the future, we intend to update to the tenth MagicDraw version in order to exploit the UML 2.0 metamodel support and to be able to complete implement the proposed transformation rules. We also intend to take into deep consideration XPDL 2.0 and in particular the support for ACID properties and exception handling. Using IsATransaction specification inside an Activity definition, a subflow, for example, can be set as being a Transaction, which will have a special behavior that is controlled through a transaction protocol.

Moreover while the translation of Backward and Forward Error Recovery into XPDL 1.0 necessitates an extension of the available workflow engines; in XPDL 2.0, these concepts are part of the language and all workflow engine compatible with the new specification will understand them. Moreover the formalisation research work started in [GM05] will be kept on by formalizing nested processes and exception handling mechanism. Our goal in fact is that of verifying not only structural properties of workflow models described in activity diagrams but also non functional ones in order to ensure step by step more trusted systems.

References

[CGGP05] A. Capozucca, B. Gallina, N. Guelfi, and P. Pelliccione. Modeling Exception Handling: a UML2.0 Platform Independent Profile for CAA. In *Proceedings of ECOOP 2005 Workshop on Exception Handling in Object Oriented Systems, Glasgow (Scotland), Department of Computer Science. LIRMM; Montpellier-II University,*, http://se2c.uni.lu/tiki/se2c-bib_download.php?id=1928, 2005.

[DG01] E. Dubois and N. Guelfi. Demande de Contribution Financière pour la Réalisation d'un Projet de Recherche: EFFICIENT, Fonds National de la Recherche, Luxembourg-Kirchberg, Luxembourg., 2001.

[EBD$^+$03] R. Eshuis, P. Brimont, E. Dubois, B. Grégoire, and S. Ramel. EFFICIENT: a Tool Set for Supporting the Modelling and Validation of ebXML. In *Proceedings of the 9th European software engineering conference held jointly with 11th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 359–362, 2003.

[Esh03] R. Eshuis. Activity Diagrams as XPDL Specification, Draft version May 16,2003.

[GCR04] N. Guelfi, G. Le Cousin, and B. Ries. Engineering of Dependable Complex Business Processes Using UML and Coordinated Atomic Actions. In Springer, editor, *International Workshop on Modeling Inter-Organizational Systems (MIOS'04), Larnaca, Cyprus*, pages 468–482, 2004.

[GGM05] B. Gallina, N. Guelfi, and A. Mammar. Structuring Business Nested Processes Using UML 2.0 Activity Diagrams and Translating into XPDL, Technical Report TR-SE2C-05-07 University of Luxembourg 2005.

[GM05] N. Guelfi and A. Mammar. A Formal Framework to Generate an XPDL Specification from a UML Activity Diagram, Technical Report available on http://se2c.uni.lu/users/AM University of Luxembourg 2005.

[JMN03] P. Jiang, Q. Mair, and J. Newman. Using uml to design distributed collaborative workflows: from uml to xpdl. In *12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, Linz, Austria, IEEE Press*, pages 71–76, 2003.

[MG05] T. Mens and P. Van Gorp. A Taxonomy of Model Transformation (s). In *International Workshop on Graph and Model Transformation (GraMoT) Tallinn, Estonia September 28,*, 2005.

[NM05] Inc No Magic. MagicDraw 9.0 Version, Commercial tool http://www.magicdraw.com/ , 2005.

[Obj03] Object Management Group (OMG). Unified Modeling Language (UML): Superstructure version 2.0, final adopted specification (02/08/2003). http://www.omg.org/cgi-bin/doc?ptc/2003-08-02, 2003.

[RtHEvdA04] N. Russell, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow Data Patterns., QUT Technical report, FIT-TR-2004-01, Queensland University of Technology, Brisbane, 2004.

[SD05] B. A. Schmit and S. Dustdar. Model-driven development of web service transactions. In M. Nüttgens and J. Mendling, editors, *XML4BPM 2005, Proceedings of the 2nd GI Workshop XML4BPM – XML Interchange Formats for Business Process Management at 11th GI Conference BTW 2005, Karlsruhe Germany, March 2005*, pages 39–54, http://wi.wu-wien.ac.at/˜mendling/XML4BPM2005/xml4bpm-2005-proceedings-schmit.pdf, March 2005.

[Tud05] Tudor Research center. Animator. http://efficient.citi.tudor.lu/releases/ , 2005.

[vdA03] W.M.P. van der Aalst. Patterns and XPDL: A Critical Evaluation of the XML Process Definition Language., QUT Technical report, FIT-TR-2003-06, Queensland University of Technology, Brisbane, 2003.

[vdA04] W.M.P. van der Aalst. Business process management demystified: A tutorial on models, systems and standards for workflow management. In Springer, editor, *In J. Desel, W. Reisig, and G. Rozenberg, editors, Lectures on Concurrency and Petri Nets, volume 3098 of Lecture Notes in Computer Science, Berlin*, pages 1–65, 2004.

[vdAtHKB00] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In Springer, editor, *In O. Etzion and P. Scheuermann, editors, 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of Lecture Notes in Computer Science, Berlin,*, 2000.

[WfM05] WfMC. http://www.wfmc.org/, 2005.

[Whi04] IBM Corp. United States S. A. White. Process Modeling Notations and Workflow Patterns. In *The Workflow Handbook 2004. WfMC, Future Strategies Inc., Lighthouse Point , FL , USA*, 2004.

[WvdAD$^+$04] P. Wohed, W.M.P. van der Aalst, M. Dumas, A.H.M. ter Hofstede, and N. Russell. Pattern-based Analysis of UML Activity Diagrams, BETA Working Paper Series, WP 129, Eindhoven University of Technology, Eindhoven, 2004.

[XPD02] XPDL1.0. http://www.wfmc.org/standards/docs/tc-1025_10_xpdl_102502.pdf, 2002.

[XPD05] XPDL2.0. http://www.wfmc.org/standards/docs/tc-1025_xpdl_2_2005-10-03.pdf, 2005.

Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages

Jan Mendling[1], Kristian Bisgaard Lassen[2], Uwe Zdun[1]

[1] Institute of Information Systems and New Media
Vienna University of Economics and Business Administration
Augasse 2-6, A-1090 Wien, Austria
{jan.mendling|uwe.zdun}@wu-wien.ac.at
[2] Department of Computer Science, University of Aarhus
IT-parken, Aabogade 34, DK-8200 Aarhus N, Denmark
k.b.lassen@daimi.au.dk

Abstract: Much recent research work discusses the transformation between different process modelling languages. This work, however, is mainly focussed on specific process modelling languages, and thus the general reusability of the applied transformation concepts is rather limited. In this paper, we aim to abstract from concrete transformation strategies by distinguishing two major paradigms for representing control flow in process modelling languages: block-oriented languages (such as BPEL and BPML) and graph-oriented languages (such as EPCs and YAWL). The contribution of this paper are generic strategies for transforming from block-oriented process languages to graph-oriented languages, and vice versa.

1 Introduction

Business process modelling (BPM) languages play an important role not only for the specification of workflows but also for the documentation of business requirements. Even after more than ten years of standardization efforts [Hol04], the primary BPM languages are still heterogeneous in syntax and semantics. This problem mainly relates to two issues: Firstly, various BPM language concepts that need to be specified in terms of control flow [vdAtHKB03] and data flow [RtHEvdA05] have been identified, and most BPM languages introduce a different sub-set of these (see [MNN04] for a comparison of BPM concepts). Secondly, the paradigm for representing control flow used in the BPM languages is another source of heterogeneity. This issue has not been discussed in full depth so far, but it is of special importance when transformations between BPM languages need to be implemented. In essence, two control flow paradigms can be distinguished, graph- and block-oriented:

- *Graph-oriented* BPM languages specify control flow via arcs that represent the temporal and logical dependencies between nodes. A graph-oriented language may include different types of nodes. These node types may be different from language to language. Workflow nets [vdA97] distinguish places and transitions similar to Petri nets. EPCs [KNS92, MN05] include function, event, and connector node types. YAWL [vdAtH05] uses graph nodes that represent tasks and conditions. Similar to XPDL [Wor02], these tasks may specify join and split rules.

- *Block-oriented* BPM languages define control flow by nesting control primitives used to represent concurrency, alternatives, and loops. XLANG [Tha01] is an example of a pure block-oriented language. BPML [Ark02] and BPEL [ACD$^+$03] are also block-oriented languages but they also include some graph-oriented concepts (i.e. links). In BPEL, the control primitives are called structured activities. Due to the widespread adoption of BPEL as a standard, we will stick to BPEL as an example of a block-oriented language. Please note that the concepts presented later are also applicable for other block-oriented languages, but as our definitions of block-oriented control flow are rather BPEL-specific, some effort is needed to customize our concepts to other block-oriented languages.

Transformations between block-oriented languages and graph-oriented languages are useful or needed in a number of scenarios. Many commercial tools support the import and export in other formats and languages, meaning that transformations in both directions are implemented by import and export filters. For instance, many graph-oriented tools are recently enhanced to export BPEL in order to support the standard for interoperability and commercial reasons. Transforming BPEL to Petri nets is done for the purpose of verification [HSS05]. BPEL does not have formal semantics and can therefore not be verified. By defining a transformation semantics for BPEL in terms of a mapping to Petri nets, it is possible to investigate behavioral properties, such as dead-locks and live-locks. BPEL process definitions are also transformed to EPCs with the goal to communicate the process behavior e.g. to business analysts in a more visual representation [MZ05]. In the other direction, model-driven development approaches start from a visual graph-oriented BPM language such as UML activity diagrams to generate executable BPEL models [Gar03]. These are only some example scenarios, where BPM transformations are needed. The contribution of this paper is to abstract from particular graph-oriented or block-oriented control flow representation, to enable a generic discussion of transformation strategies between both. The presented transformation strategies are independent from a certain application scenario and can therefore be used in any setting where transformations between graph-oriented and block-oriented languages are needed.

The rest of the paper is structured as follows. Section 2 defines the abstractions that are used throughout this paper. In particular, we define an abstraction of graph-oriented BPM languages called *Process Graph* that shares most of its concepts with EPCs and YAWL. Block-oriented languages are abstracted by a language called *BPEL Control Flow*. This language is – as mentioned before – an abstraction of BPEL concepts, but can be mapped to the concepts of other block-oriented languages such as BPML. In Section 3 we discuss strategies for transforming BPEL Control Flow to Process Graph, and in Section 4 the opposite direction. The strategies are specified using pseudo-code algorithms and their prerequisites, advantages, and shortcomings are discussed. Section 5 discusses related work, and finally Section 6 concludes and discusses future work.

2 Process Graphs and BPEL Control Flow

2.1 Introductory Example

To discuss transformations between graph-oriented and block-oriented BPM languages in a general way, we have to abstract from specific languages. Before that, we illustrate some features of process graphs and BPEL control flow.

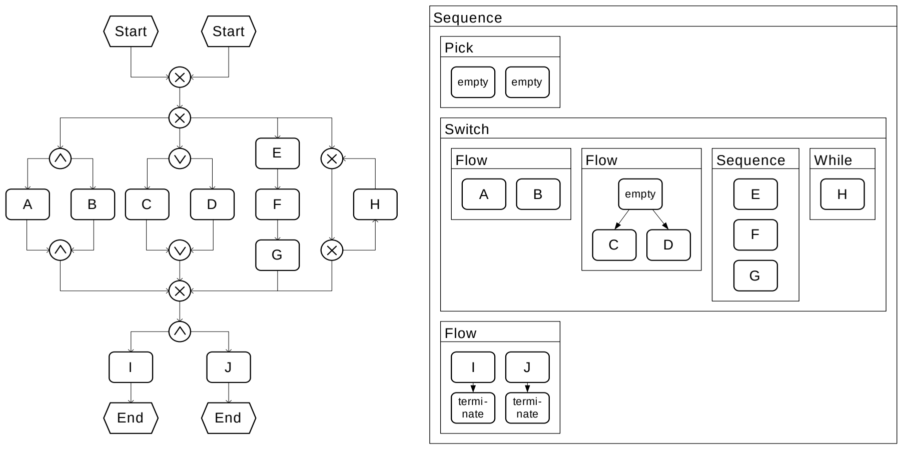


Figure 1: Process graph and BPEL control flow

The left part of Figure 1 shows a *process graph*. As we are interested in syntax transformations, we give the semantics of process graphs only in an informal manner. A process graph has at least one start event and can have multiple end events. Multiple start events represent mutually exclusive, alternative start conditions. End events have explicit termination semantics. This means that when an end event is reached, the complete process is terminated. Connectors represent split and join rules of type OR, XOR, or AND, as they are specified for YAWL [vdAtH05] or EPCs [MN05]. All of these elements are connected via arcs which may have an optional guard. Guards are logical expressions that can evaluate to true or false. If a guard of an arc from a connector node with type OR or XOR yields false, the target branch of the arc is not executed. If true, execution continues with the target function. After an XOR split, the logical expressions of guards of the subsequent arcs must be mutually exclusive.

The right part of Figure 1 gives a *BPEL control flow* with similar control flow semantics as the process graph. In the example, so-called structured activities are used whenever possible. There are structured activities to define alternative start conditions (pick), parallel execution (flow), sequential execution (sequence), conditional repetition (while), and alternative branches (switch). Structured activities can be nested for the definition of complex control flow behavior. Basic activities represent atomic elements of work. There are special basic activities to represent that nothing is done (empty) or that the BPEL control flow is terminated (terminate). Within a flow activity, complex synchronization conditions can be specified via so-called links. Each link can have a transition condition, and each activity that is a target of links can include a join condition of the type OR, XOR, or AND. For BPEL control flow, we adopt the semantics defined in the BPEL specification [ACD$^+$03].

2.2 Definition of Process Graphs

To provide for a precise description of the transformation strategies, we formalize the syntax of process graphs and those aspects of BPEL that are relevant for a transformation of control flow. We define process graphs to be close to EPCs and YAWL using an EPC-like notation. The respective syntax elements provide the core of graph-based business process modelling languages. Furthermore, AND and XOR connectors can easily be mapped to Petri nets, XPDL, or UML activity diagrams.

Notation 1 (Predecessor and Successor Nodes) Let N be a set of $nodes$ and $A \subseteq N \times N$ a binary relation over N defining the arcs. For each $node\ n \in N$, we define the set of $predecessor\ nodes\ \bullet n = \{x \in N | (x, n) \in A\}$, and the set of $successor\ nodes\ n\bullet = \{x \in N | (n, x) \in A\}$.

Definition 1 (Process Graph PG) A process graph $PG = (S, E, F, C, l, A, g)$ consists of four pairwise disjoint sets S, E, F, C, a mapping $l : C \rightarrow \{AND, OR, XOR\}$, a binary relation $A \subseteq (S \cup F \cup C) \times (E \cup F \cup C)$, and a mapping $guard : A \rightarrow expr$ such that:

– S denotes the set of start events. $|S| \geq 1$ and $\forall s \in S : |s\bullet| = 1 \wedge |\bullet s| = 0$.
– E denotes the set of end events. $|E| \geq 1$ and $\forall e \in E : |\bullet e| = 1 \wedge |e\bullet| = 0$.
– F denotes the set of functions. $\forall f \in F : |\bullet f| = 1 \wedge |f\bullet| = 1$.
– C denotes the set of connectors. $\forall c \in C : |\bullet c| = 1 \wedge |c\bullet| > 1 \vee |\bullet c| > 1 \wedge |c\bullet| = 1$
– The mapping l specifies the type of a connector $c \in C$ as AND, OR, or XOR.
– A defines the flow as a simple and directed graph. An element of A is called arc. Being a simple graph implies that $\forall n \in (E \cup F \cup C) : (n, n) \notin A$ (no reflexive arcs) and that $\forall x, y \in (E \cup F \cup C)\} : |\{(x, y) | (x, y) \in A\}| = 1$ (no multiple arcs).
– The mapping $guard$ specifies a guard for an arc $a \in A$. $expr$ is a non-terminal symbol to represent a logical expression that defines the guard condition. If and only if this expression yields true, control is propagated to the node subsequent to the guard. Guards of arcs after XOR connector nodes have to be mutually exclusive. Guards are defined on A, however it is only arcs (c,n), where $c \in C$, $l(c) \neq AND$ and $n \in E \cup F \cup C$, that can be expressed as any logical expression. All other guard always yields true; e.g. a guard from an AND-split can never yield false and each function in a sequence is always executed.

Definition 2 (Transitive Closure) Let $PG = (S, E, F, C, l, A, g)$ be defined as in Definition 1. Then $A*$ is the transitive closure of A. That is, if $(n_1, n_2) \in A*$ there is a path from n_1 to n_2 in the process graph via some arcs of A.

2.3 Definition of BPEL Control Flow

Definition 3 (BPEL Control Flow) A BPEL Control Flow BCF is a tuple $BCF = (Seq, Flow, Switch, While, Pick, Scope, Basic, Empty, Terminate, Link, de, jc, tc)$. BCF consists of pairwise disjoint sets $Seq, Flow, Switch, While, Pick, Scope, Basic,$ $Empty, Terminate$. The set $Str = Seq \cup Flow \cup Switch \cup While \cup Pick \cup Scope$ is called structured activities, the set $Bas = Basic \cup Empty \cup Terminate$ is called basic activities, and the set $Act = Str \cup Bas$ activities. Furthermore, BCF consists of a binary relation $Link \subseteq Act \times Act$, a mapping $de : S \rightarrow \mathbb{P}(A) \setminus \emptyset$, a mapping $jc : A \rightarrow expr$, and a mapping $tc : Link \rightarrow expr$, such that

– Seq defines the set of BPEL sequence activities.
– $Flow$ defines the set of BPEL flow activities.
– $Switch$ defines the set of BPEL switch activities.
– $While$ defines the set of BPEL while activities.
– $Pick$ defines the set of BPEL pick activities.
– $Scope$ defines the set of BPEL scopes.
– $Basic$ defines the set of BPEL basic activities without terminate and empty activities. As we are only interested in control flow, the distinction of various basic activities can be neglected here.
– $Empty$ defines the set of BPEL empty activities.
– $Terminate$ defines the set of BPEL terminate activities.
– $Link$ defines a directed graph of BPEL links. These need not to be coherent, but acyclic, and not be connected across the borders of a while activity.
– The mapping de denotes a decomposition relation from structured activities to set of nested activities modelled as the power set $\mathbb{P}(A)$. de is a tree, i.e. there is no recursive decomposition.
– The mapping jc defines the join condition of activities.
– The mapping tc defines the transition condition of links.

Definition 4 (Join condition) The join condition, jc, on activities is defined as a $jc : A \rightarrow expr$ using operations such as $\wedge$, $\vee$ and $\veebar$. For an activity x, where $\bullet x = \{y_1, \ldots, y_n\}$ including its predecessor in a structured activity, we use the shorthand AND, OR and XOR for the boolean expressions

$$jc(x) = tc(y_1, x) \wedge \ldots \wedge tc(y_n, x) \text{ (AND)}$$
$$jc(x) = tc(y_1, x) \vee \ldots \vee tc(y_n, x) \text{ (OR)}$$
$$jc(x) = tc(y_1, x) \veebar \ldots \veebar tc(y_n, x) \text{ (XOR)}$$

Definition 5 (Subtree Fragment) Let $Struct \subseteq Act \times Act$ be relation with $(a_1, a_2) \in Struct$ if and only if $a_2 \in de(a_1)$. $Struct*$ is the transitive closure of $Struct$. This implies that a_2 is nested in the subtree fragment of a_1.

Notice that Definition 2.3 do not describe event-, fault-, and compensation handlers. This is because our strategies do not take these into consideration. Also, we do not allow links to cross scope boundaries.

For the purpose of discussing control flow transformations, other BPEL elements than those included in the definition can be neglected. For details on BPEL semantics refer to [ACD$^+$03]. Note that e.g. BPML has similar syntax elements with comparable semantics [Ark02]. Accordingly, the strategies discussed in the following section can also be applied to define transformations between BPML and process graphs.

2.4 Structural Properties of Process Graphs and BPEL Control Flow

Various transformation choices are bound to certain structural properties of the input model. A process graph can be structured or unstructured and acyclic or cyclic. We define a process graph to be structured by the help of reduction rules. They provide not only a formalization of structuredness but also a means to define a transformation strategy from process graphs to BPEL control flow. Details on this will be explained in Section 4.

Definition 6 (Structured Process Graph) A process graph PG is structured if and only if it can be reduced to a single node by the reduction rules formally defined in [MLZ05], otherwise it is unstructured. All the reduction rules describe a certain component that is part of the process graph and then how to replace it by a single function. There are reduction rules for sequences, connector pairs, XOR-loops, and start- and end-blocks. For details refer to [MLZ05].

Definition 7 (Cyclic versus Acyclic Process Graph) Let $F \cup C$ be the set of functions and connectors of a process graph PG. If $\exists n \in F \cup C : (n,n) \in A*$, then PG is cyclic. If $\forall n \in F \cup C : (n,n) \notin A*$, then PG is acyclic. As a process graph is a simple graph, it holds that $(n,n) \notin A$ (no reflexive arcs). But if $(n,n) \in A*$, there must be a path from n to n via some further nodes $n_1, ..., n_m \in (E \cup F \cup C)$.

Definition 8 (Structured BPEL Control Flow) A BPEL Control Flow BCF is structured if and only if its set $Link = \emptyset$. Otherwise BCF is unstructured.

Furthermore, we define the point wise application of mapping functions which we need in algorithms for the transformation strategies.

Definition 9 (Point Wise Application of Functions) If a function is defined as $f : A \rightarrow B$ then we extend the behavior to sets so that $f(X) = \cup_{x \in X} f(x), X \subseteq A$.

3 BPEL Control Flow to Process Graph Transformation Strategies

3.1 Strategy 1: Flattening

Before we present the transformation algorithms, we need to define the mapping function M that transforms a BPEL basic activity to a process graph function.

Definition 10 (Mapping Function M) Let F be a set of functions of a process graph PG and $Basic$ a set of basic activities of a BCF. The mapping $M : Basic \rightarrow F$ defines a transformation of a BPEL basic activity to a process graph function.

Algorithm 1 Pseudo code for Flattening strategy

```
procedure: Flattening($BCF$)
1: $Struct \leftarrow Seq \cup Flow \cup Switch \cup While \cup Pick \cup Scope$
2: $S \leftarrow \{s\}$; $E \leftarrow \{e\}$; $F \leftarrow \emptyset$; $C \leftarrow \emptyset$; $A \leftarrow \emptyset$
3: $root \leftarrow a$, where $a \in Struct \wedge \nexists s \in Struct : de(s) = a$
4: BCFtransform($root, s, e, PG$)
5: for all $(l_1, l_2) \in Link$ do
6:     $A \leftarrow A \cup \{(c_1, c_2)\}$
7:     $guard(c_1, c_2) = tc(l_1, l_2)$
8: end for
9: return $PG$
```

The *general idea* of the Flattening strategy is to map BCF structured activities to respective process graph fragments. The nested BCF control flow then becomes a flat process graph without hierarchy. For this strategy, there are *no prerequisites*, both structured and unstructured BPEL control flow can be transformed according to this strategy. The *advantage* of Flattening is that the behavior of the whole BPEL process is mapped to one process graph. Yet, as a *drawback* the descriptive semantics of structured activities get lost. Such a transformation strategy is useful in a *scenario* where a BPEL process has to be communicated to business analysts.

The *algorithm* for the Flattening strategy takes a BCF as input and returns a PG. It recursively traverses the nested structure of BPEL control flow in a top-down manner. This is achieved by identifying the root activity and invoking the *BCFtransform(activity, predecessor, successor, partialResult)* procedure (see Algorithm 1, line 4) which is reinvoked recursively on nested elements. The respective code is given in Algorithm 2 in [MLZ05]. The first parameter $activity$ represents the activity to be processed followed by the predecessor and successor node of the output process graph between which the nested structure is hooked in; i.e. $predecessor$ and $successor$. For the root activity these are the start and end events s and e. The parameter $partialResult$ is used to forward the partial result of the transformation to the procedure. In lines 5–8 links are mapped to arcs and respective join and split connectors around the activity are added.

The $BCFtransform$ procedure (see Algorithm 2 in [MLZ05]) starts with checking whether the current activity serves as target or source for links. If so, respective connectors are added at the beginning and the end of the current activity block. There are four sub-procedures to handle the five structured activities Seq, $Flow$, $Switch$, $While$, and $Pick$. Here, it is assumed that $Pick$ is only used to model alternative start events.[1] The transformation of $Scopes$ simply calls the procedure for its nested activity.[2] $Terminate$ is mapped to an end event. Moreover, $Basic$ activities are mapped to functions using M and hooked in the process graph. $Empty$ activities map to an arc between predecessor and

[1]In BPEL, $Pick$ can be used at any place where the process waits for concurrent events. As we do not distinguish message-based and other basic activities, decisions are captured by a $Switch$ in BCF.

[2]Please note that $Scopes$ play an important role in BPEL as a local context for variables, handlers, and also $Terminate$ activities. In the algorithm we abstract from the fact that $Terminate$ only terminates the current $Scope$ but not the whole process. Furthermore, we abstract from the fact that a BPEL terminate leads to improper termination.

successor nodes.
The procedures $BCFtransformSeq$, $BCFtransformBlock$, $BCFtransformPick$, and $BCFtransformWhile$ used in the $BCFtransform$ procedure generate the process graph elements that correspond to the respective BCF structured activities. The $BCFtransformSeq$ procedure connects all nested activities of a sequence with process graph arcs. Although not explicitly defined, this transformation requires an order defined on the nested activities. For each sub-activities the $BCFtransform$ procedure is invoked again. This is similar to $BCFtransformBlock$. Here, a split and a join connector are generated. Depending on the label given as a fourth parameter the procedure can transform both $Switch$ or $Flow$. The $BCFtransformPick$ replaces the start event of the process graph with one start event for each nested sub-activity. Finally, the $BCFtransformWhile$ procedure generates a loop between an XOR join and XOR split.

3.2 Strategy 2: Hierarchy-Preservation

Many graph-based BPM languages allow to define hierarchies of processes. EPCs for example include hierarchical functions and process interfaces to model sub-processes. In YAWL tasks can be decomposed to sub-workflows. Process graphs can be extended to process graph schemas in a similar way to allow for decomposition.

Definition 11 (Process Graph Schema PGS) A process graph schema $PGS = \{PG, s\}$ consists of a set of process graphs PG and a mapping $s : F \rightarrow \{\emptyset, pg\}$ with $pg \in PG$. The mapping s is called subprocess relation. It points from a function to a refining subprocess or, if the function is not decomposed, to the empty set. The relation s is a tree, i.e. there is no recursive definition of sub-processes.

The *general idea* of the Hierarchy-Preservation strategy is to map each BCF structured activity to a process graph of a process graph schema. The nesting of structured activities is preserved as functions with subprocess relations. The algorithm can be defined in a top-down way similar to the Flattening strategy. Changes have to be defined for the transformation of structured activities as each is mapped to a new process graph. A *prerequisite* of this strategy is that the BCF is structured: links across the border of structured activities cannot the expressed by the subprocess relation. The *advantage* of the Hierarchy-Preservation strategy is that the descriptive semantics of structured activities can be preserved. Furthermore, such a transformation can correctly map the BPEL semantics of $Terminate$ activities that are nested in $Scopes$. As a *drawback*, the model hierarchy has to be navigated in order to understand the whole process. This strategy might be useful in a *scenario* where process graphs have to be mapped back to BPEL structured activities.

3.3 Strategy 3: Hierarchy-Maximization

One disadvantage of Strategy 2 is that it is bound to structured BPEL. The Hierarchy-Maximization Strategy aims at preserving as much hierarchy as possible with also being applicable to any BPEL control flow – anyway if structured or unstructured. The *general* *idea* of the strategy is to map those BCF structured activities s to subprocess hierarchies if there are no links nested that cross the border of s. Accordingly, this strategy is not subject to any structural *prerequisites*. The advantage is that as much structure as possible is preserved. Yet, the logic of both Strategy 1 and Strategy 2 need to be implemented.

4 Process Graph to BPEL Control Flow Transformation Strategies

4.1 Strategy 1: Element-Preservation

In this section we will describe the first strategy for going from process graphs to BCF. The following Definitions 12 (Annotated Process Graph) and 13 (Annotated Process Graph Node Map) are also relevant of further strategies. Before we go through the strategies we will make some definitions where we introduce the notion of an annotated process graph to ease the notation in strategy.

Definition 12 (Annotated Process Graph) Let $APG = (S, E, F, C, l, A, B)$ define an annotated graph, where S, E, F, C and l are defined as Definition 1. We define A and B as

– A is a flow relation on the nodes in PG, $A = (S \cup F \cup C \cup B) \times (E \cup F \cup C \cup B)$.

– B is a node in PG that holds an annotation in BCF.

One could think of the set B in the annotated graph, Definition 12, as the set of already translated parts of the process graph. Definition 13 shows how to translate the nodes in an annotated process graph. The *general idea* of this strategy is to map all process graph elements to a $Flow$ and map arcs to $Links$. In particular, start events are mapped to $Basic$,[3] function are mapped to elements of $Basic$, and connectors are mapped to elements of $Empty$, and end events are translated to elements of $Terminate$. M defines the identity on BPEL constructs.

Definition 13 (Annotated Process Graph Node Map) Let M define a mapping: $E \cup S \cup F \cup C \cup B \rightarrow Basic \cup Empty \cup Terminate \cup B$ and M is defined as

$$M(x) = \begin{cases} Empty(x), & \text{if } x \in C; \\ Basic(x), & \text{if } x \in F \cup S; \\ Terminate(x), & \text{if } x \in E; \\ x, & \text{if } x \in B. \end{cases}$$

an injective translation from the nodes in the graph to activities in BPEL.

It is a *prerequisite* of this strategy that the process graph needs to be acyclic, i.e. $(x, x) \notin A*$. This is because it is not possible to create an activity that logically precedes itself [ACD$^+$03]. I.e., if X precedes Y then Y cannot precede X. The *advantage* of the Element-Preservation strategy is that it is simple to implement and the resulting BPEL will be very similar to the original process graph since there is a one-to-one correspondence between

[3]As a consequence, all alternative start branches are activated when the process is started. Specific transition conditions could be defined to have only one branch being activated. In the algorithm we abstract from this issue.

the nodes. As a *drawback*, the resulting BPEL control flow includes more elements than actually needed: connectors are explicitly translated to empty activities in BPEL instead of join condition on nodes. This means that the BPEL code might have a lot of nodes which simply act as synchronization points. Furthermore, the resulting BPEL might be more difficult to understand than if structured activities, such as the Switch, where chosen to represent some part of the translated graph. If the BPEL code is used in a *scenario* where readability is important, then it should be applied only for small process graphs since all elements of the process graph are mapped to BCF.

Algorithm 2 Pseudo Code for Element-Preservation strategy

```
procedure: Element-Preservation($PG$)
1:  $Empty \leftarrow M(C)$
2:  $Basic \leftarrow M(F \cup S)$
3:  $Terminate \leftarrow M(E)$
4:  $Flow \leftarrow flow$
5:  $de(flow) \leftarrow Empty \cup Basic \cup Terminate$
6:  $Link \leftarrow \emptyset$
7:  for all $(x, y) \in A$ do
8:      $Link \leftarrow Link \cup (M(x), M(y))$
9:  end for
10: $jc(x) = \begin{cases} AND, & |\bullet M^{-1}(x)| > 1 \wedge l(M^{-1}(x)) = and; \\ XOR, & |\bullet M^{-1}(x)| > 1 \wedge l(M^{-1}(x)) = xor; \\ OR, & otherwise. \end{cases}$
11: $tc(x, y) = guard(M^{-1}(x), M^{-1}(y))$
12: return($BCF$)
```

The *algorithm* for the Element-Preservation strategy takes a process graph as input and generates a respective BCF as output. The Algorithm 2 applies the map M as defined in Definition 13 in lines 1–3. Then, a flow element is added that nests all other activities (lines 4–5). For each arc in the process graph between two nodes a link is added in the BCF between the corresponding two BCF nodes (lines 6–9). The join condition of activities is determined from their corresponding node in the process graph. If it is a connector it will get a similar join condition, i.e. AND for and, OR for or and XOR for xor. Other nodes will get an OR join condition (line 10). If two nodes are connected by a guarded arc then this guard will also be present in the BPEL (line 11).

4.2 Strategy 2: Element-Minimization

This strategy simplifies the generated BCF of strategy 1. The *general idea* is to remove the empty activities that have been generated from connectors and instead represent splitting behavior by transition conditions of links and joining behavior by join conditions of subsequent activities. As a *prerequisite* the process graph needs to be acyclic, i.e. $(x, x) \notin A*$, in order to make dead path elimination of BPEL work. The *advantage* of the resulting BCF specification is, at least to a greater extent than strategy 1, that it is in the spirit of BPEL Flow, since it removes empty activities generated from connectors. As a *drawback*, it is less intuitive to identify correspondences between the process graph and the generated BCF specification. This strategy should be used in *scenarios* where the resulting BPEL code needs to have as few nodes as possible. This might be the case when performance of the BPEL process matters. In contrast to strategy 1, the amount of nodes is decreased since all empty activities translated from connector nodes are skipped .

Algorithm 3 Pseudo code for Element-Minimization strategy

```
procedure: Element-Minimization($PG$)
 1: $BCF \leftarrow$ Element-Preservation($PG$)
 2: while $\exists x \in Empty : M^{-1}(\bullet x) \cap C = \emptyset$ do
 3:     $Link \leftarrow Link \cup \{(y_1, y_2) \mid y_1 \in \bullet x \wedge y_2 \in x\bullet\}$
 4:     for all $y \in x\bullet$ do
 5:         $jc \leftarrow \left(jc'(y') = \begin{cases} jc(y'), & y' \neq y; \\ jc(y') \wedge jc(x), & otherwise. \end{cases}\right)$
 6:     end for
 7:     $Link \leftarrow Link \setminus (\{(x, y) \mid y \in x\bullet\} \cup \{(x, y) \mid y \in \bullet x\})$
 8:     $Empty \leftarrow Empty \setminus \{x\}$
 9: end while
10: return($BCF$)
```

The *algorithm* translates a PG into a BCF using Algorithm 2 (line 1). Then, there is a loop iterating over all empty activities that have been generated from connectors (line 2) and do not have other translated connector nodes as input links. Finally all translated connector nodes will be removed. For each empty activity x, the nodes having a link to it, are connected to nodes having a link from it. Then, the join conditions of the activities subsequent to x need to be updated. The join condition of an activity is the old join condition it had, before removing x, in conjunction with the join condition of x (lines 4–6). Lines 7–9 defines the actual removal of x. This involves removing all link relations that x occurs in and removing x from the set of Empty activities.

4.3 Strategy 3: Structure-Identification

The *general idea* of this transformation strategy is to identify structured activities in the process graph and apply mappings that are similar to the reduction rules given in Definition 6 on them. As a *prerequisite* the process graph needs to be structured according to Definition 6. The *advantage* of this strategy is that all control flow is translated to structured activities. For understanding the resulting code this is the best strategy since it reveals the structured components of the process graph. As a *drawback* the relation to the original process graph might not be intuitive to identify. This transformation strategy is appropriate in a *scenario* when the BCF is to be edited by a BPEL modeling tool or, generally, when understanding the control flow of the process graph is important.
Our *algorithm* uses the reduction rules of Definition 6, but instead of substituting a pattern with a function it is replaced by an annotated node containing the BPEL translation of

the process graph fragment. This means, in reducing the process graph we generate an annotated process graph that finally includes only one single annotated node. A single function is mapped to $Basic$ in the resulting BCF, whereas annotated nodes are mapped to the set which their annotation is a member of; e.g. $Switch$ if a Switch annotation. Each of the rules identifies structure that has an equivalent representation in BPEL as follows:

– A sequence of elements is translated to a BCF sequence with activities in the same order as nodes of the process graph sequence.
– An AND-block is translated to a flow in the BCF. The nodes of the AND-block are translated to nested activities of the flow.
– An OR-block is translated to a flow in the BCF. The nodes of the OR-block are translated to nested activities of the flow with an additional empty activity. This points to each alternative branch and transition conditions are used to activate only a subset of branches. Notice that this translation makes the BCF unstructured.
– An XOR-block is translated to a switch in the BCF. Each branch of the XOR-block is mapped to a nested activity of the switch including the respective guard.
– A mixed loop has no direct representation in the BCF. As the rule in Definition 6 state the graph has the structure $c_1\bullet = \{a_1\}$, $\bullet c_1 \cap c_2\bullet = \{a_2, \ldots, a_n\}$. The condition to leave the loop is `cond`, i.e. the boolean expression $(\veebar_{x\in A} guard(x)) \wedge \neg(\veebar_{x\in B} guard(x)), A = \{(c_2, x)|x \in \bullet c_1 \cap c_2\bullet\}$ and $B = \{(c_2, x)|x \notin \bullet c_1 \cap c_2\bullet\}$. However, since exactly one of the arcs from an XOR connector node is true at a time the boolean expression can be reduced to both the left and the right part in the conjunction. Guards in PG are mapped to transition conditions in the BFC. The mixed loop can be mapped to the following BPEL pseudo code:

```
1: assign(continueLoop,true);
2: while(continueLoop) {
3:   M(a1);
4:   switch {
5:     case cond: assign(continueLoop,false);
6:     case tc(c2,a2)): M(a2);
7:     ...
8:     case tc(c2,an)): M(an);
9:   }
10:}
```

– A while-do loop is translated into a while activity with a switch inside it. It is mapped as the mixed loop with the difference that lines 1, 3, and 5 are omitted and the condition, `cond`, for looping replaces the `continueLoop` in line 2.
– A repeat-until loop has no direct representation in the BCF. It is mapped in a similar way as the mixed loop – lines 6 through 8 in the pseudo code are omitted.
– An empty loop is translated to an empty activity.
– A start-block is mapped to a Pick containing empty activities for each branch.
– An end-block is translated to a respective AND-, OR-, or XOR-block with each branch followed by a terminate activity.

Algorithm 4 describes the Structure-Identification transformation strategy. Line 1 initializes the annotated process graph. After that, a loop is iterated until the annotated process graph is reduced down to one activity. The reduction rules of Definition 6 are used to

Algorithm 4 Pseudo code for Structure-Identification strategy

```
procedure: Structure-Identification($PG$)
1: $APG \leftarrow (S, E, F, C, l, A, \emptyset)$
2: while $|F \cup C \cup B| > 1$ do
3:     $APG' \leftarrow match(APG)$ {Using rules in Definition 6}
4:     $b \leftarrow translate(APG')$ {Using the described translations above}
5:     Reduce APG substituting APG' with b {Using rules in Definition 6}
6: end while
7: return($BCF$)
```

substitute components of the process graph by corresponding BCF structured activities in the same way as the function f_C substituted components in Definition 6.

4.4 Strategy 4: Structure-Maximization

The *general idea* of this strategy is to apply the reduction rules of the Structure-Identification strategy as often as possible to identify a maximum of structure. The remaining annotated process graph is then translated following the Element-Preservation or Element-Minimization strategy. The *advantage* of this strategy is that it can be applied for arbitrary unstructured process graphs as long as its loops can be reduced via the reduction rules of Definition 6. Still this strategy is also not able to translate arbitrary cycles, i.e. cycles with multiple entrance and/or multiple exit points. A *drawback* of this strategy is that both the Structure-Identification strategy and at least the Element-Preservation strategy needs to be implemented. The strategy could be used in *scenarios* where models have to be edited by a BPEL modeling tool that uses structured activities as the primal modeling paradigm.

5 Related Work

A lot of work exists on transformation between BPEL and other process languages. We highlight only a part and refer to [MLZ05] for a more complete discussion of related work. One branch of related work is dedicated to *model-driven development* of executable BPEL process definitions. In [Gar03] a BPM-specific profile of UML is used to generate BPEL code. The aim is rather to prove the feasibility of such an approach than the discussion of different transformation alternatives. It is not clear from the paper which strategy the author choose. The code fragments suggest that an Element-Preservation strategy is taken and sequences are mapped to BPEL sequence. The Element-Preservation strategy can also be found in a mapping from EPCs to BPEL [ZM05]. The BPMN specification [Whi04] comes along with a proposal for a mapping to BPEL. As BPMN is a graph-oriented BPM language similar to process graphs, the strategies of Section 4 can be applied. The subsection 6.17 of BPMN spec presents a mapping that is close to the Structure-Identification strategy proposed in this paper. The authors introduce so-called conceptual tokens to identify structure. Yet, the mapping is given rather in prose, a precise algorithm and a definition

of required structural properties is missing. Further work using a Structure-Identification strategy is reported in [vdAJL05] where Workflow nets, and in [KvM05] where XML nets are mapped to BPEL.
A second branch of research is related to *conceptual mappings* in order to better understand BPEL behavior and its relation to other BPM languages. In [HSS05] a transformation from BPEL to Petri Nets is presented in order to give BPEL formal semantics. The authors use a Flattening strategy to generate a Petri Net that covers BPEL behavior including exceptional behavior. The generated Petri Net is used for formal static analysis of the BPEL model.

6 Conclusion and Future Work

In this paper, we addressed the problem of transformations between graph-oriented and block-oriented BPM languages. In order to discuss such transformations in a general way, we defined process graphs as an abstraction of graph-oriented BPM languages and BPEL control flow as an abstraction of BPEL that shares most of its concepts with block-oriented languages like BPML. Our major contribution is the identification of different transformation strategies between the two BPM modelling paradigms and their specification as pseudo code algorithms. In particular, we identify the Flattening, Hierarchy-Preservation, and the Hierarchy-Maximization strategy for transformations from BPEL control flow to process graphs. In the other direction we identify Element-Preservation, Element-Minimization, Structure-Identification, and Structure-Maximization strategy. As such, the strategies provide a useful generalization of many current X-to-BPEL and BPEL-to-Y papers not only for identifying design alternatives but also for discussing design decisions. We checked the applicability of these strategies in two case studies which are reported in [MLZ05].
In future research, we aim to conduct further case studies in order to identify how aspects that are not captured by process graphs and BPEL control flow can be addressed in a systematic way. Another issue is the upcoming new version of BPEL which is expected to be issued as a standard in the beginning of 2006. It will be interesting to discuss in how far that new version simplifies or complicates the mapping to and from graph-oriented BPM languages.

References

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic and S. Weerawarana. Business Process Execution Language for Web Services, Version 1.1. Specification, BEA Systems, IBM Corp., Microsoft Corp., SAP AG, Siebel Systems, 2003.

[Ark02] A. Arkin. Business Process Modeling Language (BPML). Spec., BPMI.org, 2002.

[Gar03] Tracy Gardner. UML Modelling of Automated Business Processes with a Mapping to BPEL4WS. In *Proceedings of the First European Workshop on Object Orientation and Web Services at ECOOP 2003*, 2003.

[Hol04] David Hollingsworth. *The Workflow Handbook 2004*, chapter The Workflow Reference Model: 10 Years On, pages 295–312. Workflow Management Coalition, 2004.

[HSS05] Sebastian Hinz, Karsten Schmidt and Christian Stahl. Transforming BPEL to Petri Nets. In *Proceedings of BPM 2005*, LNCS 3649, pages 220–235, 2005.

[KNS92] G. Keller, M. Nüttgens and A. W. Scheer. Semantische Prozessmodellierung auf der Grundlage "Ereignisgesteuerter Prozessketten (EPK)". Heft 89, Institut für Wirtschaftsinformatik, Saarbrücken, Germany, 1992.

[KvM05] Agnes Koschmider and Marco von Mevius. A Petri Net Based Approach for Process Model Driven Deduction of BPEL Code. In Robert Meersman, Zahir Tari and Pilar Herrero, editors, *OTM Workshops*, volume 3762 of *Lecture Notes in Computer Science*, pages 495–505. Springer, 2005.

[MLZ05] J. Mendling, K. Lassen and U. Zdun. Transformation Strategies between Block-Oriented and Graph-Oriented Process Modelling Languages. Technical Report JM-2005-10-10, WU Vienna, October 2005.

[MN05] Jan Mendling and Markus Nüttgens. EPC Markup Language (EPML) - An XML-Based Interchange Format for Event-Driven Process Chains (EPC). Technical Report JM-2005-03-10, WU Wien, Austria, 2005.

[MNN04] Jan Mendling, Markus Nüttgens and Gustaf Neumann. A Comparison of XML Interchange Formats for Business Process Modelling. In F. Feltz, A. Oberweis and B. Otjacques, editors, *Proceedings of EMISA 2004*, LNI 56, pages 129–140, 2004.

[MZ05] J. Mendling and J. Ziemann. EPK-Visualisierung von BPEL4WS Prozessdefinitionen. In *Proc. of Workshop on Software Reengineering, Germany*, 2005.

[RtHEvdA05] Nick Russell, A.H.M. ter Hofstede, D. Edmond and Wil M.P. van der Aalst. Workflow Data Patterns: Identification, Representation and Tool Support. In *Proc. of the 24th International Conference on Conceptual Modeling (ER 2005)*, LNCS, 2005.

[Tha01] S. Thatte. XLANG. Specification, Microsoft Corp., 2001.

[vdA97] W. M. P. van der Aalst. Verification of Workflow Nets. In Pierre Azéma and Gianfranco Balbo, editors, *Application and Theory of Petri Nets*, LNCS 1248, pages 407–426, 1997.

[vdAJL05] Wil M.P. van der Aalst, Jens Bæk Jørgensen and Kristian Bisgaard Lassen. Let's Go All the Way: From Requirements via Colored Workflow Nets to a BPEL Implementation of a New Bank System. In R. Meersman and Z.Tari, editors, *Proceedings of CoopIS/DOA/ODBASE 2005, Agia Napa, Cyprus*, LNCS 3760, pages 22–39, 2005.

[vdAtH05] Wil M. P. van der Aalst and Arthur H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30(4):245–275, 2005.

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski and Alistair P. Barros. Workflow Patterns. *Distributed and Parallel Databases*, 14(1):5–51, July 2003.

[Whi04] S. A. White. Business Process Modeling Notation. Specification, BPMI.org, 2004.

[Wor02] Workflow Management Coalition. Workflow Process Definition Interface – XML Process Definition Language. Document Number WFMC-TC-1025, October 25, 2002, Version 1.0, Workflow Management Coalition, 2002.

[ZM05] J. Ziemann and J. Mendling. EPC-Based Modelling of BPEL Processes: a Pragmatic Transformation Approach. In *Proceedings of MITIP 2005, Italy*, 2005.

Automated Derivation of Executable Business Processes from Choreographies in Virtual Organizations

Ingo Weber[1], Jochen Haller[1], Jutta A. Mülle[2]
[1]SAP Research
[2]IPD, Universität Karlsruhe (TH)
Karlsruhe (Germany)
{ingo.weber, jochen.haller}@sap.com, muelle@ipd.uka.de

Abstract: In this paper, we address the challenge of deriving both, executable WS-BPEL processes and their respective WSDL interface specifications from choreographies written in WS-CDL for business processes in Virtual Organizations (VOs). The major issues hereby are the differences in the vocabulary of WS-CDL and WSBPEL as well as the information gap between a choreography and an executable orchestration. The information gap results from the requirement imposed by the VO environment to establish a process-based collaboration in a top-down fashion. High-level choreography descriptions are hereby the basis for the derivation of detailed executable processes. The first issue is addressed with a detailed translation table; the second, more severe one requires the use of a role specific knowledge base. This knowledge base delivers process subsets modeling detailed role internal activities while avoiding their exposure to collaborating roles. The combined solution is a CDL2BPEL algorithm.

1 Introduction

In today's business world, there is a strong need for information technology integration across organizational boundaries. Following the trend of Service Oriented Architectures (SOAs), enterprises will continue exposing well-defined communication interfaces to their respective business partners, in order to enable the information exchange and thus the co-operation of applications on geographically distributed information systems. Emerging standards mainly from OASIS[1] and W3C[2], especially in the Web service area, allow for a cross-domain business collaboration based on open standards. However, there have to be some explicit, harmonized facts and coherency in the interactions between the systems of several partnering roles: For application interoperation, a non-legally binding contractual agreement has to be followed, guaranteeing a common understanding of which information has to be communicated and when. This contractual agreement can take the form of a *choreography*, specifying the interactions and local activities between all roles involved in a collaborative business process at a higher level. I.e., only the interaction points and their respective order are defined in the choreography and not the details of the local activities.

[1]http://www.oasis-open.org/
[2]http://www.w3.org

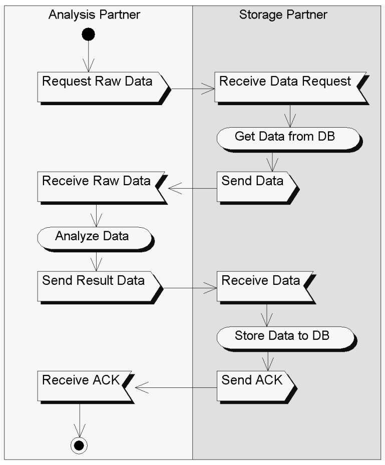


Figure 1: The Analysis-Storage choreography part

A simple example from Collaborative Engineering is shown in Figure 1. The graph, modeled as an UML activity diagram[3], shows an excerpt of a choreography that involves the roles Analysis (*AP*) and Storage Partner (*SP*). Assume, the Analysis Partner receives the request to perform an analysis on engineering data stored within the Storage Partner's domain. The raw data is referenced uniquely, and *AP* has the reference information. Now, *AP* requests data from *SP*, who retrieves it from the local storage facility and sends it to *AP*. *AP* then locally performs the actual analysis work. The results are transmitted back to *SP*, who in turn stores them in the local database. This example will be further developed throughout the remainder of the paper.

At the choreography level, the partners specify the global view of their co-operation instances. That is, each time when a business objective arises, a new instance of a collaborative business process is created and executed by each collaborating role involved in the choreography template. However, a choreography can be seen as the combination of a set of public interfaces and is not executable as process: it only contains the public knowledge for all roles required to participate in a collaboration, but not the private knowledge specifying the detailed activities performed by each single role within its own domain. E.g., in Figure 1 the interactions between the two roles are stated, but the knowledge on how the activity 'Analyze Data' is to be done is of no relevance to the choreography and thus not modeled at this level. Therefore, an executable representation of the business process of each role or partner needs to be created. In any of the cases, the local business process representations have to contain the local knowledge that is not present in the choreography.

This research work was conducted in the context of *Virtual Organizations (VOs)* within the European Union funded IST project TrustCoM, which imposes specific requirements: A VO is formed in response to a business objective that can not be addressed by just one partner alone. A swift reaction to the emerging business need, fast partner consortium formation, and quick, automatic adaptation of IT infrastructures are of essence. Thus, an automated solution deriving executable business processes from a choreography was de-

[3]Unified Modeling Language, see http://www.uml.org/

sired, following a top-down approach which is aligned with the VO formation and partner selection processes [RKH05].

Business application logic is hereby encapsulated in a service implementation. Therefore, a business process at one role can be seen as the ordering structure around local web service invocations, also called *orchestration*. Orchestration captures the local, role specific view on business processes, which orders the calls on available services and guarantee a defined execution order. In contrast, the global view encompasses the collaborative business process, which orders the interactions between the involved roles.

Given a choreography, the presented approach generates executable processes for each role along with public views on them. The underlying process model follows the view based process model as introduced in [SO02], where private, executable processes maintain their confidentiality by only exposing views in terms of public processes to collaborating roles. The required local knowledge for role specific processes is introduced via a *Knowledge Base (KB)*. Although the approach focusses on the specific needs of virtual organizations, it is applicable in most other scenarios as well. The VO environment imposes in that sense more challenges on the solution, since all aspects of executable processes, including the local knowledge of the partners, have to be available and specified already at derivation time.

The remainder of this paper is structured as follows: The problem statement, i.e., choreography support for VOs along with basic definitions form Section 2. Section 3 describes our solution comprising the local knowledge bases and the CDL2BPEL algorithm, which is based on the transformation rules available in the full version of this work in [Web05]. Related work is presented in Section 4. Section 5 outlines future work and concludes.

2 Choreography support for Virtual Organisations

Definition 1: A **Virtual Organization** is a combination of various parties (persons and/or organisations) located over a wide geographical area which are committed to achieving a collective goal by pooling their core competencies and resources. The partners in a Virtual Organisation enjoy equal status and are dependent upon electronic connections (ICT infrastructure) for the co-ordination of their activities. (From [BvW98]). The conducted research was motivated by the VO environment introduced within the EU-funded project *TrustCoM*. TrustCoM focuses on VOs tackling collaborative projects in swift reaction to an emerging business opportunity. The life cycle of a VO is typically divided into four phases (from [SLS98]):

- During the **VO Identification Phase**, the opportunity is identified, evaluated, and selected.
- The **VO Formation Phase** comprises of the partner identification, evaluation, and selection.
- In the **VO Operation & Evolution Phase**, services and resources of the single VO partners are integrated, and VO-wide collaborative processes aim at the achievement

of shared business objectives. Membership and the structure of VOs may evolve over time in response to changes of objectives or to adapt to new opportunities in the business environment.

- The **VO Dissolution Phase** is initiated when the market opportunity is fulfilled or has ceased to exist. Here, the distribution of results and products takes place, along with billing and the like.

Initially, in Identification and Formation phase the VO is assembled. This involves that a VO initiator, e.g. an aerospace industry system integrator, selects suitable partners to fill the roles required to enact business processes during the operational phase. Such roles may range from simple storage providers to specialised experts in subsystem, for instance wing, fuel tank or antenna, design and manufacturing. The partners are not required to have an existing business relationship of any kind. Departing from the highest level, fitting real partner organizations to roles during identification phase, the subsequent VO formation becomes more detailed when ICT, security, trust, and various other infrastructures and systems of the partners have to be connected [RKH05]. Consequently, the top-down approach also holds for the establishment of collaborative business processes. Departing from a high-level choreography description available to the VO initiator, it is required to derive executable business processes for each participating role. Since VOs are dynamic environments which are intended to act and form fast upon emerging business opportunities, an automatic derivation methodology is highly desired.

Definition 2: A **choreography** describes collaborations of parties by defining from a global viewpoint their common and complementary observable behavior, where information exchanges occur, when the jointly agreed ordering rules are satisfied. [...] The Choreography offers a means by which the rules of participation within a collaboration can be clearly defined and agreed to, jointly. Each entity may then implement its portion of the Choreography as determined by the common or global view. (from [W3C05]).

In TrustCoM, the view based collaborative business process model is used, as presented by Schulz et al in [SO02], [SO01], which, in turn, is based on the specifications of the WfMC[4]. There, the needs for confidentiality of entire processes or workflows of the respective partners and the integration of multiple private workflows into a global view are identified as critical for successful operation of virtual enterprises, extended enterprises and virtual organizations. On the one hand, an organization may not be willing to share detailed information about a complete business process, since the information in it represents an asset to its owner. On the other hand, enough information has to be provided to the coalition (or the VO in our context) in order to get a coherent and stable public workflow. The TrustCoM approach introduces a coalition model with three tiers: private processes, public views of these processes, and a (global) public process. These three tiers correspond to the notions of private business processes, the interfaces of these processes, and choreographies, respectively.

Applying the collaborative business process model to the VO environment, the collaborating members are informed about the VO objective through the shared choreography. They

[4]Workflow Management Coalition, among others see [WfM99] and [WfM02]

can thus infer the behavior that is expected from them during the VO operation phase which corresponds to their public process. A partner is only required to expose the public process to the VO which serves as the interface for the confidential private process.

Following the Service Oriented Architecture (SOA) paradigm, implemented modular pieces of application logic are assumed to be available as services. Therefore, the private business process can be seen as a stateful wrapper around the services, guaranteeing the defined order of calls to the services. Thus, it is also called **orchestration**, providing a local, role specific view on a private/public process pair, in contrast to the choreography which captures the global view on a collaboration among different roles.

2.1 The Information Gap

The information gap is our term of different levels of detail within the described collaborative business process model. Choreographies model the interplay between the multiple parties in a collaborative business process and are not concerned about the details of each individual role's activities. Orchestrations, however, need to contain all details required for the execution of a single partner's business process. Thus, the sum of the orchestrations contains more knowledge than the respective choreography.

In detail, the information gap contains the following points of information differences between the choreography and the orchestrations:

1. The internal or private actions for each role, which are of no interest to the choreography.
2. Branching conditions in the orchestration, which are not observable on the collaborative level.
3. Extensions, specifying e.g. annotating security requirements and transactional behavior.
4. Details in error and compensation handling, which are not treated at the choreography level.
5. Runtime details, such as initial and glueing activities, which monitor e.g. service endpoint behaviour.

While the points 1 and 2 are addressed by the work presented in the following section, 3 is subject to future work. Point 4 should in our opinion be addressed by best practices for choreography design, and 5 tackles runtime behaviour which is considered out of scope for the presented work.

2.2 Language Differences

From the extensive set of available languages[5], we made the following three choices addressing the requirements and constraints as presented above in this section:

- The **Web Service Choreography Description Language (WS-CDL)** [W3C05] is used to specify choreographies. Although it is still in the process of standardization and severe structural critique was expressed [BDO05], it represents the most promising and expressive approach currently available.
- For executable (private) processes, the **Web Service Business Process Execution Language (WSBPEL)** is employed. All generated private processes are executable BPEL processes. This language was chosen, because it is mature, widely known and used, and draws strong attention from industry.
- The public views are expressed in the **Web Service Description Language (WSDL)**. Other possibilities such as abstract BPEL processes were considered and would have been more flexible than static WSDL descriptions. This choice was taken, since WSDL is a rather mature and stable standard, at the same time supporting the choice of executable BPEL. Abstract BPEL is specified, but its intended usage and coherent runtime mapping to executable BPEL is still under discussion in the OASIS Technical Group standardizing WSBPEL.

In order to qualify elements and functions in the remainder of this paper, the following prefixes are employed: *cdl* refers to language items from WS-CDL, *bpel* refers to WSBPEL, and *wsdl* to WSDL language elements.

Although the WS-CDL specification intends to be compliant with executable process languages as WSBPEL, the language elements differ in many respects. Where BPEL offers a set of atomic activities which can be combined elegantly to the desired behavior, WS-CDL knows certain elements which are far from being atomic: The cdl:interaction activity combines actual interactions with transactionality, timeouts, and assignments. cdl:WorkUnits are the sum of loops, conditional execution, variable value evaluation, and potential partial ordering of parallel activities. These and other differences make the actual derivation of executable BPEL processes hard, since the complex semantics of very expressive WS-CDL activities are not matched in the goal language. In certain cases, workarounds are possible. But in a subset of these cases, the workarounds cannot guarantee that the behavior of the generated processes is fully equivalent to the intended behavior from the choreography. The resulting semantic differences are outlined in the evaluation section.

In this section, the introduced terms were defined and the problems addressed by this paper were stated. Namely, there is the issue of the information gap as a result of the different points of view taken by choreographies and orchestrations, and the problem of language

[5]As for XML-based standard languages (or languages in the process of becoming standardized), there are multiple alternatives for each category, such as WSCI, WSCL, BPSS, and WfXML for choreographies and BPML, XPDL, XLANG, WSFL, and BPDM for executable private processes.

differences resulting from the chosen set of languages. The next section is concerned with the solutions to these problems.

3 Technical Solution

In order to achieve an automated derivation of executable processes, the elements of a WS-CDL document are translated in a depth-first search through the XML tree. For each role in the choreography, a process is derived. Each element in the source document is added only to the processes of the roles for which it is relevant. If a part of the choreography cannot be translated, i.e. it falls into one of the categories of the information gap, it is sent as a request to a **Knowledge Base (KB)**. In the KB, the private, local, or confidential details of a private process are placed. If there are elements of a choreography which still cannot be translated, a list of these elements (and potentially other errors) is returned.

Figure 2 shows a graphical representation of the outcomes of the BPEL derivation from the choreography in Figure 1. Clearly, the orchestrations are far more detailed, and the local substitutes for the high-level, private activities from the choreography are in place (the local service calls, which depend on the environment).

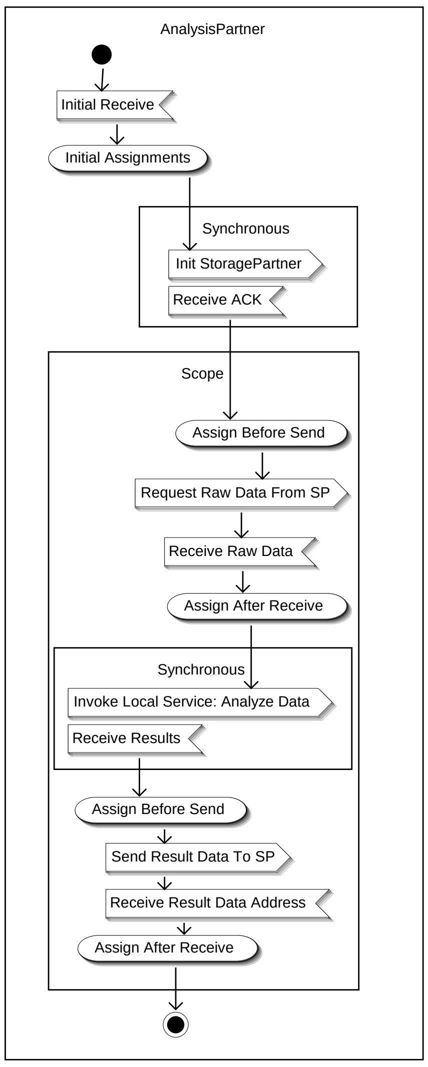
Also, the executable processes are concerned with initialization of the processes, which always means incoming messages in BPEL[6], as well as gluing activities. The latter include variable and message initialization, and apparent assignment statements. These runtime requirements are not to hard to meet, and the details of how this can be done are omitted here.

The derivation of BPEL and WSDL from WS-CDL is achieved in a 5-step algorithm, which is outlined in section 3.2. In short, the algorithm is an extended compiler, in that it reads a source document and generates an object tree for it, performs validation and transformation on the tree, and serializes the resulting object trees to a set of documents in the target languages. We call it 'extended', because it includes a dynamic part - the Knowledge Base - in addition to the static program code.

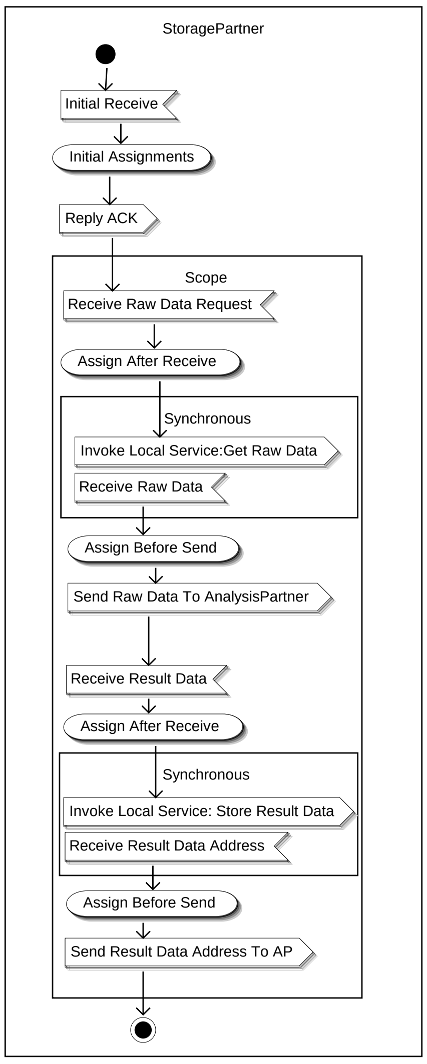
3.1 The Knowledge Base

The KB contains CDL patterns and their respective replacements in terms of BPEL activities as well as optional WSDL elements and even deployment artifacts for all roles of interest. When queried, the KB tries to find a pattern matching the WS-CDL part from the request. If such a pattern is found, the respective BPEL and WSDL parts and the deployment information are retrieved. Since the patterns can be generic in that they contain placeholders for variable or partner names, the KB then replaces these placeholders with the instances from the query. Subsequently, the results are returned to the CDL2BPEL algorithm, which weaves them into the goal documents.

[6]Note, that here only the AnalysisProvider's process needs to be invoked from outside. It then calls the SP process synchronously, so that one can be sure, all processes are available before starting the work.



(a)



(b)

Figure 2: The resulting BPEL processes, derived from the Analysis-Storage choreography part in Figure 1. **(a)** The process for the AnalysisPartner role. **(b)** The process for the StorageProvider role.

Using this technique encourages the reuse of both choreographies and patterns and introduces a dynamic element into the derivation. The KB can be deployed and accessed solely locally at each member of a VO, satisfying the confidentiality requirement that comes with optimized process parts and internal implementation. The KB also enables a late binding-like way to connect local services to a process. In that sense, the Knowledge Base can be seen as the material filling up the information gap.

3.2 The CDL2BPEL Algorithm

We did not attempt using XSLT[7] for the transformations due to its limitations. XSLT is designed for generating XML result documents out of XML source documents in a straight-forward manner. Here, we need more sophisticated mechanisms e.g. for validation: WS-CDL channel variables with a usage set to 'once' may actually be used more than once, and unrolling all the possible contingent execution paths that a choreography can take is a virtually impossible quest for XSLT. However, the implementation needs to be able to detect such errors in the choreography.

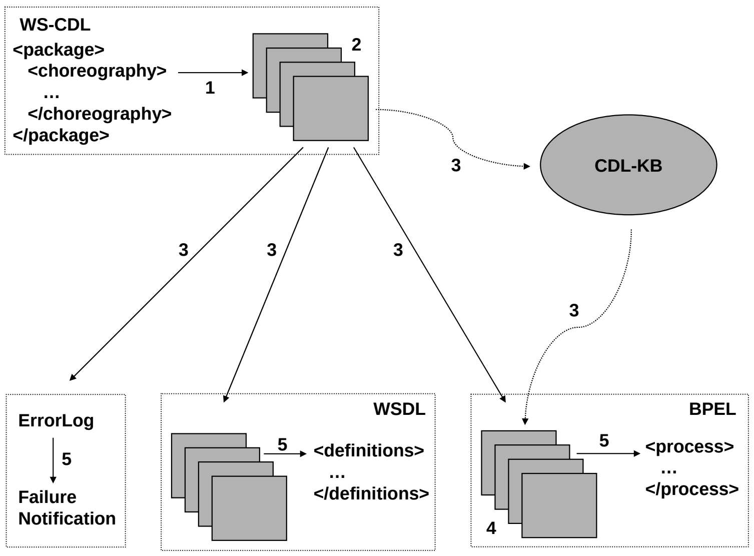


Figure 3: The five steps of the CDL2BPEL algorithm

Figure 3 shows the various steps of the algorithm graphically. In detail, the five steps are:

1. Read the choreography, create and initialize the corresponding Java objects from the WS-CDL elements.

[7]Extensible Stylesheet Language Transformations, see XSLT 1.0 [W3C99] and XLST 2.0 (http://www.w3.org/TR/xslt20/ , work in progress)

2. Validate the choreography (Correct variable and channel usage, the correct number of child elements with appropriate attributes and conditions such as guards)

3. Translate from CDL to BPEL and WSDL

   - Traverse the XML tree from the root choreography, adding each activity to the BPEL process of involved roles (Structuring activities, such as sequence or parallel, are added to the processes of all roles)
   - If the current element cannot be translated directly, try a KB lookup with the non-translatable part as input
   - For activities or sets of activities which still cannot be transformed, make an error note (Report all errors back after traversing the whole tree, see step 5)
   - Extract WSDL files from interactions and tokens / token locators (Generate operations, port types, message schemas, bpel:partner link types, bpel:properties)

4. Validate the generated BPEL processes from the holistic perspective, focussing on the partner links, operations, port types and exchanged variables. This static check ensures that the set of BPEL processes can be executed together based on their sequence of exchanges.

   - Add necessary gluing activities where obvious, e.g. assignments.
   - Remove superfluously structuring activities (e.g. a sequence with one child) which are leftovers from step 3).

5. Generate the BPEL and WSDL code from the objects OR return failure note

The results of this algorithm are a set of documents, namely for each cdl:roleType a private, executable BPEL process and the public view on it as a WSDL definition. Note that WS-CDL knows an element called *participant*, which groups together multiple role types that, during execution, must be played by a single entity. Therefore, another approach could be to generate one process per participant. However, we decided on the above solution, because one organization representing a participant with multiple roles could work with multiple BPEL engines for the differing purposes of the roles. E.g. in a buyer-seller-shipper choreography where the seller and shipper roles have to be played by one entity, the company who plays these roles could have distinct departments with each maintaining an own BPEL engine. Thus, this decision can be seen as enabling more flexibility, but shifting the enforcement of the participant-role constraints to another spot.

For further reading, the details of the translation are listed in the translation table in [Web05]. For most of the WS-CDL elements, a match in BPEL was found. In particular, the cdl:silentAction and cdl:choice with non-observable conditions are cases for which, by design of WS-CDL, no match can be found in BPEL. One other construct has to be mentioned here as well: A cdl:choice with both, blocking and non-blocking cdl:workUnits as children can only be translated to a workaround - a set of activities in BPEL which can differ in their execution from the intended behavior in the choreography. This is only the case if certain complex interdependent sets of guard conditions fall together with events taking place at particular points in time. It is the most challenging case of language differences between WS-CDL and BPEL, and only for that we did not find a perfectly satisfying solution.

3.3 Prototypical Implementation and Evaluation

For both, the CDL2BPEL algorithm and the KB, a proof-of-concept prototype has been implemented as Java Web Services based on Axis (1.2RC3) ([Fou05]). The Knowledge Base builds on a relational database. The business processes in Figure 2 demonstrate graphically the result of applying the implementation to the CDL document belonging to the choreography from Figure 1.

[Web05] gives a conceptual mapping for all WS-CDL elements to BPEL and WSDL. The CDL2BPEL algorithm uses this mapping and the Knowledge Base in order to derive executable processes and their interfaces from choreographies. The prototype proves the validity of this approach for the addressed problem by applying the implementation to a set of Collaborative Engineering choreographies within a TrustCoM VO. The outputs are processes which are indeed executable.

However, for certain cases of the cdl:choice element, no semantically equivalent construct was found. That is, a possible workaround could differ in its behavior as follows:

- Timing issues can switch the order of condition evaluation, since there are no event-based BPEL constructs matching the cdl:isVariableAvailable and cdl:variablesAligned functions.
- Communication delays in Web Service invocations can cause a duration-based time-out to happen later than intended or even missing an event completely.

A solution to these issues requires basically changes or extensions in BPEL, thus being postponed until the next version[8] of BPEL becomes available.

As already mentioned above, the implementation was tested with TrustCoM choreographies, stemming from collaborative engineering examples. The choreographies we used departed from simple examples with two roles and eleven activities up to more complex examples with five roles and 40 activities, not counting interactions. In principle, the approach of the CDL2BPEL algorithm is valid beyond the TrustCoM related samples. The translation table is independent of any application scenario as is the Knowledge Base concept. To apply a particular implementation instance in an application scenario, the only application specific dependency relates to the Knowledge Base content. An occurring cdl:silentAction to ”Analyze data” for instance will be resolved by a Knowledge Base query and therefore, its content has to deliver the fitting process part. ”Analyze data” may be relevant for an aerospace choreography as wells as an automotive one, but the process parts will differ depending on the example.

[8]The OASIS WSBPEL Technical Committee plans two further versions of the BPEL standard, before handing over the control to W3C.

The evaluation has shown, that in most practically relevant cases the algorithm yields executable processes. Exceptions, besides above cdl:choice example, may arise with the use of the *align* attribute in cdl:interactions which express required transactional behaviour. Translated to BPEL, this involves the usage of compensation handlers and rollback variables storing the used variable's initial state. While BPEL alone can cope with simple transactions, we believe that in complex collaborations involving for instance long running or dynamic transactions among multiple roles would require the use of other WS standards such as WS-Transaction in conjunction with WS-Coordination. Another exception is the channel concept in WS-CDL which has no direct counterpart in BPEL. Furthermore, tokens can be declared in WS-CDL which relate to correlation sets in BPEL. In contrast to CDL, an activity which initialises a correlation set has to be explicitly defined [Web05].

4 Related Work

The main issue of this paper is the automated derivation of executable business processes specified in WSBPEL and WSDL from WS-CDL modelled business choreographies in an interorganizational environment.

There are several publications on issues related to interorganizational workflows ([ACea04, CCJL04, vdA00, vdAW01]). The overlapping and differing aspects of orchestrations and choreographies were already identified in [Pel03], but focussing on WSCI [W3C] and not on WS-CDL.

The tranformation of choreographies to WSBPEL is part of a comprehensive design methodology. Several approaches exist that propose such architectures for Business Process Modeling. As one of these, [Hav05] provides an exemplary BPM architecture, which is built on three standards: WS-CDL, the Business Process Modeling Notation (BPMN), and BPEL. This theoretical architecture envisions automated mappings from WS-CDL to BPMN (and compliance checks in the opposite direction), as well as from BPMN to BPEL. This approach is motivated as a good BPM solution for single companies, whose processes must comply to agreed-on choreographies for the interaction with business partners. Their architecture is more of a bottom-up approach, whereas our solution is meant for VOs, with intrinsic necessity for top-down solutions. Also, the author states that "BPMN has no behind-the-scenes open XML representation"[9] which could be used for automated BPMN generation, based on a given WS-CDL definition.

The ebXML BPSS[10] offers a public view on business processes during design time which has some similarities to the choreographies used in our approach. In contrast to WS-CDL, the ebXML business processes can only specify binary relations, posing restrictions to the expressive power for ordering constraints of multi-party choreographies. Furthermore, ebXML does not explicitly address executable business processes, e.g. modeled in BPEL. Businesses mutually agree to follow a BPSS in a CPA[11]. Instead of striving for automation

[9][Hav05], p. 41

[10]ebXML: Electronic Business using eXtensible Markup Language (http://www.ebxml.org/), BPSS: Business Process Specification Schema. See [Ira01], [Mar03], and [NDE+01]

[11]Collaboration Protocol Agreement, part of ebXML

in setting up a collaboration, ebXML rather expects people to agree upon a CPA, taking existing executable processes into account.

In [MNN04], 15 different XML-based specifications for business process modeling are compared. The authors state, that WSBPEL is one of the most complete language in terms of available features, that supports our choice of BPEL as target language. The already obsolete BPML[12] co-exists, but receives far less support from the industry, possibly because it is not directly related to Web Service orchestration.

A conceptual model for the mapping from WSBPEL to WS-CDL is presented in [MH05]. Additionally, a proof-of-concept implementation of the mapping was realized with the Extensible Stylesheet Language Transformations (XSLT), enabling the generation of BPEL stubs from WS-CDL documents. Although the goal seems similar to our approach at a first glance, there are notable differences: our goal is to derive executable BPEL processes that can be deployed without human interaction and are executable in the sense that they really run through when called, and we provide a complete translation table not only a partial one as in [MH05]. Also, the prototypical implementation of our mapping is implemented as an extended compiler, offering dynamic translation opportunities and sophisticated consistency and correctness validation - qualities that XSLT cannot provide without extensions.

5 Conclusion and Future Work

The paper describes the task of deriving executable public and private processes from a high-level choreography, and presents a solution in form of the CDL2BPEL algorithm. In conclusion, the main challenges initially pointed out, namely overcoming in the first place the information gap when departing from high-level choreography descriptions could be solved with the introduction of the knowledge base. The latter fills the gaps between the global and the local collaboration perspective with partner-internal local knowledge. Both, knowledge base and the Web service oriented implementation of the CDL2BPEL algorithm achieve the remaining goal of an automated derivation approach within highly dynamic VO environments.

Future work is planned to build incrementally upon the presented derivation approach. As an immediate extension, we planned automatic process, private and public, deployment within a process execution environment. Technologywise, the choice of BPEL for private and WSDL for public processes introduces a large number of possible BPEL engines for this purpose. In the case at hand, the open source ActiveBPEL (http://www.activebpel.org) engine was chosen after careful testing due to its stable implementation and sensible architecture. Furthermore, TrustCoM aims at provisioning of secure collaborative business processes. The confidentiality of private processes can be enforced through their deployment environment. Process execution at runtime needs to be reactive to security subsystem decisions taken outside the process execution environment, but within the same administrative domain. A security control concept for collaborative business processes is already available and was published by one of the authors [HRK05]. It will be implemented based on the presented work.

[12]Business Process Modeling Language, [Ark02]

Far-term future work could include basing the knowledge base on BP-focussed semantic descriptions of the tasks to be fulfilled. Therefore, the patterns could be generic, in that they would be compared with a functional description of a part of the choreography.

References

[ACea04] G. Alonso, F. Casati, and et al. *Web Services - Concepts, Architectures and Applications*. Springer, 2004.

[Ark02] Assaf Arkin. Business Process Modeling Language. San Mateo, CA: BPMI.org, 2002. Proposed Final Draft.

[BDO05] Alistair Barros, Marlon Dumas, and Phillipa Oaks. A Critical Overview of the Web Services Choreography Description Languages (WS-CDL). BPTrends Newsletter, Vol. 3, March 2005.

[BvW98] René Bultje and Jacoliene van Wijk. Taxonomy of Virtual Organisations, based on definitions, characteristics and typology. VoNet: The Netwsletter @ http://www.virtual-organization.net/, 1998.

[CCJL04] L.F. Cabrera, G. Copeland, J. Johnson, and D. Langworthy. Coordinating Web Services Activities with WS-Coordination, WS-AtomicTransaction, and WS-BusinessActivity, 2004.

[Fou05] Apache Software Foundation. Web Services - Axis, November 2005.

[Hav05] Mike Havey. Essential Business Process Modeling. Copyright 2005 O'Reilly Media, Inc, 2005.

[HRK05] Jochen Haller, Philip Robinson, and Yuecel Karabulut. Security Controls in Collaborative Business Processes. In *6th IFIP Working Conference on VIRTUAL ENTERPRISES (PRO-VE'05)*, 2005.

[Ira01] Romin Irani. Collaborative Electronic Business is here to stay: An Introduction to ebXML. http://www.webservicesarchitect.com/content/articles/irani02.asp, 2001.

[Mar03] Benoit Marchal. An Introduction to the ebXML CPP. http://www.developer.com/xml/article.php/2247851, 2003.

[MH05] Jan Mendling and Michael Hafner. From Inter-Organizational Workflows to Process Execution: Generating BPEL from WS-CDL. In *Proceedings of OTM 2005 Workshops. Lecture Notes in Computer Science 3762*, pages 506–515. Springer Verlag, October 2005.

[MNN04] Jan Mendling, Markus Nüttgens, and Gustaf Neumann. A Comparison of XML Interchange Formats for Business Process Modelling, 2004.

[NDE^{+}01] Duane Nickull, Jean-Jacques Dubray, Colleen Evans, Pim van der Eijk, Vivek Chopra, David A Chappell, Betty Harvey, Marcel Noordzij, Jan Vegtand, Tim McGrath, and Bruce Peat. *Professional ebXML Foundations*. Wrox Press, 2001.

[Pel03] C. Peltz. Web Services Orchestration and Choreography. *Computer*, 36(10):46–52, 2003.

[RKH05] Philip Robinson, Yuecel Karabulut, and Jochen Haller. Dynamic Virtual Organization Management for Service Oriented Enterprise Applications. In *to appear: The First International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2005)*, 2005.

[SLS98] Troy J. Strader, Fu-Ren Lin, and Michael J. Shaw. Information Infrastructure for Electronic Virtual Organization Management. *Decis. Support Syst.*, 23(1):75–94, 1998.

[SO01] Karsten A. Schulz and Maria E. Orlowska. Architectural Issues for Cross-Organisational B2B Interactions, 2001.

[SO02] Karsten A. Schulz and Maria E. Orlowska. Towards a Cross-Organizational Workflow Model. In *PRO-VE '02: Proceedings of the IFIP TC5/WG5.5 Third Working Conference on Infrastructures for Virtual Enterprises*, page 652. Kluwer, B.V., 2002.

[vdA00] W. M. P. van der Aalst. Loosely Coupled Interorganizational Workflows: Modeling and Analyzing Workflows Crossing Organizational Boundaries. *Information and Management*, 37:67–75, 2000.

[vdAW01] W.M.P. van der Aalst and M. Weske. *The P2P Approach to Interorganizational Workflows*, pages 140–156. Springer, 2001.

[W3C] W3C. *Web Service Choreography Interface (WSCI) 1.0.* W3C Note 8 August 2002.

[W3C99] W3C. XSL Transformations (XSLT) Version 1.0, 1999. W3C Recommendation 16 November 1999.

[W3C05] W3C. Web Services Choreography Description Language, 2005. W3C Latest Working Draft from October 8th, 2005, work in progress.

[Web05] Ingo Weber. Automation in Collaborative Business Process Instantiation. Master thesis at the department of informatics, Universität Karlsruhe, Germany, November 2005.

[WfM99] WfMC. Interface 1: Process Definition Interchange Process Model. Document number wfmc-tc-1016-p version 1.1 final, Workflow Management Coalition, 1999.

[WfM02] WfMC. Workflow Process Definition Interface - XML Process Definition Language. v1.0 Final Draft. Document number wfmc-tc-1025, Workflow Management Coalition, October 2002.

Integrating Process and Organization Models of Collaborations through Object Petri Nets

Kamyar Sarshar[1], Thomas Theling[1], Peter Loos[1], Mirko Jerrentrup[2]

Institute for Information Systems at the German DFKI
Stuhlsatzenhausweg, Geb. 43.8
D-66123 Saarbruecken, Germany
{sarshar|theling|loos}@iwi.uni-sb.de

Interactive Software Solutions
Saarterrassen, Hochstr. 63
D-66115 Saarbruecken, Germany
jerrentrup@interactive-software.de

Abstract: The management of virtual enterprises needs extended and integrated approaches of business modeling. While most formal approaches to business process modeling consider only the control-flow perspective, it is essential in an inter-organizational context to link tasks with the enterprise responsible for their execution. This paper presents the concept of the transforming BPMN-conform XML representation of process models and a proprietary OMN-XML representation of organization models into a special type of object Petri nets called Reference net. The benefit of our approach is that by using Reference nets, the control-flow and the inter-organizational perspective of a business process can be integrated into a unique formalism ready to by analyzed and simulated by appropriate Petri net tools like RENEW. After introducing the transformation concept its application will be demonstrated by an example.

1 Introduction

To manage virtual enterprises and the collaboration of businesses, existing concepts for business process management need to be adapted and extended. A theoretical framework for collaborations is the theory of transaction costs based on the work of Coase [Co38; Wi95]. Institutions of the market like corporations, governmental organizations, or legal conditions are analyzed in this theory. Rights, goods, or outputs are transferred (delivered) between organizations by transactions, which are based either on contracts or hierarchies. Therefore the modeling notation used in our approach must be able to depict these contracts and deliveries resp. hierarchies in an adequate way.

Within the project ArKoS[1], a core team of eight universities and companies have determined requirements for the design, implementation and evaluation of an architecture appropriate for collaborative businesses. An essential component of the established architecture [Th05a; Th05b] is a distributed repository, which provides collaboration-wide process and organizational models. A challenge is the interoperability of different modeling notations within the architecture: On the one hand, collaborating corporations may use different modeling tools applying different modeling languages. On the other hand, collaboration-wide and enterprise-intern models are notated in different modeling languages. For this reason, all collaboration-wide required models are transformed and stored in repository-wide unique data formats. By using converters, different modeling software and modeling notations can be integrated. The data format applied to the repository is a BPMN-conform [Wh04] XML format for business processes, while inter-organization models are stored in a proprietary XML format, called OMN-XML. Both formats have been defined within the project ArKoS and are described below.

Another task of the established architecture is to support management and controlling of collaborations. In early collaboration phases, organizational and process models can be used to simulate the collaboration's behavior a priori. A formal notation of organizational and process models is an Object Petri net [Va04]. Object Petri nets allow to formalize and to integrate process and organizational models within *one* notation which can be used for analysis as well as simulation by according Petri net tools like RENEW [Ku04]. The major advantage of our approach is that it has a broader view on business processes by formalizing not only their control-flow but also considering their organization perspective. This enables to consider aspects like capacity utilizations, to estimate lead-times by process simulation, or to deliver useful data for improving collaborations' processes and organizational structures.

This paper presents the concept on how to transform the repository-wide XML-formats for semi-formal process and organization models into two interrelated PNML files (Petri Net Markup Language) [Bi03] representing a Reference net [Ku02] which is a special type of object Petri nets [Va04].

The remainder of this paper is organized as follows: After this introduction, the next section will give an overview of related work and assorts used modeling languages of the established architecture. Chapter 3 introduces briefly the object Petri nets and Reference nets. Chapter 4 presents the developed transformation concept, while chapter 5 shows an example for the conversion by a converter implemented based on the introduced concept. Chapter 6 gives a summary and an overview of future work.

[1] http://www.arkos.info

2 Previous Research

The concepts of virtual enterprises and collaborative business [Wi95] are discussed on the basis of several economical theories like transaction costs [Wi95; Co38], the market-based view proposing strategic groups to evaluate cooperations' effects on the market [CP77], or the resource-based view emphasizing competencies of corporations [HP90].

In order to setup and maintain collaboration between enterprises to jointly produce goods and services, it is essential to represent workflow and business processes by appropriate notations [Aa02; EM00; RKC98]. For the purpose of business process modeling, several notations have been discussed within literature. Petri nets, originating from the early work of Carl Adam Petri [Pe62] have successfully been applied to process modeling, analysis and simulation by several authors [AHH94; Zi77]. A further approach to process modeling is BPMN (Business Process Modeling Notation) [Wh04], which is a graphical notation and defines activities as well as control flows to visualize business process operations.

However, the focus of business process modeling is the representation of the execution order of activities which is described through constructs for sequence, choice, parallelism or synchronization. When business processes are jointly performed in an inter-organizational context, it is also essential to link tasks with the enterprise responsible for its performance. Numerous authors have investigated such an integrated approach to process and organization models. Zur Muehlen [Mu04] proposes a workflow lifecycle considering an organizational (or resource) perspective in each stage. Van der Aalst describes an organizational model in UML and converts it into an XML DTD in order to integrate it with XML based representations of workflows [AKV03].

The idea of mapping system and object nets to the organization and process perspective of business processes has been sketched by a minor case study of the Dutch Justice department [Va98]. Van der Aalst generalizes the idea and introduces based on Reference nets a conceptual framework for inter-organizational workflow enactment by which different perspectives of workflows including control-flow, resource, data, task and operation can be represented by reference nets [Aa99].

An alternative approach is to use elementary Petri nets like Place/Transition nets for the purpose of representing the control-flow and the organizational perspective of a business process [AH02]. However, using elementary Petri nets for business process modeling purposes would lead to complex models with reduced readability. This has led to prefer high level petri nets for business process modeling purposes. And this is even more the case when in addition to control-flow the organizational perspective is considered. The advantage of the object Petri net approach is to have a formalism which integrates both perspectives while each perspective is represented by a distinct Petri net (system and object net). These are integrated by their dynamic behavior.

3 Object Petri Net Approach

Petri net is a generic term for a number of modeling techniques, graphical representations and notational conventions that are all based on the concept of net formalism introduced by Carl Adam Petri [Pe62]. Since their introduction Petri nets have been extensively investigated within the scientific community, whereby different extensions and applications were introduced [RR98]. A significant difference between the Petri net types is their token concept. Within elementary Petri nets, undistinguishable black tokens represent the availability of pre-condition of transitions, while at high-level Petri nets, tokens represent passive data structures which are transformed by transitions. The use of structured tokens permits the representation of more complex systems, while they are still passive and have no dynamic behavior. However, with the emergence of object-orientation some research has been conducted to combine Petri net models with the object-oriented paradigm [GV03]. The object Petri nets approach adds dynamic behavior to tokens by defining them as Petri nets again [Va04]. The approach has its origins in works describing the execution of task systems in systems of functional units [JV87]. It allows a multi-level modeling technique whereby one or more so called object nets move through a system net as ordinary tokens.

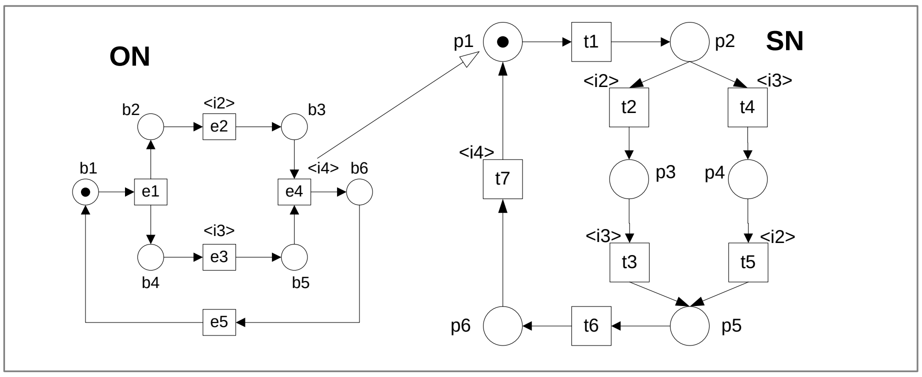


Figure 1: Object Petri Net Example [Va98]

Figure 1 gives an example of three possible interaction relations between the system and object nets of object Petri nets [Va98]. The object net (ON) illustrated on the left is located at the place p1 of the system net (SN). A label <i_n> synchronizes steps between the respective transitions of the object net and system net; a missing label indicates mutually autonomous steps. Since there is no such label at transition e1 and t1 an object autonomous step of the object net and a system autonomous step of the system net is possible. After theses steps, object and system net have reached a point where an interaction between the two levels at e2 and t2 as well as e3 and t4 are possible next steps.

A central characteristic of the approach is the distinction between reference and value semantics. The reference semantics restricts the system net to refer to identical copies of object nets. The dynamic behavior of the reference semantic is formalized by the bi-marking. On this basis, Reference nets have been developed and implemented by the

RENEW-Tool [Ku04]. The value semantics which allow instances of an object net to be independent copies use the p-marking in order to execute consistently. For more details on the object Petri net approach we refer to the referred literature.

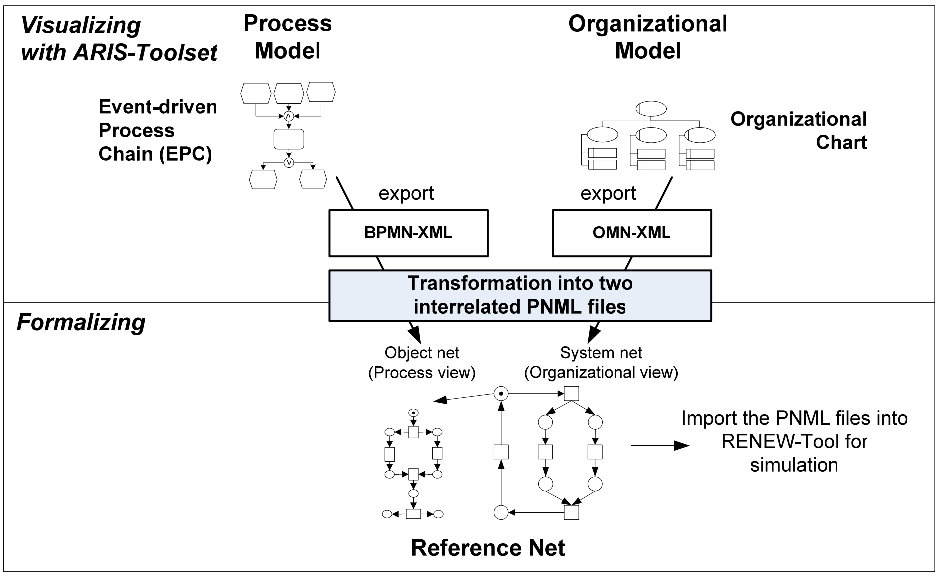
To be able to simulate the object and system nets in a simulation engine, we have decided to apply the Reference nets and use the RENEW-Tool which has an import interface for PNML files. We interpret the elements of the Reference nets in the following way:

The object net represents a process. Transitions of the object net correspond to tasks and the states the conditions, which have to be fulfilled in order to execute a task. Token of the object net illustrate the case (process object) that is transformed by the execution of the process [Aa98]. The system net represents the organization structure. In this context, each place of the system net is interpreted as a single enterprise connected with other enterprises through contract and delivery relations (transitions of the system net). The object autonomous step stands for the execution of the process within an organization. A system autonomous step transports the process (object net) from one organization to another without changing the state of the object net. By an interaction, a task of the process is executed while it is transported to another organization. In other words, an interaction causes transitions of the object and system nets to fire synchronously. The synchronous execution of Reference nets is realized by labeling the according transitions with *downlinks* in case of the system net and corresponding *uplinks* at the transitions of the object net.

4 Transformation Concept

4.1 Conceptual Framework

Within the ArKoS-architecture, inter-organizational process and organization models are stored in a common repository. Figure 2 illustrates different notations and their application within the ArKoS-project. It distinguishes between different conceptual levels. At a visualization level the ARIS-Toolset [Da01] is used for modeling inter-organization structures of the dependencies of collaborating enterprises as well as the processes performed jointly by them. In order to prevent the repository from being dependent from the ARIS-Toolset and the Event-driven Process Chain (EPC), process models are exported into BPMN-XML, while organization models are exported into OMN-XML. These formats which are introduced in detail in section 4.2 are used to store the models in the common, collaboration-wide available repository. The transformation concept we introduce in this contribution uses these two XML files as input and produces two interrelated PNML files representing an Reference net, which will be introduced briefly within the next section.

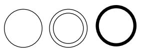
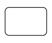


Visualizing with ARIS-Toolset
Process Model
Organizational Model
Event-driven Process Chain (EPC)
Organizational Chart
export
export
BPMN-XML
OMN-XML
Formalizing
Transformation into two interrelated PNML files
Object net (Process view)
System net (Organizational view)
Import the PNML files into RENEW-Tool for simulation
Reference Net

Figure 2: Selected Modeling Notations used in ArKoS

4.2 XML Representation of the Input Files

Table 1 illustrates the XML-representation of BPMN elements which is the format for process models. The included elements reflect basically the elements needed by the EPC since this was the notation used to model business processes with the ARIS-Toolset. Hence, from BPMN only events, activities, AND gateway and XOR gateway have an XML-representation. The only exception we made so far was to exclude the OR-connector because it has a non-local behavior [ADK02] and we have not developed a proper solution for its transformation yet.

BPMN	Graphical Representation	BPMN-XML-Representation
Element **EVENT** Subtypes *START, INTERMEDIATE, END*		`<ControlFlowObjectDefinition controlflow-object-type="EVENT" controlflow-object-definition-id="E1" controlflow-object-subtype= "start|intermediate|end" artifact-idlist="..." lane-idlist="..." linked-process-id="..."> </ControlFlowObjectDefinition>`
Element **ACTIVITY**		`<ControlFlowObjectDefinition controlflow-object-type="ACTIVITY" controlflow-object-definition-id="F2" controlflow-object-subtype="task" artifact-idlist="..." lane-idlist="..."`

		`linked-process-id="...">`
		`</ControlFlowObjectDefinition>`

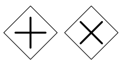
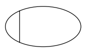
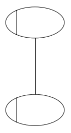
Element **GATEWAY** Subtypes *AND, XOR,*		`<ControlFlowObjectDefinition`
		`controlflow-object-type="GATEWAY"`
		`controlflow-object-definition-id="AND1"`
		`controlflow-object-subtype="AND|XOR"`
		`artifact-idlist="..."`
		`artifact-dirlist="..."`
		`lane-idlist="..."`
		`linked-process-id="...">`
		`</ControlFlowObjectDefinition>`

Table 1: XML-Representation of the Input BMPN Elements for the Process View

Equivalent to the processes which are modeled within ARIS-Toolset and exported into the BPMN-XML explained above, a proprietary OMN-XML is used to represent the organization model of ARIS-Toolset. The elements for representing the organization are the organization unit and three relations between them: the hierarchy, the delivery and the contract relation. The hierarchy relation would be sufficient as long as the organization units within one organization have to be represented. However, the relation between whole organizations can also be organized as a hierarchy or alternatively as a network. This was the idea behind adding contract and delivery relations to the OMN-XML in order to be able to represent network organization. With these two additional forms of relations we can indicate where the execution of a business process is passed from one organization to another.

OMN	Graphical Representation	OMN-XML-Representation

Organization unit		`<OMN-ObjectDefinition id="OE1"`
		`type="ORGUNIT"`
		`subtype="CONSORTIUM">`
		`</OMN-ObjectDefinition>`

Hierarchy between organizations		`<OMN-EdgeDefinition id="E1"`
		`type="ORG"`
		`source-object-definition-id="OE1"`
		`target-object-definition-id="OE2">`
		`</OMN-EdgeDefinition>`

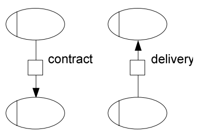
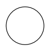
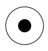
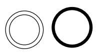
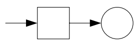
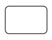
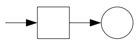
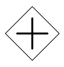
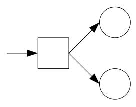
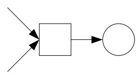
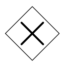
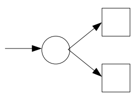
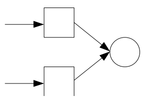
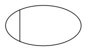
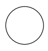
Contract resp. delivery relation between organizations		`<OMN-EdgeDefinition id="E2"`
		`type="ORG"`
		`subtype="CONTRACT|DELIVERY"`
		`source-object-definition-id="OE1"`
		`target-object-definition-id="OE2">`
		`</OMN-EdgeDefinition>`

Table 2: XML-Representation of the Input OMN Elements for the Organization View

4.3 Transformation Rules

Table 3 and 4 intend to illustrate the transformation concept which is based on the semantics of the elements. Since a graphical representation of the transformation is easier to read then XML-code, we present the transformation rules in table 3 graphically. The XML-representation of the input elements can be derived easily by table 1 and 2. For output elements see table 4 where the PNML representation of each Petri net element is demonstrated. Table 4 also includes the characteristic of Reference nets of adding uplinks and downlinks to transitions in order to enable them to fire synchronously.

BPMN-XML		**PNML**	
Element **EVENT**			
Subtype *START*			
Subtypes *INTERMEDIATE, END*			
Element **ACTIVITY**			
Element **GATEWAY**			
Subtype *AND*		AND-Split	
		AND-Join	
Subtype *XOR*		XOR-Split	
		XOR-Join	
OMN-XML		**PNML**	
Organization			

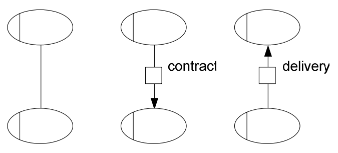
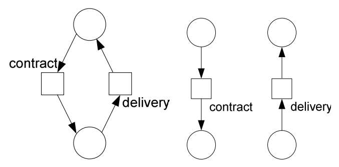
Relations between organizations		

Table 3: Converting Rules for the Process and Organization Perspective

Petri Net Element	Graphical Representation	PNML-Representation

Unmarked place	○	`<place id="...">`
		`  <text>...</text>`
		`</place>`

Marked place	◉	`<place id="...">`
		`  <initialMarking>`
		`    <text>...</text>`
		`  </initialMarking>`
		`</place>`

Transition	□	`<transition id="...">`
		`  <name>`
		`    <text>...</text>`
		`  </name>`
		`</transition>`

Arc	→	`<arc id="E1_F1" source="E1" target="F1" />`
Refence Net Chracteristic	**Graphical Representation**	**PNML-Representation**

Uplink	□
	uplink-name
		`<transition id="...">`
		`  <name>`
		`    <text>...</text>`
		`  </name>`
		`  <uplink>`
		`    <text>...</text>`
		`  </uplink>`
		`</transition>`

Downlink	□
	downlink-name
		`<transition id="...">`
		`  <name>`
		`    <text>...</text>`
		`  </name>`
		`  <downlink>`
		`    <text>...</text>`
		`  </downlink>`
		`</transition>`

Table 4: PNML-Representation of Petri Net Elements

5 Showcase

5.1 Visualizing Level

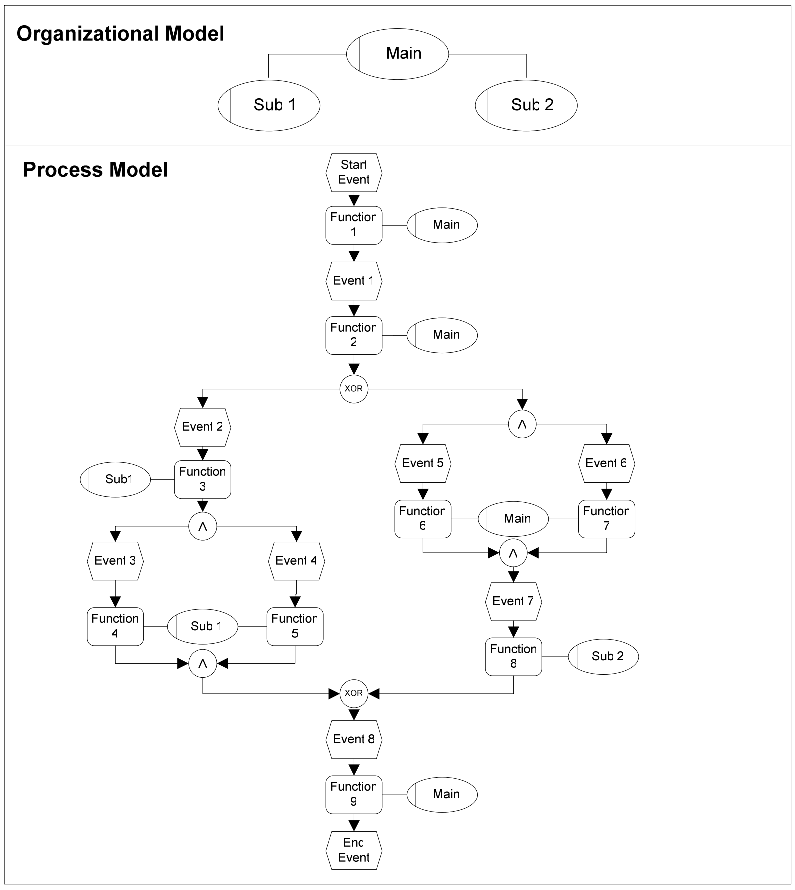


Figure 3: Example for Visualization of an Inter-Organizational Process and Organization Model

To demonstrate our concept we designed a business process concerning three companies, which are hierarchically organized. The organization model of the collaboration is depicted in the upper part of figure 3. The process is depicted as an EPC (cf. lower part of figure 3) and describes an abstract process. In this example a company *Main* coordinates the processes executed by *Sub 1* and *Sub 2*. After a Start Event, *Function 1* and *Function 2* are executed by *Main*. Afterwards, the process splits into two alternative parts: either *Function 3* is performed by *Sub1* or *Function 6* and *Function 7* are concurrently executed by *Main*. After *Function 3* in the left part, *Sub 1* performs also *Function 4* and *Function 5* concurrently. In the right path *Function 6* and *Function 7* are synchronized. Afterwards *Function 8* is performed by *Sub 2*. In any case, the process ends with *Function 9* performed by *Main*, and the *End Event*.

After visualizing, the process model as well as the organization model are exported into BPMN-XML and OMN-XML and stored in the repository. To implement the transformation rules introduced above, we have developed a JAVA-based converter which uses these files as input and creates two interrelated output files.

5.2 Interrelating the two PNML output files

The interrelation between the two PNML files is based on uplinks and downlinks indicating which transitions of the two models have to fire synchronously within object and system net. Hence, we first have to adopt a convention for naming the labels of such synchronous channels connecting the states of the system net and use them appropriately for the object net. This is illustrated by an example in figure 4. *Main*, *Sub1* and *Sub2* are connected by transition. Each transition of the system net has a downlink. The naming convention is that the label illustrates to which organization unit the transition transports the process when it is fired. For instance the label of the transition passing a process from *Main* to *Sub*" is *x:SUB_1()* (the syntax is based on the RENEW-Tool). In order to interlink the dynamic behavior of process and organization models we have to add these labels to appropriate transitions of the object net. To detect where an interlink is needed, every two successive functions of the process model represented by the object net have to be compared on their organization assignments. If they differ, an uplink has to be assigned to the event lying in between the two functions. This has lead to add the uplink *:SUB1()* to the *Event 2* which is located between *function 2* and *function 3* of the output PNML file representing the object net as shown in table 5.

Object Net (Process View)	System Net (Organizational View)
`<pnml>`	`<pnml>`
`  <net type="RefNet" id="1">`	`  <net   type="RefNet"   id="1">`
`    <name>`	`    <name>`
`      <text>objectnet</text>`	`      <text>systemnet</text>`
`    </name>`	`    </name>`
`    […]`	`    […]`
`    <transition id="Event 2">`	`    <transition  id="MAIN_SUB1">`
`      <name>`	`      <name>`
`        <text>Event 2</text>`	`        <text>contract</text>`
`      </name>`	`      </name>`
`      <uplink>`	**`      <downlink>`**
`        <text>:SUB1()</text>`	**`        <text>x:SUB1()</text>`**
`      </uplink>`	**`      </downlink>`**
`    </transition>`	`    </transition>`
`    […]`	`    […]`
`  </net>`	`  </net>`
`</pnml>`	`</pnml>`

Table 5: Interrelating the Output PNML Files

5.3 Graphical Representation of the output PNML files

Figure 4 depicts the graphical representation of both PNML files after importing and manually adjusting their representation in the RENEW-Tool.

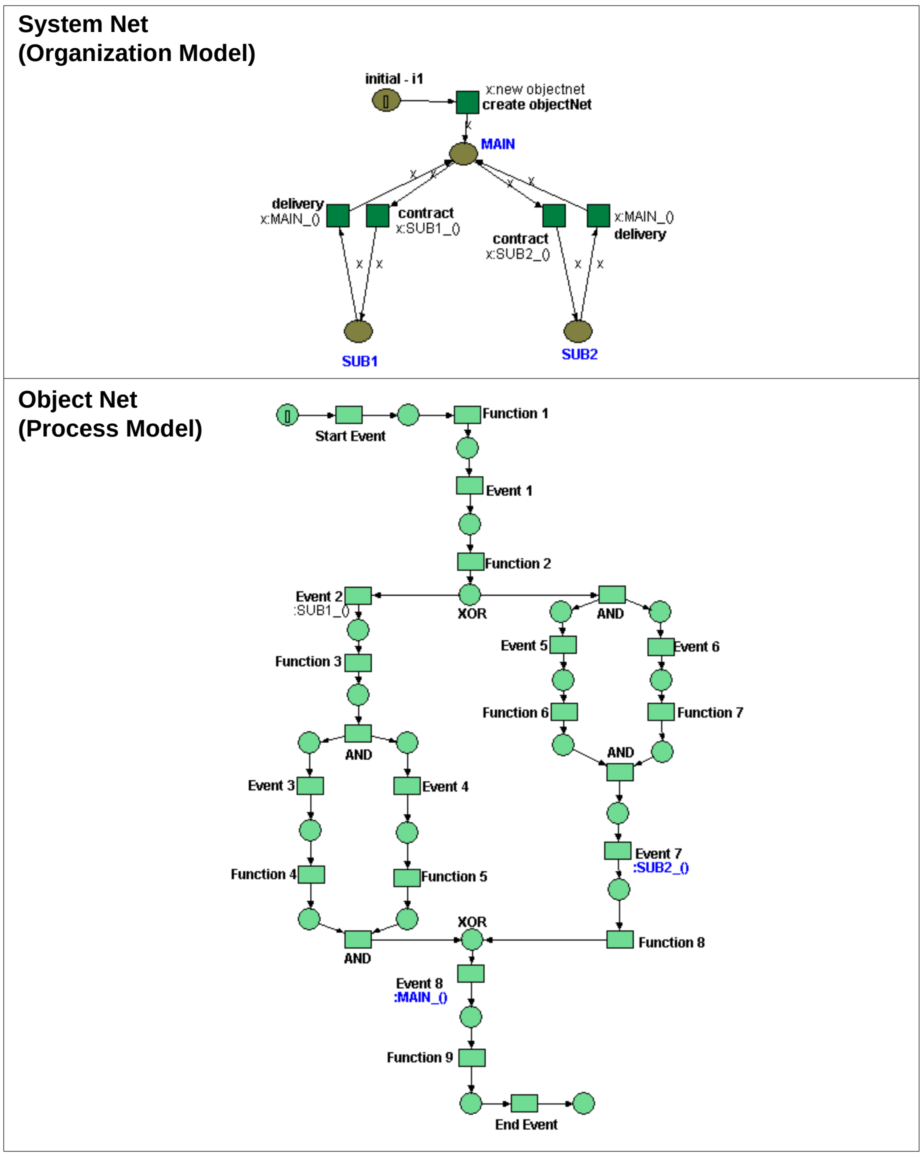


Figure 4: Transformation of the Models in Figure 3 into an Object Petri Net

The upper part represents the system net, which stands for the organization view. The lower part represents the process view. Relations between organization units are modeled by delivery and contract transitions. After instantiating the process model from the organization model with *x:new objectnet*, the model can be simulated. A validation has the potential to prove the resulting models concerning their correctness, e. g. interdependencies between organization units or established communications. The structure and behavior of the Object Petri net can be verified, so e. g. if the net is terminated correctly or deadlocks are reached. Different performance indicators such as process times, process costs, or the number of produced output can be ascertained. These can be used to detect bottle necks, deadlocks, or livelocks, so finally process and organizational models can be changed without having trouble in runtime processes.

The step-by-step simulation of the RENEW-Tool allows the tracking how in each organization the process is partly executed and then passed to the next organization. However, currently only one instance exists at a time and can be passed. Hence no process concurrencies involving multiple organizations can be realized with this design. The reason for this is that currently only one copy of the process instance is available. Multiple copies of the process instance would lead to the possibility to execute concurrent process tasks spitted by an AND-connector by different organizations. Additional limitation is based on the Reference net formalism deployed which is restricted to have one individual object net only in order to avoid inconsistent execution. To model the execution of different parts of the process within multiple organizations independently, we need to have multiple individual copies of the same process. This would cause to apply the value semantic of object Petri nets and the p-marking which is not possible with the RENEW-Tool.

6 Conclusion

This paper presented the concept of the transforming BPMN-conform XML for process models and a proprietary OMN-XML representing organization models into two interrelated PNML files representing a Reference net which is a special type of object Petri nets. The output PNML files can be imported into the RENEW-Tool and simulated considering control-flow as well as resource perspective of a business process.

Based on the introduced concept we have implemented a JAVA based converter using the BPMN-XML and OMN-XML and creating according PNML files. Currently the concept and our implementation are at a prototype stage. While it can demonstrate the general concept of interrelating process and organization models to a singe formal notation, there is still lot more to do. Firstly we need to broaden the concept of the system net to have multiple copies of one process instance. This would lead to model concurrent execution of tasks derived from an AND-split. Secondly, we need to add further BPMN elements including the OR-connector. Since Petri nets allow analyzing processes, it would be helpful to deliver processing time, resource availability, path probability and market demand information with the BPMN input processes. The converter could use this information to generate timed Petri nets which can be used for performance analysis and capacity planning or elementary Petri nets to apply common verification techniques. However, it has to be determined how these analysis techniques can be applied to the object Petri net approach.

Acknowledgement

The research project ArKoS – Architecture of Collaborative Scenarios – has been funded by the German Federal Ministry of Education and Research. The authors gratefully acknowledge the support of Ingo Taraske und Dariusz Kleczek with the implementation of the converter.

References

[Aa02] van der Aalst, W. M. P.: Making Work Flow: On the Application of Petri Nets to Business Process Management. In: J. Esparza; C. Lakos (Eds.): Applications and Theory of Petri Nets 2002: Proc. of 23rd International Conference (ICATPN). Bd. 2360, Adelaide, Australia 2002, pp. 1-22.

[Aa98] van der Aalst, W. M. P.: The Application of Petri Nets to Workflow Management. In: The Journal of Circuits, Systems and Computers 8 (1998) 1, pp. 21-66.

[Aa99] van der Aalst, W. M. P.; van der Moldt, D.; Valk, R.; Wienberg, F.: Enacting Interorganizational Workflow Using Nets in Nets. In: J. Becker; M. zur Muehlen; M. Rosemann (Eds.): Workflow Management '99. Münster, Germany 1999, pp. 117-136.

[ADK02] van der Aalst, W. M. P.; Desel, J.; Kindler, E.: On the semantics of EPCs: A vicious circle. In: F. J. Rump (Eds.): EPK 2002 - Geschäftsprozessmanagement mit Ereignisgesteuerten Prozessketten. Trier, Germany 2002, pp. 71-79.

[AH02] van der Aalst, W. M. P.; van Hee, K. M.: Workflow Management : Models, Methods, and Systems. Cambridge 2002.

[AHH94] van der Aalst, W. M. P.; van Hee, K. M.; Houben, G. J.: Modelling and analysing workflow using a Petri-net based approach. In: G. D. Michelis; C. Ellis; G. Memmi (Eds.): 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms. Zaragoza, Spain 1994, pp. 31-50.

[AKV03] van der Aalst, W. M. P.; Kumar, A.; Verbeek, H. M. W.: Organizational Modeling in UML and XML in the context of Workflow Systems. In: J. Carroll (Eds.): Symposium on Applied Computing (SAC) 2003. Melbourne, Florida, USA 2003, pp. 603-608.

[Bi03] Billington, J.; Christensen, S.; van Hee, K. M.; Kindler, E.; Kummer, O.; Petrucci, L.; Post, R.; Stehno, C.; Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: E. Best (Eds.): Applications and Theory of Petri Nets 2003, 24th International Conference, ICATPN 2003. Bd. 2679, Eindhoven, The Netherlands 2003, pp. 483-505.

[Co38] Coase, R. H.: The Nature of Firm. In: Economica 4 (1938), pp. 386-405.

[CP77] Caves, R. E.; Porter, M. E.: From entry barriers to mobility barriers: conjectural decisions and contrived deterrence to new competition. In: Quarterly Journal of Economics 91 (1977) 2, pp. 241-261.

[Da01] Davis, R.: Business process modelling with ARIS : a practical guide. London et al. 2001.

[EM00] Evaristo, R.; Munkvold, B. E.: Collaborative Infrastructure Formation in Virtual Projects. Sixth Americas Conference on Information Systems (AMCIS). Long Beach, California, USA 2000, pp. 1705-1710.

[GV03] Girault, C.; Valk, R.: Petri Nets for System Engineering : A Guide to Modeling, Verification and Applications. Berlin et al. 2003.

[HP90] Hamel, G.; Prahalad, C. K.: The Core Competence of the Corporation. In: Harvard Business Review (1990) May 1991, pp. 79-91.

[JV87] Jessen, E.; Valk, R.: Rechensysteme: Grundlagen der Modellbildung. Berlin et al. 1987.

[Ku02] Kummer, O.: Referenznetze. Berlin 2002.

[Ku04] Kummer, O.; Wienberg, F.; Duvigneau, M.; Schumacher, J.; Köhler, M.; Moldt, D.; Rölke, H.; Valk, R.: An extensible editor and simulation engine for Petri nets: Renew. In: W. Reisig (Eds.): 5th International Conference on Application and Theory of Petri Nets (ICATPN 2004). Bd. 3099, Bologna, Italy 2004, pp. 484-493.

[Mu04] zur Muehlen, M.: Organizational Management in Workflow Applications - Issues and Directions. In: Information Technology and Management 5 (2004) 4.

[Pe62] Petri, C. A.: Kommunikation mit Automaten. Bonn 1962.

[RKC98] Rittenbruch, M.; Kahler, H.; Cremers, A. B.: Supporting Cooperation in a Virtual Organization. In: R. Hirschheim; M. Newman; J. I. DeGross (Eds.): Nineteenth International Conference on Information Systems (ICIS). Helsinki, Finland 1998, pp. 30-38.

[RR98] Reisig, W.; Rozenberg, G.: Lectures on Petri Nets I: Basic Models. Bd. 1491, Berlin et al 1998.

[Th05a] Theling, T.; Zwicker, J.; Loos, P.; Adam, O.; Hofer, A.: Enabling Dynamic Networks using an Architecture for Collaborative Scenarios. In: R. J. Scherer; P. Katranuschkov; S.-E. Schapke (Eds.): CIB W78 - 22nd Conference on Information Technology in Construction. Bd. 304, Dresden, Germany 2005, pp. 83-90.

[Th05b] Theling, T.; Zwicker, J.; Loos, P.; Vanderhaeghen, D.: An Architecture for Collaborative Scenarios applying a common BPMN-Repository. In: L. Kutvonen; N. Alonistioti (Eds.): Distributed Applications and Interoperable Systems: 5th IFIP WG 6.1 International Conference (DAIS). Bd. 3543, Athen, Greece 2005, pp. 169-180.

[Va04] Valk, R.: Object Petri Nets - Using the Nets-within-Nets Paradigm. In: G. Rozenberg (Eds.): Lectures on Concurrency and Petri Nets: Advances in Petri Nets. Bd. 3098, Berlin et al. 2004, pp. 819-848.

[Va98] Valk, R.: Petri Nets as Token Objects - An Introduction to Elementary Object Nets. In: J. Desel; M. Silva (Eds.): 19th Int. Conf. on Application and Theory of Petri Nets, ICATPN'98. Bd. 1420, Lisbon, Portugal 1998, pp. 1-25.

[Wh04] White, S. A.: Introduction to BPMN. http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf, Abruf 2005-11-17.

[Wi95] Williamson, O. E.: Organization Theory: From Chester Barnard to the Present and Beyond. Oxford 1995.

[Zi77] Zisman, M. D.: Representation, Specification and Automation of Office Procedures. Philadelphia, Pennsylvania 1977.

Wrapping Legacy Software for Reuse in a SOA

Harry M. Sneed
AneCon GmbH, Wien
E-mail: Harry.Sneed@anecon.com

Abstract: Legacy programs, i. e. programs which have been developed with an outdated technology make-up for the vast majority of programs in many user application environments. It is these programs which actually run the information systems of the business world. Moving to a new technology such as service oriented architecture is impossible without taking these programs along. This contribution presents a tool supported method for achieving that goal. Legacy code is wrapped behind an XML shell which allows individual functions within the programs, to be offered as web services to any external user. By means of this wrapping technology, a significant part of the company software assets can be preserved within the framework of a service oriented architecture.

Keywords: Service Oriented Architecture, legacy software, system integration, wrapping, web services, XML, WSDL

1 Legacy Software

Legacy programs can be divided into three basic categories in regard to the degree of dependence on their environment.

- programs which are not dependent on their environment,
- programs which are partially dependent on their environment,
- programs which are totally dependent on their environment [1]

The first category includes programs written in the conventional languages Fortran, COBOL, and C/C++. These programs can be readily reused in any environment which has a compiler to compile them. The second category encompasses programs written in a language which uses run time or link time functions. To this category belong PL/I, Smalltalk, and Forté. These programs can be reused in another environment, but only if their runtime routines are substituted by compilable modules written in the host language itself.

The third category consists of all of the 4th generation language programs, such as ADS-Online, Natural, CSP and Oracle Frames, requiring a specific environment to run in. Such software can not be reused in another environment. It is environment dependent. Therefore, the only way to reuse these programs is to keep them in their native environment and to build runtime links to that environment.

One can summarize from this observation, that the more elementary a programming language is, i.e. the less bells and whistles it has, the easier it is to reuse. This is something that managers should consider when choosing a development technology. They must choose between short range productivity and long range reusability and portability. [2]

2 Service oriented architecture

The main goal of a service oriented architecture is to make the software functionality available to all who need it and who are authorized to use it. Not only that, they should also be able to combine the functionality in any way they deem appropriate, i.e. to embed it as steps in their business processes. By invoking the methods offered by the service architecture they can fulfill the functions referred to within their business process language – BPEL – procedures without having to code and test them themselves. The price for that is modeling the evolution of legacy systems to the WSDL interface, setting the parameters according to the interface specification. [3]

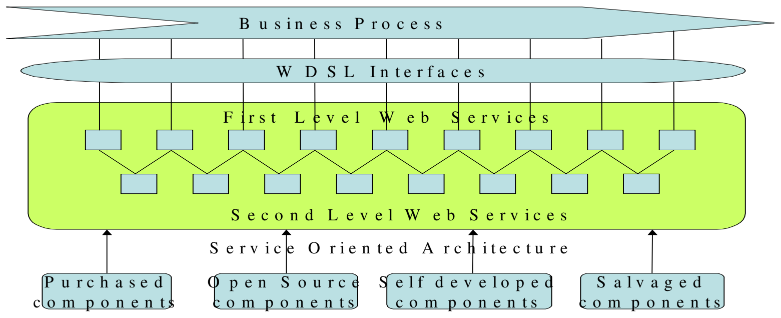


Figure 1: Sources of Web Services

Seen from this perspective, the service oriented architecture is similar to a giant subroutine library, with the difference that the user must not copy it onto his computer, compile it and link it with his own programs. He can access it at execution time via the internet. This way he is assured of always using the latest versions and does not have to worry about continual updates. The price he has to pay is twofold.

First, there is a performance price. Sending subroutine calls with long parameter lists across the networks requires time. The more services that are invoked and the more parameters passed, the longer will be the transmission times. An XSLT-based transformation as a single WSDL interface can take up to 500 milliseconds. Therefore, it is advisable to minimize both the number of calls and the number of parameters. [4]

The second price is that of complexity, resulting in more effort to use the service. The greater the functionality of the individual web service, the more complex is the interface to it. Not only will the service method require more input parameters, it will also produces more results, all of which have to be specified in the interface. The web service call becomes increasingly complicated, with layers of nested data and parameter lists until it becomes more and more like a program itself. At some point, one must ask if it were not simpler to code the service oneself rather than spending so much time and effort to build up a WSDL interface. On top of that, complex interfaces require many test cases to validate and are error prone. It could well be, that it requires more effort to test the web service interface than it does to program the service oneself. [5]

The conclusion here is that web services must remain simple in order to be readily usable and to be built into the user's business processes with a minimum of effort and a maximum of performance. In this respect service oriented architectures are similar to other technological solutions of the past. They can reduce the development and maintenance effort of the user by offering ready made software functionality, but they extract a price in comprehending and testing as well as in performance. The goal of the SOA designer should be to keep these costs as low as possible by offering a large number of simple elementary services with interfaces that are easy to serve and to test. The designer should strive to minimize the number of input parameters and to return no more results than necessary for any one invocation. In other words, the web services should be limited and their interfaces as narrow as possible. [6]

Having set these design goals the question then comes up as to where the web services come from, i.e. how are they supplied. As with all standard components there are four basic sources:

- they can be bought from a web service vendor,
- they can be borrowed from the open source community,
- they can be developed individually or
- they can be taken from existing applications.

There are several vendors which now offer off the shelf web services including the major software producers Microsoft, IBM, SUN and SAP. [7] For a user company bent on building up a service oriented architecture it is advisable to consult the catalogues of these vendors and to purchase those services which fit their requirements. Of course, the user then becomes dependent on the vendor to maintain the services purchased, but this has always been the price of standard solutions.

The same applies to the open source community. Here too scores of individual software services are available and the number is constantly increasing. The draw back here is that the user must maintain the services himself, i.e., he is dependent on the ability of his own programming staff to comprehend the foreign code and to adjust it to his local needs. That requires knowledge, time and tools. The comprehension problem with web services is no less than with any other foreign software components. [8]

Developing the web services oneself is always an alternative. Large user organizations can set up a special development group to produce such common services, just as was the case with the common subroutine libraries and the common class libraries. There is no real difference here, only the interface languages changes. Instead of processing parameter lists or linkage sections, the developers now have to deal with WSDL schemas. Besides developing the services, the user also has too test them. This could be an obstacle to many users who are not versed in testing technology. Testing a WSDL interface is more demanding than testing a GUI. The GUI can be created and validated visually at test time. The WSDL is basically invisible. The interface has to be generated by a program and sent to the target service via a middleware product. The results have to be received by a program, recorded and validated against the expected results. All of this requires tools and experienced testers, something most user organizations do not have. [9]

The fourth and final source of web services is the existing software. Every user organization which has been using information technology for any length of time will have accumulated a significant amount of legacy software. Some of this software will be tightly coupled to the environment for which it was developed, in particular the presentation software which is presenting maps or GUIs. Other parts of the software will be tightly coupled with a particular database system, namely the data access software. In so far as the same database is used for web applications, this software can be reused. A third and significant part of the application software will be devoted to processing the business logic, which have to be mined out of the existing code. [10]

Locating and salvaging such business-oriented software is referred to as code mining or software recycling. It is similar to salvaging valuable building blocks from the ruins of an old building in order to reuse them in a new edifice. The technology for doing this has been available since the mid 1990's and has been well covered in the reengineering literature. [11] What is new here, is the attempt to reuse these old code blocks as web services in a service oriented architecture. The technology for doing that is the subject of this paper.

The advantage of reusing one's own code as opposed to the other sources of web services is obvious. Using off the shelf web services is inexpensive, but such services will seldom fulfill the exact requirements of any particular user organization. At best they can be used to supplement the user's own unique services. Besides, since they do not belong to the user, the user is dependent upon the supplier to maintain and evolve them. Developing new web services from scratch is an enticing alternative, especially for developers eager to experiment with the new technology, but one always underestimates the effort required to test new services and to bring them up to a quality standard where they can be relied upon. [12]

The costs of developing high quality web services are for many users simply too high. Even large organizations cannot or will not afford it. So, developing one's own web services is a long range goal which can be achieved in the course of many years, but it is not something that can be achieved within a short run. That leaves the user with a choice of either retrofitting his business processes to accommodate the standard web services available or reusing his existing software which was built from the beginning to fit his particular business processes.

3 Candidates for web services

According to the marketing director of the Software A.G., the core functionality of most public administration offices is buried deep in their existing application software. [13] It is futile to attempt to reproduce it in another form. The only practical solution is to wrap it and make it available as a universal public service. What is not mentioned here is that this functionality must first be salvaged and brought up from the depth in order to reuse it. E-government has become a prime candidate for the techniques of software recycling.

The same applies to the functionality in business administration. There are scores of individual company specific tasks unique to every enterprise. Typical examples are the modes of payment, the granting of credit, the computation of interest rates and the handling of privileged customers. Conventional business processes are full of such user specific solutions which have been evolved and tuned over many years. They are an essential part of the company operation.

The problem is that this customized business logic is not readily accessible. It must be identified through various mining techniques as pointed out by Aversano and Tortorella in their work on salvaging public administration systems for eGovernment applications. [14] One is fortunate to even find a particular business function in a single module. In a bank application reengineered by the author the opening of an account was scattered across five different components, thus violating the principle of locality of reference. None the less, the functionality was present and distinguishable from the other functions around it, even though they shared some common code.

In reusing existing code, the first task is to identify the candidates for a web service. User organizations wanting to move to a service oriented architecture must make a portfolio analysis of their existing applications and to list out the essential application operations. In doing so, it will be necessary to break the complex operations down into elementary operations which are self contained logical units. In an order entry application the basic operations might be

- confirming the credibility of the customer,
- reducing the stock,
- billing the customer and
- handling back orders.

The elementary business operations such as reducing the stock can be reused directly in another context. The billing of the customer is, however, a too complex operation, which has to be further broken down into

- aggregating the billing items
- computing the sales tax
- obtaining the customer address data
- producing the bill
- dispatching the bill.

These elementary operations are candidates for web services. They have a limited number of arguments, i.e. input variables and a single compound result. As such they can be fitted conveniently into any business process.

The second step is to assess the business value of these reuse candidates. Ben-Menachem suggests a classification scheme, which involves categorizing the items, calculating each item's value and assigning a value coefficient. Existing software components can be categorized by language, purpose, type and criticality. Calculating an item's value is based on cost analysis of the development costs, the maintenance costs, the estimated replacement costs and the annual business value contributed by that item. In assigning a value coefficient, the business value over a three year period minus the maintenance costs is divided by the costs of replacement, i.e redevelopment of that item.

Business_Value – Maint_Costs

Cost of Replacement

The reusable code items are then ranked based on their value coefficient. The ranking shows which business operations have the highest potential as web services. [15]

The key to defining suitable web services is the granularity of the services. They must be broken down to a level of granularity where each service performs a single well defined transformation or computation upon a limited set of parameters to provide a singular result. Furthermore, they should be stateless. A web service should not be required to maintain it's own state. If a web service is invoked a second time, the user cannot expect for it to remember what the result of the last invocation was. The preservation of persistent objects such as the article data in the order entry example is the responsibility of the overlying business process. Prior to invoking the web service "Stock – reduction", the article data would have to be retrieved from the article data base and afterwards restored in the altered state.

It is true that the business processes will become over burdened with the many web service invocations, but this way it will not be necessary to constantly change the web services. One has to choose here between two evils. Either the control logic is included in the web services or it is contained within the business process. The web services should remain as constant as possible. All changes should be made at the business process level, by changing the order of web service invocation, by altering the parameters or by invoking additional services.

The essence of a successful service oriented architecture is according to Prof. Scheer, the father of the ARIS business process modelling system, flexibility. [16] The architecture must be adaptable to changes in the business environment with a minimum of effort and time. This goal can only be achieved if the underlying services are kept at a low level of complexity. The complexity should be built into the business processes where it can be more readily managed.

4 Creating web services from legacy code

There are three basic steps required to create web services from legacy code.

- salvaging the legacy code
- wrapping the salvaged code and
- making the code available as a web service.

These three steps will be described in the following sections.

4.1 Salvaging the legacy code

To be able to salvage code from an existing legacy code base it is first necessary to locate that code and to determine if it is worth reusing. It is not a problem to analyze and evaluate the code of a few small programs. That can be done by any programmer familiar with the code using a comfortable text editor. It is quite different to analyze several hundred large programs in search of a few reusable blocks of code. Here too a domain expert is required, but he must be supported by automated reverse engineering tools.

The key to discovering the business operations are the results which they produce. By identifying the variables which are returned by the functions processing the business operations one can also identify the functions. If the programs were structured in such a way that the business functions were assigned to one code block such as a function in C, an internal procedure in PL/I, a subroutine in Natural, or a paragraph in COBOL, then this task would be simple, but they seldom are. More likely a business function is scattered throughout several blocks of code in several modules. On the other hand, one block of code may be processing several business functions. So there is a n:m relationship between code blocks and business operations.

By making a data flow analysis based on the final results, it is possible to trace the result back through all of the statements which contributed toward producing it. Once the statements are identified, then it is possible to locate in what code units, i.e. procedures, paragraphs, subroutines, etc., they are in. Only those units are then copied from the original source together with the variables they refer to. This technique is known as “Code stripping”. It was originally used in testing to verify the path leading to a given output. However, it applies equally well to the task of extracting elementary business operations. [17]

The essential point here is that a business operation is defined as an algorithm for computing a given result. This result may be a yes or no answer for instance to determine whether a customer is a VOP customer or not. There may also be several results produced in different places, for instance an order entry process which not only confirms the fulfillment of that order, but also updates the amount of the item ordered and generates a billing position and a dispatch order. To extract the code for processing an order, it would be necessary to identify all of the data objects affected as a result of that processing.

The next step after identifying the code of a business operation, is to extract that code and to reassemble it as a separate module with its own interface. This is done by copying the impacted code units into a common framework and by placing all of the data objects they refer to into a common data interface. In C the interfaces are parameters of the type structure, in Cobol the objects are level 1 items in the linkage section, in PL/I the objects are based data structures with the pointers to them as parameters to the main procedure. The end result will be, in all cases, a subroutine with a call interface. The original input arguments will be input parameters and the original output arguments output parameters. In this respect the business logic code will have been disconnected from the original user interface and made into a self contained subprogram. This is a prerequisite to wrapping it. [18]

A useful byproduct of this code reengineering process is a documentation of the existing business operations. For each data result of a particular use case, the conditions, assignments, computations and IO operations required to produce that outcome are presented in the form of a data flow tree. The final result state builds the root node of the tree. The other nodes are the arguments and intermediate variables which flow into that final result. The branches of the tree represent the state transitions, which are triggered by conditional statements such as if, case- and loop statements. Since a business operation is an intersection of control and data flow, it is necessary to depict both perspectives.

With the aide of these diagrams, it becomes possible for the user to decide whether an existing operation, implemented within a legacy system is worth reusing as a public function in a service oriented architecture. The decision requires a full comprehension of the current operation as well as a notion of its economic value. If an operation has a high economic value and a low level of implementation, it may be better to rewrite it again as a separate entity. Operations with an acceptable implementation and a medium to high economic value are the prime candidates for reuse.

4.2. Wrapping the legacy code

Once a business operation has been located, documented and found worthy of reuse, the next step it to wrap it. The goal of the wrapping process is to provide the component extracted from the legacy code with a WSDL interface. The technique used is to transform each entry into a method and to transform each parameter into an XML data element. The data structures will become complex elements with one or more sub-elements. The methods will have their arguments and results as references to the data element descriptions. Both the methods and the parameters will be built into an XML schema.

(see Sample 1: Input Interface Schema)

The tool SoftWrap has been developed to automate this transformation for the languages PL/I, COBOL, and C/C++. Besides, creating the WSDL interface description, it also enhances the wrapped component with two additional modules. One module is for parsing the incoming message and extracting the data from it. The extracted values are then assigned to the corresponding arguments in the wrapped component. The other module is for creating the return message from the results produced by the wrapped component. In this way an elementary operation can be reused as a web service without having to change the code. The two generated subroutines act as a bridge between the WSDL interface and the call interface of the original code.

(see Sample 2: Output Interface Schema)

In PL/I these two subroutines are implemented as external procedures, in COBOL as subprograms and in CPP as separate classes. The purpose is to avoid manual manipulation of the legacy code, since manual intervention is not only costly, but also error prone. To be effective wrapping must be automated. This simple fact has been acknowledged by the major EAI vendors, who offer wrapping solutions for entire programs and databases. [19] The approach proposed here differs from these other commercial solutions, in that it deals not with the original online transactions, screens and programs, but with artificially constructed segments of code extracted from the original programs for the sole purpose of being used as web services.

The motivation of the EAI vendors is to enable the user to link together diverse applications via a hub software. The goal of a service oriented architecture is to replace the existing applications altogether by a series of fine grained components which the user can assemble into dynamic applications on demand. This difference in strategy makes it impossible to reuse the old programs as they are. Nevertheless, the logic they contain can be reused, but only if it is extracted from the original contact and transformed into another web compatible one.

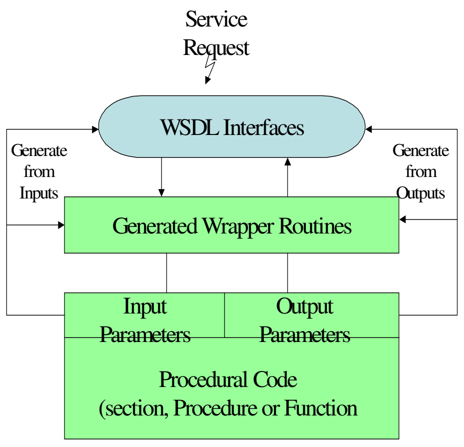


Figure 3: Wrapping Salvaged Components

4.3 Linking the web services to the business processes

The third and final step in creating web services from legacy code is to link the web services to the overlying business processes. This is made by means of a proxy component. The business process actually invokes the proxy which is available in the same address space as the process definition. Like in CORBA the proxy checks the parameters and generates the WSDL interface which is then dispatched by some message service such as MQ-Series to the application server. This proxy technique has been referred to by Aversano and Canfora in a previous paper discussing the wrapping of legacy application for web access. [20]

On the application server there is a scheduler, which receives the incoming message, determines which web-service is to be performed and forwards the WSDL contents to that particular service, in our case the wrapped legacy code. The wrapper of the code parses the XML input data and moves the values to the appropriate addresses in the wrapped component.
Once the wrapped component has been executed, it's result is transformed by the wrapper into an XML output data structure, which goes back to the scheduler to be transmitted back to the web client. In this way the business process can be executed on any client anywhere and still is able to access the legacy functions on the original application server. [21]

5 Case Study of a legacy web service

The example selected to demonstrate the integration of legacy components into a service oriented architecture is that of a calendar function extracted from the legacy software of a Swiss bank. The function accepts a date, a language code and an adjustment parameter. It returns the day of the week in the language indicated by the language code, adjusted either to the left or the right of the text line depending on the text adjustment parameter. This function was originally implemented in Assembler and was later converted to COBOL within the scope of a major migration project. Later it was decided to reuse it as a service in a web architecture.

5.1 Extracting Business Operations

Within the COBOL code, the function was a separate section, so the first step in wrapping it was to extract it from the source text of the program and to place it in a separately compilable module with its own data division and linkage section. The result of the business operation for computing the name of the week-day-name , the variable referred to as DAY-NAME, was used to locate the code that computed it. The first reference to DAY-NAME was found in an initialization paragraph where it was set to spaces, the second reference in the leap year processing paragraph where it was set to a default value and the third reference in a paragraph which prepared the output.

A trace of the input arguments showed that they were only referenced in these three sections of code. Therefore, there three paragraphs were cut out of the original procedure division and placed in a new procedure. The variables they used – the arguments, the result, the intermediate variables and the constants – for instance the table of weekdays in the three Swiss languages were scattered throughout the original Data Division. They were collected together to form a new Working-Storage section. The three input parameters and the one output parameter were placed in the Linkage-Section. The result of this reengineering process was a new COBOL module with its own Data-Division, working-storage, linkage-section and procedural code.

The procedural code consisted of the three paragraphs extracted from the original program – initialization, leap year handling and output setting. All of these reengineering steps are performed automatically by the SoftWrap tool. The user need only identify the input and output variables of the operations he wants to extract. The tool SoftWrap has been described in previous reports. [22]

5.2 Wrapping Business Operations

The second step, also performed by SoftWrap, is to wrap the new module extracted from the old code. This entailed generating a driver module which reads the input parameters – date, language and adjustment – from a WSDL input file and writes the result – the day of the week or error message – into a WSDL output file. In addition, SoftWrap also produces an XML schema both for the input and the output file.

The schema describes the structure and the attributes of the parameters. Besides the usual XML attributes such as name, type and occurrence, each data element has some additional attributes necessary to convert the XML data types into COBOL data types and to set them into or to receive them from the corresponding COBOL address. For example, it is necessary to know that the DAY-NAME is at the 10th position in the parameter string, that it is 10 bytes long and that it is a character field.

The generated driver module parses the schema in order to interpret the incoming WSDL message with the date, the language code and the adjustment code, and to create an outgoing WSDL message with the day of the week or an error message.

In this way, the calendar function has been wrapped. It can be invoked via a WSDL interface with the three input parameters and the name of the method to be executed. When the calendar method has been executed, it will then return the result to the business process requesting it.

5.3 Integrating Business Operations

It remains now to implement the business operation by invoking it from a business process. The language for implementing business processes is BPEL4WS. BPEL4WS establishes links to partners, defines the link types, declares the parameters to be sent and the results to be received, and invokes the web services. [23] The sample process script sets the parameters for the date, language and adjustment and then identifies the service by name as depicted in Sample 3. (see Sample 3: User Business Process)

A WSDL interface is generated by the BPEL interpreter with the input and output parameters, the function name, the messages, the port type with its input and output messages and finally, the SOAP binding description. This is all created from a standard template so that the author of the business process has nothing to do with it. He only sees the result which is returned, namely the day of the week. It is important that all of these technical details be hidden from the designer of the business processes to a great an extent as possible. (see Samples 3 & 4.: WSDL Messages)

6 Conclusion

Web Services offered within the framework of a Service Oriented Architecture promise to make applications more flexible, easier to compose and cheaper to develop. [24] In this paper it has been demonstrated how legacy code can be reused to help construct such web services. It would be unwise to ignore the vast amount of proven legacy software available within corporations and public administrations, when migrating to a service oriented architecture. Before developing or purchasing new service components, one should try to reuse the old ones. The technology for doing so is available. The approach presented in this paper is only one of several similar ones developed at the RCOST research institute in Benevento, at the University of Bari and at the IBM Research Institute in Toronto. [25] It has been proven there and elsewhere that specific business functions can be extracted from existing programs, wrapped and integrated into an eBusiness application framework. Doing so avoids the cost and risks of having to develop them from scratch. The savings is the difference between the cost of salvaging and wrapping the legacy functions as opposed to the cost of designing, coding and testing. It promises to be significant.

The approach described in this paper has been applied successfully for the integration of both COBOL and C++ programs.[26] There exists a Pl/I version, but it has yet to be proven in practice. The main problem has turned out to be reentrancy. The state of the data contained within a wrapped web service is that of the last caller. Thus, if different processes are using the same service, their data will be mixed. One solution is to store the internal data state in a temporary database under the id of that user. The other solution is to have a scheduler. Both solutions have advantages and disadvantages. However, this is not a problem specific to wrapped legacy code, but to all web services. It has to be solved in order for this technology to be accepted.

References

[1] Miller, H.: Reengineering Legacy Software Systems, Digital Press, Boston, 1998, p. 13

[2] Warren, Ian: The Renaissance of Legacy Systems, Springer Pub., London, 1999, p. 2

[3] Lavery,J./Boldyreff,B./Ling,B./Allison,C.: “Modelling the evolution of legacy systems to Web-based systems”, Journal of Software Maintenance and Evolution,Vol.16, Nr. 1,2004, p.5

[4] Litoiu, M.: “Migrating to Web Services – a performance engineering approach” Journal of Software Maintenance and Evolution, Vol. 16, Nr. 1, 2004, p. 51

[5] Tonella, P./ Ricca, F.: Statistical Testing of Web Applications, Journal of Software Maintenance and Evolution, Vol. 16, Nr. 1, 2004, p. 103

[6] Krafzig,D./Banke,K./Slama,D: Enterprise SOA, The Coad Series, Prentice-Hall Pub., Upper Saddle River, N.J., 2004, p. 6

[7] Egyed, A./Müller, H./Perry, D.: "Integrating COTS into the Development Process", IEEE Software, July, 2005, p. 16

[8] Gold, N./Bennett, K.: "Program Comprehension for Web Services", Proc. of 12th IWPC, IEEE Computer Society, Bari, June, 2004, p. 151

[9] Sneed, H.: "Testing a Web Application", Proc. of 6th Web Site Evolution, IEEE Computer Society, Chicago, 2004, p. 3

[10] Bovenzi, D./ Canforna, G./ Fasolina, A.: "Enabling Legacy System Accessibility by Web Heterogeneneos Clients", Proc. of 7th CSMR-2003, IEEE Computer Society Press, Benevento, March, 2003, p. 73

[11] Bodhuin, T./Guardabascio, E./ Totorella, M.: "Migrating COBOL Systems to the WEB", Proc. of 9th WCRE-2002, IEEE Computer Society, Richmond Va., Nov. 2002, p. 329

[12] Tilley, S./ Gerdes, J./ Hamilton, T./ Huang, S./ Müller, H./Smith, D./Wong, K.: " On the business value and technical challenges of adapting Web services", Journal of Software Maintenance and Evolution, Vol. 16, Nr. 1, 2004, p. 31

[13] Vorsamer, A.: "Java Tools help with Host-Integration", in Computer Zeitung, Nr. 32, August, 2005, s. 19

[14] Aversano, L./ Tortorella, M.: "An Assessment Strategy for identifying legancy system evolution requirements in eBusiness Context", Journal of Software Maintenance and Evolution, Vol. 16, Nr. 4, 2004, p. 255

[15] Ben-Menachem, M.: "Web Metadata Standards – Observations and Prescriptions", IEEE Software, February, 2005, p. 78

[16] Scheer, A.W.: "Where will the Program Code remain", in Computerwoche, Nr. 15, April, 2005, p. 22

[17] Sneed, H./ Erdoes, K.: "Extracting Business Rules from Source Code", Proc. of 4th IWPC-1996, IEEE Computer Society, Berlin, March 1996, p. 240

[18] Sneed, H.: Extracting Business Logic from existing COBOL Programs as a Basis for Reuse", Proc. of 9th IWPC-2001, IEEE Computer Society, Toronto, May, 2001, p. 167

[19] Hasselbrink, W.: "Information System Integration" Comm. Of ACM, Vol. 43, No. 6, June 2000, p. 33

[20] Aversano, L./Canfora, G./Cimitile, A./ DeLucia, A.: "Migrating Legacy Systems to the Web" Proc of 5th CSMR, IEEE Computer Society Press, Lisabon, March 2001, p. 148

[21] Sneed, H.: "Wrapping Legacy COBOL Programs behind an XML Interface" Proc. of 8th WCRE-2001, IEEE Computer Siciety Press, Stuttgart, Oct. 2001, p. 189

[22] Sneed, H.: "Program Interface Reengineering for Wrapping", Proc. of 4th WCRE, IEEE Computer Society Press, Amsterdam, Oct. 1997, p. 206

[23] Juric, M./Mathew, B./ Poornachandra, S.: Business Process Execution Languge for Web Services, Packt Pub., Birmingham, U.K., 2004, p. 17

[24] Jones, S.: "Towards an acceptable Definition of Services", IEEE Software, May 2005, p. 87

[25] Zou, Y./ Lau, T./ Kontogiannis, K.: "Model Driven Business Process Recovery", Proc. of 11th WCRE-2004, IEEE Computer Society Press, Delft, N.L. Nov. 2004, p. 224

[26] Sneed, H./ Sneed, S.: Web-based System Integration, Vieweg Verlag, Wiesbaden, 2004, p. 257

Sample 1 : Input Schema generated from COBOL Module

```
<schema   name = "xm059i"
          xmlns= "XSDCOB">
  <XSDCOB:complexType type = "#file" name = "xm059i"
           content = "eltOnly" model = "closed">
      <XSDCOB:complexType type = "#params" name = "DayofWeekRequest"
             content = "eltOnly" model = "closed" level = "02"
             occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "unbounded">
        <XSDCOB:element type = "#dec" name = "DAY"
                 content = "TextOnly" model = "closed" level = "03"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0001" lng = "0002"
                 pic = "99" usage = "DISPLAY"/>
        <XSDCOB:element type = "#dec" name = "MONTH"
                 content = "TextOnly" model = "closed" level = "03"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0003" lng = "0002"
                 pic = "99" usage = "DISPLAY"/>
        <XSDCOB:element type = "#dec" name = "YEAR"
                 content = "TextOnly" model = "closed" level = "03"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0005" lng = "0004"
                 pic = "9999" usage = "DISPLAY"/>
        <XSDCOB:element type = "#dec" name = "LANGUAGE"
                 content = "TextOnly" model = "closed" level = "03"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0009" lng = "0001"
                 pic = "9" usage = "DISPLAY"/>
        <XSDCOB:element type = "#char" name = "ALIGNMENT"
                 content = "TextOnly" model = "closed" level = "03"
                 occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
                 pos = "0010" lng = "0001"
                 pic = "X" usage = "DISPLAY"/>
      </XSDCOB:complexType>
```

Sample 2 : Output Schema generated from COBOL Module

```
<schema   name = "xm059o"
          xmlns= "XSDCOB">
  <XSDCOB:complexType type = "#file" name = "xm059o"
           content = "eltOnly" model = "closed">
    <XSDCOB:complexType type = "#params" name = "DayofWeekResponse"
             content = "eltOnly" model = "closed" level = "02"
             occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "unbounded">
......<XSDCOB:element type = "#char" name = "RETURN-CODE"
             content = "TextOnly" model = "closed" level = "02"
             occurs = "1" minOccurs = "0001" maxOccurs = "0001"
             pos = "0000" lng = "0002"
             pic = "XX" usage = "DISPLAY"/>
      <XSDCOB:element type = "#char" name = "DAYOFWEEK"
             content = "TextOnly" model = "closed" level = "03"
             occurs = "ONEORMORE" minOccurs = "0001" maxOccurs = "0001"
             pos = "0011" lng = "0010"
             pic = "X(10)" usage = "DISPLAY"/>
    </XSDCOB:complexType>
  </XSDCOB:complexType>
</schema>
```

Sample 3 : User Business Process in BPEL4WS (Fragment of Code)

```
<process name = "Calender"
 xmlns:calender = "http://anecon.com/sneed/sample/" >
    <partnerLinks>
    <PartnerLink name = "CalenderUser"
```

```
                partnerLinkType = "calender:User"
                myRole = "Provider"
          partnerRole = "User" />
    </partnerLinks>
    <variables>
        <!-- inputs for Calender Functions -->
        <variable name = "Day" messageType = "calender:DayofWeekRequest"/>
        <variable name = "Month" messageType = "calender:DayofWeekRequest"/>
        <variable name = "Year" messageType = "calender:DayofWeekRequest"/>
        <variable name = "Language" messageType = "calender:DayofWeekRequest"/>
        <variable name = "Alignment" messageType = "calender:DayofWeekRequest"/>
        <!-- outputs for Calender Functions -->
        <variable name = "ResponseCode" messageType = "calender:DayofWeekResponse"/>
        <variable name = "DayofWeek" messageType = "calender:DayofWeekResponse"/>
    </variables>
    <assign>
        <copy>
            <from variable = "Current_Day" part = "Date" />
            <to variable = "Day" part = "DayofWeekRequest" />
        </copy>
        <copy>
            <from variable = "Current_Month" part = "Date" />
            <to variable = "Month" part = "DayofWeekRequest" />
        </copy>
        <copy>
            <from variable = "Current_Year" part = "Date" />
            <to variable = "Year" part = "DayofWeekRequest" />
        </copy>
    </assign>
    <!-- call Calender Service to provide Day -->
    <invoke partnerLink = "CalenderUser"
          portType = "CalenderStatusPT"
          operation = "GetDayofWeek"
          inputVariable = "DayofWeekRequest"
          output Variable = "DayofWeekResponse" />
    <assign>
        <copy>
            <from variable = "DayofWeek" part = "DayofWeekResponse" />
            <to variable = "WeekDay" part = "LetterHeader" />
        </copy>
    </assign>
</process>
```

Sample 4 : Input Message from User Business Process

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!DOCTYPE "xm059i" SYSTEM "xm059i.xsd">
<xm059i>
  <DayofWeekRequest>
     <DAY>12</DAY>
     <MONTH>10</MONTH>
     <YEAR>1977</YEAR>
     <LANGUAGE>3</LANGUAGE>
     <ALIGNMENT>1</ALIGNMENT>
  </DayofWeekRequest>
</xm059i>
```

Sample 5 : Output Message to User Business Process

```
<?xml version = "1.0" encoding = "ISO-8859-1"?>
<!--DOCTYPE  XM059O  SYSTEM "XM059O.xsd"-->
<XM059O>
  <DayofWeekResponse>
    <RETURN-CODE>00</RETURN-CODE>
    <DAYOFWEEK>MERCOLEDI</DAYOFWEEK>
  </DayofWeekResponse>
</XM059O>
```