TU VIENNA

TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

PhD Thesis

Business Process Modelling - Languages, Goals, and Variabilities

Conducted for the purpose of receiving the academic title
'Doktorin der Sozial- und Wirtschaftswissenschaften'

Supervisors

o.Univ.-Prof. Dipl.-Ing. Mag. Dr. Gerti Kappel
Institute of Software Technology and Interactive Systems
Vienna University of Technology

Ao. Univ.-Prof. Mag. Dr. Christian Huemer
Institute of Software Technology and Interactive Systems
Vienna University of Technology

Submitted to the
Vienna University of Technology
Faculty of Informatics
by

Birgit Korherr

9925559
Viktor Kaplanstrasse 10
3151 St. Georgen

Vienna, January 9th, 2008

Contents

List of Figures

List of Tables

Acknowledgement

I would like to thank Prof. Christian Huemer for the excellent mentoring/supervision throughout the entire PhD thesis. I also would like to thank Prof. Gerti Kappel and Dr. Beate List for supporting me.

Moreover, I would like to thank Matthias and my parents for their patience during this work.

But their are a lot of other people I would like to thank for their interest, friendship, and ideas and criticism:

Andrea Schauerhuber, Veronika Stefanov, Martina Umlauft, Sabine Graf, Nevena Stolba, Marion Murzek, Elke Michlmayr, Stefanie Scherzinger, Ulrike Pastner, Doris Kastner, Sonja Willinger, Gerhard Kramler, Horst Kargl, Michael Strommer, Manuel Wimmer, Martina Seidl, Michael Schadler, Daniela Knitel, Christian Breiteneder, Christiane Floyd, Ute Riedler, and many more.

Abstract

Over the last decade more and more companies started to optimize their business processes in a way to meet its business goals. They develop business process models defining which activities have to be executed in which order under which conditions by whom and by using which resources. For this purpose a lot of different approaches to business process modelling have been developed, which resulted in many different Business Process Modelling Languages (BPMLs).

The definition of a business process has to cover many different aspects (e.g. control flow, organizational view, data view, etc.). A perfect business process modelling approach would address all the different aspects. Unfortunately, none of the existing approaches provides concepts for addressing all of these aspects. Each of them concentrates on some aspects. The focus on certain aspects is mainly due to the different applications areas, e.g. business engineering or software engineering etc.

Although BPMLs are well established in industry and science, a comprehensive evaluation or a framework for an evaluation to compare the different BPMLs is still missing. Thus, it is the goal of this thesis to provide an evaluation framework for the comparison of BPMLs and to apply this framework in the evaluation of the currently most popular BPMLs. The resulting framework is based on a generic metamodel that captures all of the concepts appearing in any of the state-of-the-art BPMLs. On a high level this framework addresses the following views: Business Process Context Perspective, Behavioural Perspective, Functional Perspective, Informational Perspective, and Organisational Perspective. An evaluation based on this framework checks whether the certain aspects in each of these perspectives is supported by the concepts of each of the considered BPMLs. In the evaluation of this thesis, we used the following languages: UML 2 Activity Diagram, Business Process Modelling Notation, Event Driven Process Chain, IDEF3, Petri Net, Role Activity Diagram.

According to the evaluation we were able to identify three main problems in current BPMLs. The first problem is that the definition of the dependency between business processes and their supporting software systems is inadequately supported. In our approach we support the elicitation of requirements from business process models for the software systems to be developed by extending current BPMLs with software requirements and components to ensure a business-goal oriented software development.

The second problem concerns the variability of similar, but well-distinguished software products within a software product line. These software products not only differ in its structural definition, but also in the process to create them. Today, variability modelling is a domain specific modelling technique that is limited to the structural definition of similar software products. In our approach we extend the concepts of variability modeling to integrate the dynamical aspects into the UML. The resulting approach is based on a well defined dependency between UML class diagrams and UML activity diagrams.

The third problem is that current conceptual BPMLs do not provide explicit modelling means for process goals and their measures. The modelling of goals and its monitoring is a critical step in business process modeling. Hence, we extend the metamodels of UML 2 AD, EPC and BPMN with business process goals and performance measures. These concepts become explicitly visible in the corresponding models. Furthermore, a mapping of the performance measures onto the Business Process Execution Language (BPEL) enables their monitoring in an execution environment.

Kurzfassung

Seit dem letzten Jahrzehnt begannen Unternehmen ihre Geschäftsprozesse im Hinblick auf ihre Geschäftsziele zu optimieren. Geschäftsprozessmodelle wurden definiert, die zeigen welche Aktivitäten unter bestimmten Bedingungen und in einer bestimmten Reihenfolge unter der Verwendung von Ressourcen in einem Prozess durchgeführt werden. Für diesen Zweck sind eine hohe Anzahl von Geschäftsprozessmodellen entwickelt worden, aus denen viele unterschiedliche Geschäftsprozesssprachen hervor kamen.

Die Definition eines Geschäftsprozesses muss viele unterschiedliche Aspekte (Kontrollfluss, Organisationssicht, Datensicht, usw.) beinhalten. Ein perfekter Ansatz für die Geschäftsprozessmodellierung würde all diese Konzepte umfassen. Momentan erfüllt aber kein existierender Ansatz für die Modellierung von Geschäftsprozessen all jene Aspekte, da sich eine jede dieser Sprachen nur auf bestimmte Aspekte konzentriert, die sich auf ihr bestimmtes Anwendungsgebiet beziehen, wie z.B. Software Engineering oder Business Engineering.

Obwohl Geschäftsprozessmodellierungssprachen (GPMS) ihren Platz in Wirtschaft und Wissenschaft gefunden haben, gibt es weder eine umfassende Evaluierung noch ein Framework für eine Evaluierung, um die unterschiedlichen Sprachen zu vergleichen.

Deshalb ist es das Ziel dieser Arbeit, ein Evaluierungsframework für einen Vergleich der verschiedenen Sprachen zu entwickeln. Das resultierende Framework basiert auf einem generischen Metamodell, das jene Konzepte beinhaltet, die auch in gängigen Modellierungssprachen vorkommen. Das Framework besteht aus folgenden fünf Sichten: Geschäftsprozess-Kontext-Sicht, Verhaltenssicht, funktionale Sicht, Informationssicht und Organisationssicht.

Eine Evaluierung basierend auf diesem Framework macht Aussagen darüber, ob die untersuchten Sprachen die Konzepte der unterschiedlichen Sichten erfüllen oder nicht. In dieser Evaluierung wurden folgende Sprachen untersucht:

UML 2 Activity Diagramm (UML 2 AD), Business Process Modelling Notation (BPMN), Event Driven Process Chain (EPC), IDEF3, Petri Net, Role Activity Diagramm (RAD).

Anhand dieser Evaluierung war es nun möglich, drei grundlegende Probleme in den verschiedenen Sprachen zu identifizieren.

Das erste Problem behandelt den Umstand, dass die Abhängigkeiten zwischen Geschäftsprozessen und ihren unterstützenden Softwaresystemen nur unzureichend dargestellt werden können. In unserem Ansatz werden die Anforderungen, die Geschäftsprozessmodelle an Softwaresysteme stellen, unterstützt, in dem eine existierende GPMS um Softwareanforderungen und -komponenten erweitert wurde.

Das zweite Problem adressiert die unterschiedlichen Variabilitäten von Softwareprodukten innerhalb einer Softwareproduktlinie. Diese Softwareprodukte unterscheiden sich nicht nur in ihrer strukturellen Definition, sondern auch im Prozess, um diese zu kreieren. Variabilitätsmodellierung ist ein domänenspezifischer Ansatz, der aber auf die strukturelle Definition von ähnlichen Softwareprodukten beschränkt ist. In dieser Arbeit werden die Konzepte der Variabilitätsmodellierung erweitert, um deren dynamischen Aspekt in UML integrieren zu können. Der resultierende Ansatz basiert auf einer definierten Abhängigkeitsbeziehung zwischen UML Klassen- und Aktivitätsdiagrammen.

Das dritte Problem behandelt das Thema, dass jetzige Geschäftsprozessmodellierungssprachen keine Konzepte beinhalten, um Prozessziele und deren Kennzahlen zu modellieren. Das Modellieren von Zielen und deren Überwachen ist ein wichtiger Schritt in der Geschäftsprozessmodellierung. Deshalb wurden im Rahmen dieser Arbeit die Metamodelle von UML 2 AD, EPC und BPMN um Ziele und die Kennzahlen Zeit, Kosten und Qualität erweitert, um diese auch grafisch sichtbar zu machen. Weiters ermöglicht ein Mapping der Kennzahlen Zeit zur Business Process Execution Language (BPEL) die Überwachung in einer Ausführungsumgebung.

1 Introduction

1.1 Problem Statement

Conceptual Business Process Modelling Languages express different aspects of processes (e.g., activities, roles, interactions, data, etc.) and address various application areas. To adequately describe a business process, a process model must integrate different aspects. Today, there are many conceptual Business Process Modelling Languages (BPMLs) available. BPMLs differ in the extent to which they support the various aspects of business process modelling. Some languages were developed from a software engineering background, and other languages have a process engineering background. This leads to the following problems:

Problem 1. Today, there are a lot of Conceptual Business Process Modelling Languages available. To adequately describe a business process, many types of information must be integrated into a process model. Information that people want to extract from process models includes what is going to be done, who is going to do it, when and where will it be done, how and why will it be done, and who is dependent on its being done [CKO92]. BPMLs also differ in the extent to which their constructs highlight the information that answers these different questions. Although BPMLs have been widely used in research and industry, a comprehensive comparison is missing. For evaluating BPMLs, a general framework is required, but such a framework is not available.

Problem 2. Business processes are often the starting point for software development. They define requirements for software systems to be developed. However, the connection between business processes and software development is inadequately supported in conceptual modelling. It is not possible to show the software requirements of a process or the software components that are used to successfully implement a process.

Problem 3. A variability model defines the variability of a software product line and can be used during different life cycle stages of software product lines [PBvdL05]. Variability modelling is a domain specific modelling technique which is becoming more and more integrated into traditional software engineering. Variability models also have an impact on processes, because variabilities can change the process flow. Unfortunately they are not part of a well-known modelling framework, like the Unified Modelling Language, which would increase their visibility and their usability. Furthermore, it is also not possible to show variabilities in a business process model.

Problem 4. A business process is defined as a "group of tasks that together create a result of value to a customer" [Ham96]. Its purpose is to offer each customer the right product or service, i.e., the right deliverable, with a high degree of performance measured against cost, longevity, service and quality [JEJ94]. Although business process performance measurement is an important topic in research and industry [Cas05], current conceptual BPMLs do not provide explicit modelling means for process goals and their measures. Already in 1997, Kueng and Kawalek argued that little attention is paid to the value of making goals explicit [KK97]. Furthermore, the measures need to be integrated into the process execution and require continuous monitoring. Although process goals and performance measures are available in process theory, they lack the visibility in conceptual BPMLs.

1.2 Contribution

The contribution of this thesis is to provide solutions to the above mentioned problems:

1.2.1 Generic Metamodel for Evaluating BPMLs

We have developed a generic metamodel that captures a wide range of business process concepts, and is structured into four perspectives according to the framework of Curtis et al. [CKO92], namely the organisational, functional, behavioural, and informational perspectives. Furthermore we have added a further perspective, the business process context perspective to capture important business process context information like process goals or process type.

We have evaluated six BPMLs according to this metamodel (UML 2 Activity Diagram, Business Process Definition Metamodel, Business Process Modelling Notation, Event Driven Process Chain, IDEF3, Petri Net, Role Activity Diagram). Based on the comprehensive evaluation of the BPMLs we recognized which languages have a high potential to become successful in the future. These languages are the Activity Diagram, the Business Process Modelling Notation, and the Event Driven Process Chain. They support most concepts of the functional and behavioural perspectives, and some concepts of the organisational and information perspective. Furthermore it can be shown that the business process context perspective is not explicitly supported by all evaluated languages.

1.2.2 Capturing Software Requirements for Software Engineering Purposes

Business process models can be utilised to identify requirements for a new software system, and also for checking whether the features of an existing software system match the requirements of a new business process.

We support these issues by extending current BPMLs with elements that represent software requirements and components, in order to ensure a business-goal oriented software development. A linking between business processes and software is a further step towards bridging the gap between business process engineering and software engineering.

1.2.3 Variabilities in Business processes

Variability models are related to business process models, because variabilities can change the process flow. For example, in a car engine manufacturing process the decision between the variability *manufacture a diesel engine* or *manufacture a petrol engine* changes the process flow. We have provided a UML 2 profile for variability models, which can be applied to activity diagrams to make the relationship between variability models and process models visible.

1.2.4 Goals and Performance Measures in BPML

The modelling of goals is a critical step in the creation of useful process models. Goal are used for structuring and evaluating the process design, and allow for a better understanding of the broader implication of the process design, and can also be used for evaluating the operating process [KK97].

Unfortunately, process goals and performance measures are only available in process theory. We have addressed this limitation by extending conceptual BPMLs with business process goals and performance measures to make them conceptually visible. Furthermore, a mapping of the performance measures onto the Business Process Execution Language (BPEL) makes them available for execution and monitoring. The performance measure *time* was exemplary made available for execution and monitoring by mapping it to the Business Process Execution Language (BPEL).

1.2.5 Extension Mechanisms used for BPMLs

For implementing business process goals and performance measures in the BPMLs Activity Diagram, BPMN, and EPC, an adequate mechanism is needed.

The Activity Diagram is part of the Unified Modeling Language (UML). UML offers a mechanism for extending its metamodel to a specific application area. Through the creation of profiles it is possible to build UML models for particular domains or purposes [OMG05c]. A profile can extend a metamodel or another profile [OMG05c] while preserving the syntax and semantic of existing UML elements. It adds elements which extend existing metaclasses. UML profiles consist of stereotypes, constraints and tagged values. UML profiles are UML packages with the stereotype "profile". The activity diagram can be easily enhanced with new concepts by extending its metamodel and creating a UML profile. Figure 1.1 (based on the language architecture of UML shown in [HKRK05]) describes the relation between UML and its meta-metamodel *Infrastructure* of which UML is an instance.

BPMN provides a graphical notation to express business processes. The Business Process Definition Metamodel (BPDM) [OMG07a], which is an XML-based proposal for modeling business processes, offers a mapping to BPMN. Furthermore BPDM provides a general process modeling metamodel that supports the BPMN notation. We have deduced a simple MOF-based metamodel of BPDM, which includes all metaclasses which are used to describe a business process with BPMN.

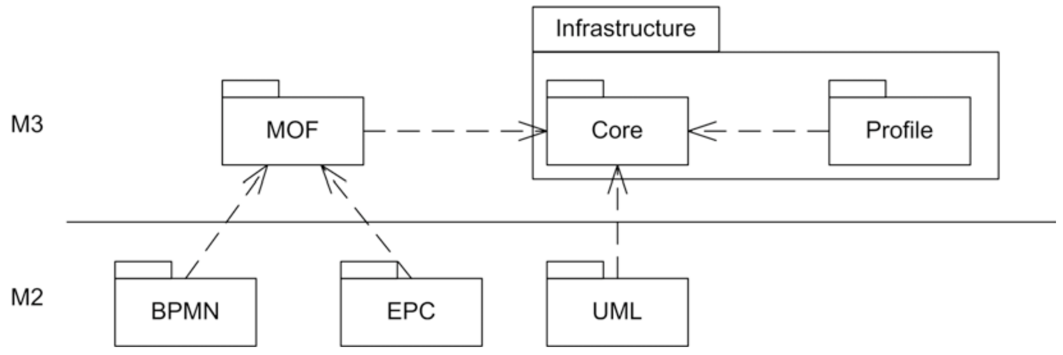


Figure 1.1: Relationships between BPMLs and their metamodels

The EPC consists of different views, and each view has its own metamodel. The problem is that EPC only provides metamodels for its views, but not an integrated metamodel that contains all views in one model. We have derived a metamodel for the EPC which shows all views in one metamodel.

The metamodels of BPMN and EPC are MOF-based, as figure 1.1 shows. The Meta-Object Facility (MOF) is a language for specifying metamodels, and is itself a meta-metamodel. MOF defines a modelling language that is identical to a subset of UML [OMG05a]. Figure 1.1 shows the relationship between BPMN, EPC and UML 2 AD compared to MOF. From the statements above we conclude that every language that has a metamodel which is based on MOF can be extended is the same way. We have adapted and extended the derived metamodels to integrate goals and performance measures as well as software concepts.

1.3 Structure of the Thesis

Section 2 gives an overview of the basic concepts of a conceptual Business Process Modelling Language (BPML). It discusses the evaluation of six well-known BPMLs. In order to adequately describe a business process, many types of process concepts must be integrated into a business process model. It shows which aspects of processes (e.g., activities, roles, interactions, data, etc.) are expressed in the different BPMLs and which concepts are not supported. The section also presents a generic generic metamodel that captures a wide range of business process concepts.

Section 3 provides the extension of the UML 2 Activity Diagram (UML 2 AD), the Event-Driven Process Chain (EPC), and the Business Process Modeling Notation (BPMN) with goals and performance measures. UML 2 AD, EPC and BPMN are designed for modelling business processes, but do not yet include any means for modelling process goals and their measures. To fill this gap, section 3 presents an extension of these BPMLs with business process goals and performance measures. This includes both the derived metamodels for the EPC and BPMN languages, and also the extensions of the metamodels of UML 2 AD, EPC and BPMN with business process goals and performance measures to make them conceptually visible.

Section 4 describes the mapping of the UML 2 Profile for Business Process Goals and Performance Measures (introduced in section 3) to BPEL, which makes it available for automatic execution and monitoring. Measures need to be integrated into the process execution and require continuous monitoring. The UML 2 profile and its mapping onto BPEL enable the transformation of the business processes models developed in a UML modelling tool into BPEL. Thus, the performance measures conceptually described in the business process model can be directly transformed into the execution language and can be used to monitor the process instances continuously.

Section 5 is focused on the connection of business processes with software. On the on hand side it is not possible to show software requirements or components which have an influence on a business process directly with a BPML. On the other hand, it is also not possible to show the variabilities of a business process. To integrate software requirements and components into a BPML, section 5 describes how the UML 2 profile for EPCs we had developed previously [KL06] can be linked with software. The profile is connected to UML 2 elements, to describe the software requirements of a process as well as software components that are needed to successfully realise a process. Variability models are designed for modelling variabilities of software, but they are not part of a well-known modelling framework like the Unified Modelling Language, which would give them higher usability. To address this limitation, section 5 introduces the UML 2 profile for variability models. Furthermore, the dependencies between this UML profile and activity diagrams are shown to make the relationship between variability models and process models visible.

Section 6 discusses related work in both the area of evaluation frameworks as well as business process model extensions.

There exist many frameworks with different perspectives for evaluating languages, like pattern frameworks of metamodels, which are described in section 6.

The main focus of conceptual BPMLs is organizing the flow of tasks of a business process, but not the explicit modeling of process goals and their measures. The section gives overview of approaches for integrating business process goals and their measures in a BPMLs.

Although business processes are often the starting point for software development and define requirements for software systems, linking business processes with software systems is inadequately supported in conceptual modelling. It is not possible to show software requirements or components which have an influence on a business process, or to show the variabilities involved. A section of relevant approaches to solve these gaps is presented in section 6.

2 Evaluation of Conceptual Business Process Modelling Languages

Contents

2.1 Introduction

Today, there are a lot of conceptual Business Process Modelling Languages (BPMLs) available. This section discusses the evaluation of six well-known conceptual BPMLs. It shows which aspects of processes (e.g. activities, roles, interactions, data, etc.) are expressed in the different BPMLs and which are not supported. In order to adequately describe a business process, many forms of process concepts must be integrated into a business process model. Information that people want to extract from process models are [CKO92]:

- what is going to be done,
- who is going to do it,
- when and where will it be done,
- how and why will it be done,
- and who is dependent on its being done.

BPMLs differ in the extent to which their constructs address these questions. The differences result from the various source domains (e.g. process or software engineering etc.), as well as from the application areas targeted. Although BPMLs have been widely used in research and industry, a comprehensive comparison is missing.

Also, a general framework for an evaluation of BPMLs is not available. In order to overcome these gaps, we address these limitations by:

- Developing a generic metamodel that captures a wide range of business process concepts, because metamodels represent the core concepts of BPMLs and are a good foundation for an evaluation.
- Evaluating six well-established BPMLs according to the generic metamodel.

Our generic metamodel (Section 2.2) is categorised according the framework of Curtis et al. [CKO92]. This metamodel is derived from business process theory [Ham96] [Har91] [JEJ94] [Mar95] and well-established industry and research concepts.We have evaluated six BPMLs according to the meta-model and additional criteria: UML 2 Activity Diagram, Business Process Modelling Notation, Event Driven Process Chain, IDEF3, Petri Net, Role Activity Diagram. The contribution of this evaluation of conceptual BPMLs is:

- It provides a comprehensive evaluation of the most well-established and widely-used BPMLs, and those which have a high potential to become successful in the future.
- It stresses strengths and limitations of BPMLs.
- The evaluation facilitates selecting the BPML, which is adequate for a certain purpose.
- The metamodel provides a common foundation for evaluating BPMLs. This ensures an objective evaluation that covers basic process concepts and their relationships.
- The comparison between the BPMLs illustrates the differences and the similarities of the languages.
- The evaluation of six BPMLs provides a foundation that can be extended with further BPMLs.

In particular, the remainder of this section comprises the following sections:

- Subsection 2.2 describes the generic metamodel we have developed to compare different BPMLs.
- Subsection 2.3 characterises the different BPMLs which have been chosen for evaluation.
- Subsection 2.4 defines the evaluation method and presents the results of the evaluation.

2.2 The Metamodel

In this section, the generic metamodel in figure 2.1 that serves as the basis for the evaluation of the BPMLs is described with examples for a better understanding. We apply the conceptual framework of Curtis et al. [CKO92] in order to receive a comprehensive metamodel and to ensure that the basic building blocks of business processes are covered. The framework consists of four perspectives: organisational, functional, behavioural, and informational. As these perspectives do not capture important information like process goals or measures, we extend the framework with a further perspective, namely the business process context perspective. The generic metamodel is inspired by business process theory [Ham96] [Har91] [JEJ94] [Mar95], workflow patterns [RtHEvdA05], [vdAtHKB03], and the Workflow Management Coalition (WfMC) [Wor98]. In the following subsections, we describe the different perspectives in general and the generic metamodel elements in particular.

2.2.1 Functional Perspective

The functional perspective represents the process elements which are performed during a business process [CKO92]. The basic elements of a business process are *Activities*. They can be either *Atomic Activities* or *Sub-Processes*, which are recursively refined by activities. For instance a business process in an insurance company is *Processing of Automobile Insurance Claims*.

The main activity consists of the sub-processes *Assertion of the Claim* and *Compensation of the Claim*. Furthermore the sub-process *Assertion of the Claim* consists of the atomic activities *Record the Claim* and *Calculate the Insurance Sum*.

2.2.2 Organisational Perspective

The organisational perspective represents where and by whom (which agents) process elements are performed [CKO92]. The metamodel elements of this perspective are inspired by the 4 types of workflow participants of the WfMC [Wor98]: the organisational unit, the role, the (individual) human, and the (automatic) resource. In the metamodel in figure 2.1, we use the term *Process Participant*. If the process participant is a member of the organisation, then it is called an *Internal*. For instance

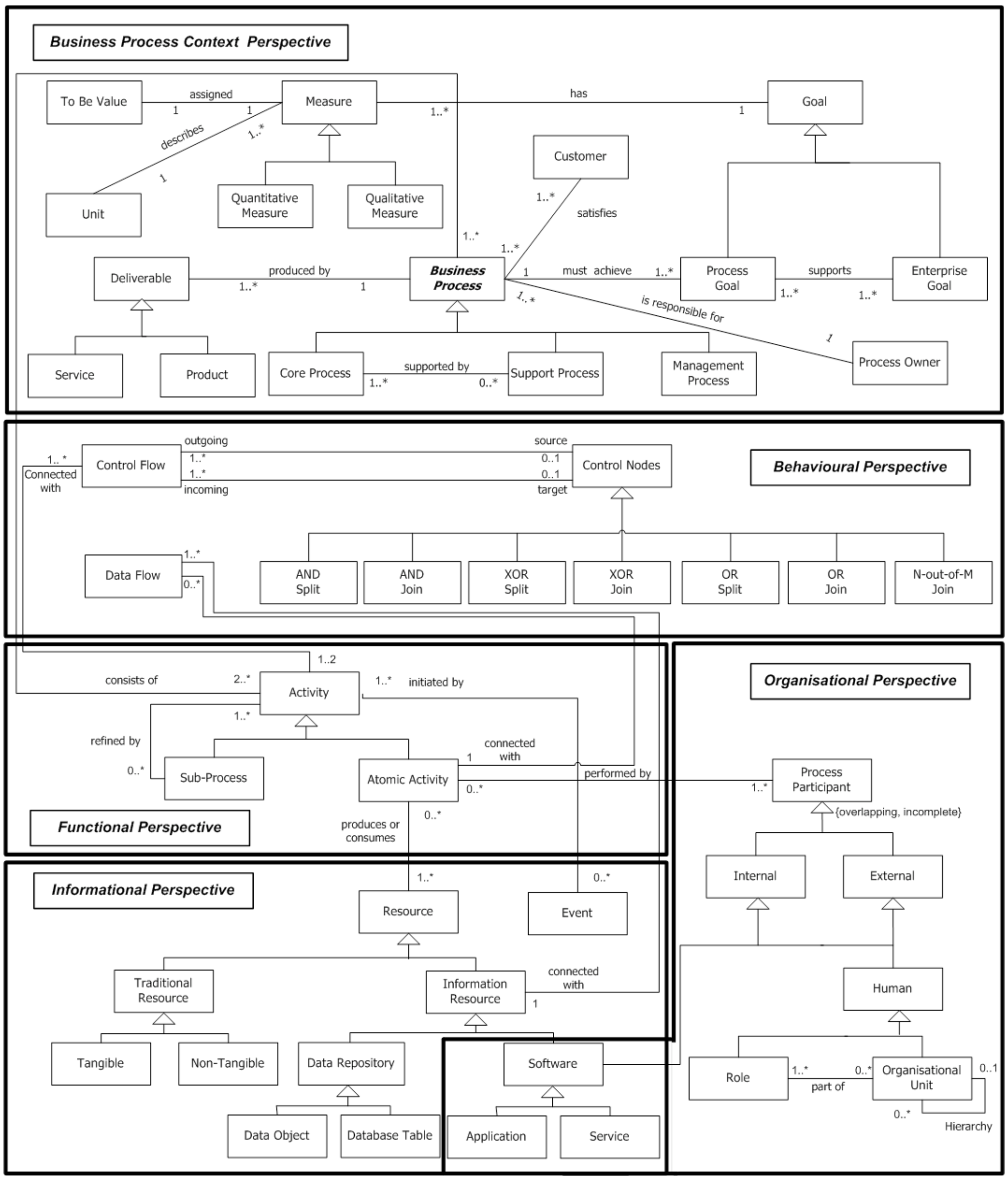


Figure 2.1: Generic Metamodel of a Business Process

a customer or a supplier, that is not part of an organisation, is called *External*. We identified 3 types of participants that perform a process: the *Organisational Unit*, the

Role, and the *Software*. If an organisational unit is addressed, its members may perform the activity. If a role is addressed, an activity is performed by a role or skill set. In this context a role is a function of a human in an organisation. In the metamodel a *Human* is represented as a role or an organisational unit. More and more activities are performed automatically by software. It should be possible to make this transparent in a process model; Software can be either an *Application* or a *Service*.

The process participants of the business process *Processing of Automobile Insurance Claims* are the organisational roles *Financial Claim Specialist* and *Claim Administrator*. The *Financial Claim Specialist* is responsible for the sub-process *Assertion of the Claim* and the organisational role *Claim Administrator* for *Compensation of the Claim*.

2.2.3 Behavioural Perspective

The behavioural perspective represents when process elements are performed (e.g., sequencing), as well as aspects of how they are performed through feedback loops, iteration, complex decision-making conditions, entry and exit criteria, and so forth [CKO92]. The *Data Flow* connects atomic activities with information resources. The other metamodel elements of this perspective are adopted from the workflow control patterns [vdAtHKB03]. These patterns capture elementary aspects of process control. All basic control workflow patterns are integrated in the behavioural perspective, since they focus on the flow of the business process. These patterns are: *Sequence, AND Split, AND Join, XOR Split, XOR Join*. All these elements are not workflow specific and are also required for modelling business processes. They are also available in a lot of BPMLs. In this thesis, the sequence is called *Control Flow*, and the operators are called *Control Nodes*. Furthermore, we integrated the advanced branching and synchronisation patterns [vdAtHKB03] for business processes. The patterns cover three types of merge operations addressing three different types of synchronisation. As synchronisation is not an issue for business processes, we integrate an OR Join representing all three merges as well as the corresponding OR Split. Further, we integrate the *N-out-of-M Join* that merges many execution paths. For example, a paper needs to be sent to three external reviewers. Upon receiving two reviews on time the paper can be processed, the third one can be ignored.

2.2.4 Informational Perspective

The informational perspective represents the informational entities produced or manipulated by a process; these entities include data, artefacts, products (intermediate and end), and objects [CKO92]. The metamodel of the informational perspective is inspired by the workflow data patterns [RtHEvdA05] as well as by the input / output view of the Architecture of Integrated Information Systems (ARIS) [Sch99]. The basic elements of the informational perspective shown in figure 2.1 are resources and events. An *Event* may trigger an activity. A *Resource* is an entity to be produced or consumed by an atomic activity. We distinguish between traditional and information resources. *Traditional Resources* have been inspired by ARIS [Sch99] and can be either *Tangible* (e.g. a product) or *Non-Tangible* (e.g. service). Information Resources are inspired by the workflow data patterns [RtHEvdA05], which propose three types of environment data: data repositories, applications and services. We created a *Data Repository Resource* and distinguish between a *Database Table* and a *Data Object*, which contains persistent data, e.g. data in a document or a form. *Applications* and *Services* are both *Software Resources*. They are modelled in the informational perspective as well as in the organisational perspective.

The atomic activity *Calculate the Insurance Sum* needs for further processing the application *Claim Management System* to save, delete, destroy files of the insurance claim. If processing *Calculate the Insurance Sum* is successful, then the service *Insurance Sum Calculated* is produced.

2.2.5 Business Process Context Perspective

In a previous work [LK05] we have developed the business process context perspective, which shown in figure 2.1. The perspective presents a business process from a wide angle. It provides an overview perspective of the process and describes major business process characteristics, such as goals and their measures, the deliverables, the process owner, the process type and the customer at a glance. We have integrated the process characteristics into the metamodel, because they represent essential process theory that should be transparent in a process model. People who do not know or do not need to know the process in detail will get a high level understanding of the process without working through the complex process logic. All other perspectives cover the detailed sequence of the process and do not address theses important process characteristics. The metamodel in figure 2.1 presents all characteristics of a business process. A *Business Process* has a certain process type that can be either a *Core Process*, a *Support Process* or a *Management Process* [Mar95]. A core process is either independent from support processes or supported by one or more support processes. A business process satisfies one or more *Customers*. Activities describe a business process in detail. *A Process Owner* [Ham96] is responsible for one or more business processes. Each business process generates one or more *Deliverables* [JEJ94], which are either *Services* or *Products*. Each business process must achieve one or more *Process Goals* [KK97], which in turn support one or more *Enterprise Goals*. Concrete *Measures* describe the achievement of goals [Har91], both process and enterprise goals. Each measure is assigned to a *To Be Value*, which should be reached by the corresponding process instance. Furthermore a *Unit* is connected with one or more measures.

The core process *Processing of Automobile Insurance Claims* has to fulfill the process goal *Fast Processing of Claims*, which supports the enterprise goal *High Customer Satisfaction*. The core process produces the service *Payment of Compensation* for a customer, that is an insured person or organisation. The process owner *Deputy CEO* is responsible for *Processing of Automobile Insurance Claims*.

2.3 The Business Process Modelling Languages - An Overview

In this section, we describe the BPMLs which have been chosen for evaluation. The languages have either a future potential or are well-established in research or industry. Furthermore, each language is described with an example business process of an insurance claim.

2.3.1 UML 2.0 Activity Diagram (AD)

The origin of UML lies in the development of software. It consists of six structural diagrams, and seven behavioural diagram. The AD [OMG05c] belongs to the behavioural diagrams, and is designed for modelling business processes and flows in software systems. The diagram is approximated on Petri Nets, and uses also the notion of token. The main concepts of the AD are actions and activity partions. An activity partition used to group actions, that are executed by a certain role. Figure 2.3 shows the abstract syntax of UML 2 AD, and figure 2.2 the concrete syntax of the main elements.

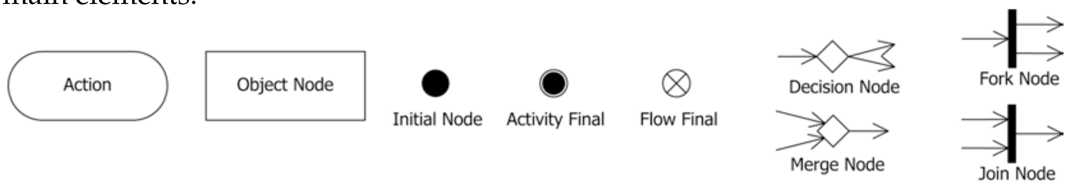


Figure 2.2: Elements of the UML 2 Activity Diagram

Figure 2.3 shows the main metaclasses in a section of the UML 2 metamodel of the Activity Diagram. The main elements of an Activity Diagram is the *Activity* and its different *ActivityNodes*. *Action, ObjectNode, ControlNode* are a specialisation of an *ActivityNode*. *Action* describe the atomic task of an AD. The abstract metaclass *ObjectNode* represents the instance of a specific classifier. The Control Nodes define the behaviour of an activity diagram. The *InitialNode* starts the Activity. If an Activity contains more *InitialNodes*, then the different flows execute concurrently. The *FinalNode* is split up into *ActivityFinalNode* and *FlowFinalNode*. While the *ActivityfinalNode* terminates all flows within an Activity, the *FlowFinalNode* only terminates one flow, and the Activity is unaffected. The *ForkNode* splits the flow into concurrent pathes. A *ForkNode* has one incoming flow and several outgoing flows. A *JoinNode* has serveral incoming flows and one outgoing flow. The *JoinNode* merges the concurrent pathes into one outgoing flow. Furthermore a condition may be placed at the incoming edges of the *JoinNode*. A *DecisionNode* has one incoming flow and several outgoing flows. A *DecisionNode* splits up into several alternative flows. Only one outgoing flow will be chosen for further processing. The *MergeNode* merges the outgoing flows of the *DecisionNode*. A merge node brings together multiple alternate flows and it is not used to synchronize concurrent flows. In an activity the flow of control from one node to another is modeled using *ControlFlow* edges and *DataFlow* edges. The *ControlFlow* models the flow between Actions, and the *DataFlow* between *ObjectNodes* and Actions. An *ActivityPartition* groups set of actions that have something in common.

Example 2.1 The example business process of an UML 2 AD is presented in figure 2.4. The business process starts with an initial node, to activate the first action, *Record the Claim*. Record the claim passes the token to the next action to *Calculate*

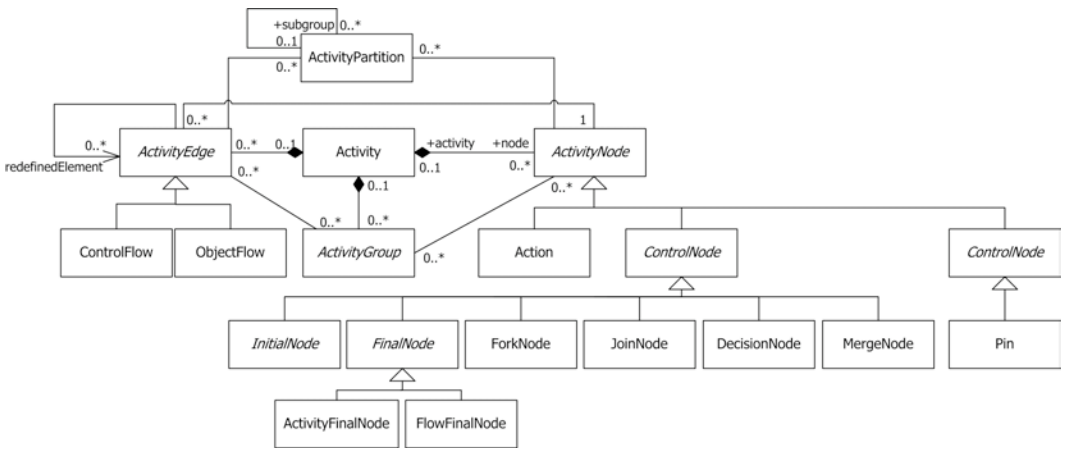


Figure 2.3: Section of the UML 2 Metamodel for the Activity Diagram

the Insurance Sum. These two actions are part of the activity partition *Financial Claim Specialist*. After calculating the insurance sum the path is split up by a decision node into two alternative flows, depending if the insurance sum has a minor amount or a major amount. If the insurance sum has a minor amount, then the action *Contacting the Garage* starts. If the insurance sum has a major amount, then the flow is split up into parallel pathes by a fork node. That means that the actions *Contacting the Garage* and *Checking History of the Customer* are executed concurrently. A merge node combines the different flows, and accepts the token as well as of one path or of both pathes. The action Examination of Results decides that the claim is handled either positive or negative. Therefore a decision node splits up the path in two alternative flows, with the actions *Pay for the Damage* or *Do Not Pay for the Damage*. After that decision the business process ends with an flow final node.

2.3.2 Event Driven Process Chain (EPC)

The EPC [Sch99] has been developed within the framework of ARIS (see fig. 2.6) and is used by many companies for modelling, analysing, and redesigning business processes. EPCs were developed in 1992 at the Institute for Information Systems of the University of Saarland, Germany, in collaboration with SAP AG. It is the key component of SAP R/3s modelling concepts for business engineering and customising. The EPC is based on the concepts of stochastic networks and Petri nets. EPCs are a graphical business process description language. They describe processes on the level of their business logic, and are targeted to be easy understood and used by

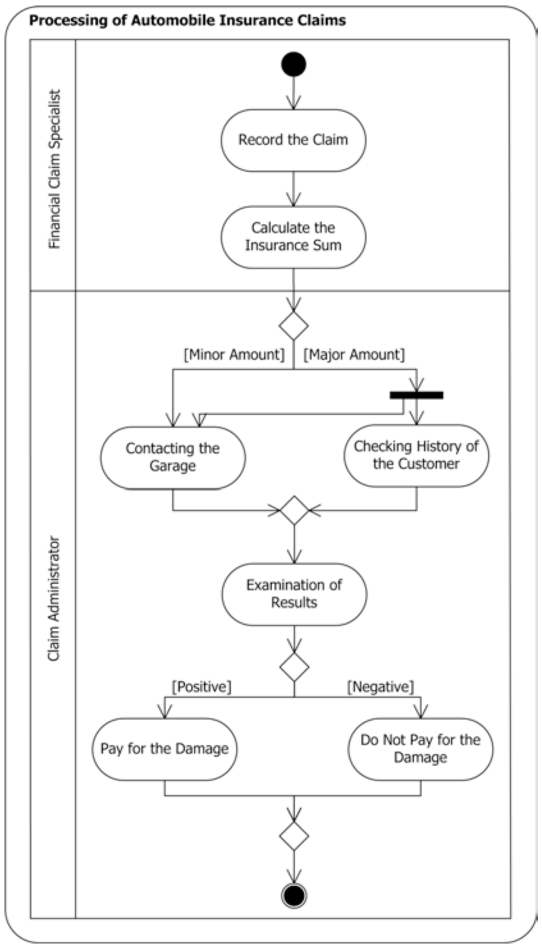

Processing of Automobile Insurance Claims
Financial Claim Specialist
Record the Claim
Calculate the Insurance Sum
Claim Administrator
[Minor Amount]
[Major Amount]
Contacting the Garage
Checking History of the Customer
Examination of Results
[Positive]
[Negative]
Pay for the Damage
Do Not Pay for the Damage

Figure 2.4: Example business process of an Activity Diagram

business people. The name represents the control flow structure of the process as a chain of events and functions. The different notation elements of EPC are shown in 2.5. A basic EPC consists of the following elements:

- Functions are active elements and model the activities within the company.
- Events are created by processing functions or by actors outside of the model. An event acts as a pre-condition of one function, or correspond to the post-condition of another one.
- Logical operators connect functions and events. There are three types of logical operators: AND, XOR (exclusive or) and OR.

The extended EPC consists of the following elements:

- The Organisation Unit or Role is responsible for performing an activity or a function.
- The Information Objects portray input data serving as the basis for a function, or output data produced by a function. They correspond to entities or attributes of the ER model.
- The Deliverables represent results (services or products) functions produce or input functions require.

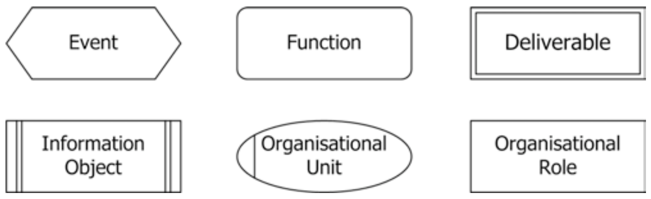


Figure 2.5: Notation of EPC Elements

The Architecture of Integrated Information System (ARIS) concept [Sch99] involves dividing complex business processes models into separate views, in order to reduce the complexity. Three views focus on functions, data, and the organisation. The forth view, the control view, focus on the integration of the other three views. The different ARIS views are shown in figure 2.6. On the one hand side the views can be handled independently, for instance to show with the oranisation view an organigram,. On the other hand side the views can operate together, for example to describe a business process with its function, event and organisation view.

The *Data View* contains events and statuses. Events such as ″customer order received″, or ″invoice written″ are objects that represent data. Statuses such as ″customer status″ and ″article status″ are also represented by data.

Since the entity-relationship (ER) model of Chen et al. was the most widespread designing method in the area of data modelling, for providing the data view with a description method. Today, the UML class diagram is in use for data modelling.. The *Function View* contains the description of the activities to be performed, the individual sub-functions, and relationships that exist between the functions. The *Organisation View* represents the organisational structure. This includes the relationships

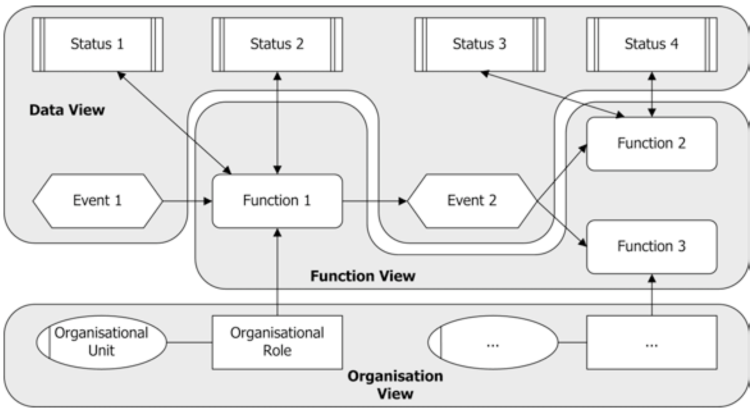


Figure 2.6: ARIS Views

between organisational units, between employees and organisational units, and between employees and their roles. The *Control View* links functions, organisation and data. It integrates the design results, which were initially developed separately for reasons of simplification. The functions, events, information resources, and organisation units are connected into a common context by the control flow. The resulting model is the EPC.

We have developed a metamodel for EPC which contains all views. The metamodel is shown in figure 2.7. An EPC consists of *functions*, *events*, *control flow connectors*, *logical operators*, and *additional process objects*. Each EPC consists of one or more *Functions* and two or more *Events*. An EPC starts and ends with an event and requires at least one function for describing a process. That means that a function has at least one successor and one predecessor. Both, functions and events can be (re)used in several EPCs. An event has four attributes: start, intermediate, end and trigger. Start, intermediate and end shows whether the event is at the beginning, middle or end of a process. Trigger demonstrates when an event triggers a logical operator.

A function can be either an *Elementary Function* or a *Complex Function*, and the latter is refined by at least one function. A function is connected with two *Control Flow Connectors* and has to fulfil at least one *Process Goal*. Furthermore a function may be connected with one or more *Additional Process Objects*. An event is connected with one or two control flow connectors, as an event starts and terminates the EPC. Control flows link events with functions, but also events or functions with *Logical*

Operators. A logical operator can be either an XOR, OR or AND. It is connected at least with three control flows, one or more incoming as well as outgoing connectors. An *Additional Process Object* may be assigned to one or more functions. A *Deliverable*, an *Information Object* and an *Organisational Structure* are called additional process objects. All three types of additional process objects may be assigned to one or more functions. The organisational structure can be an *Organisational Role* or an *Organisational Unit*, the latter is refined by one or more organisational roles. The organisational structure is connected with one or more Organisational Flow Connectors. The information object is connected with one or more *Data Flow Connectors* and the deliverable is connected with one or more *Input/Output Flow Connectors*.

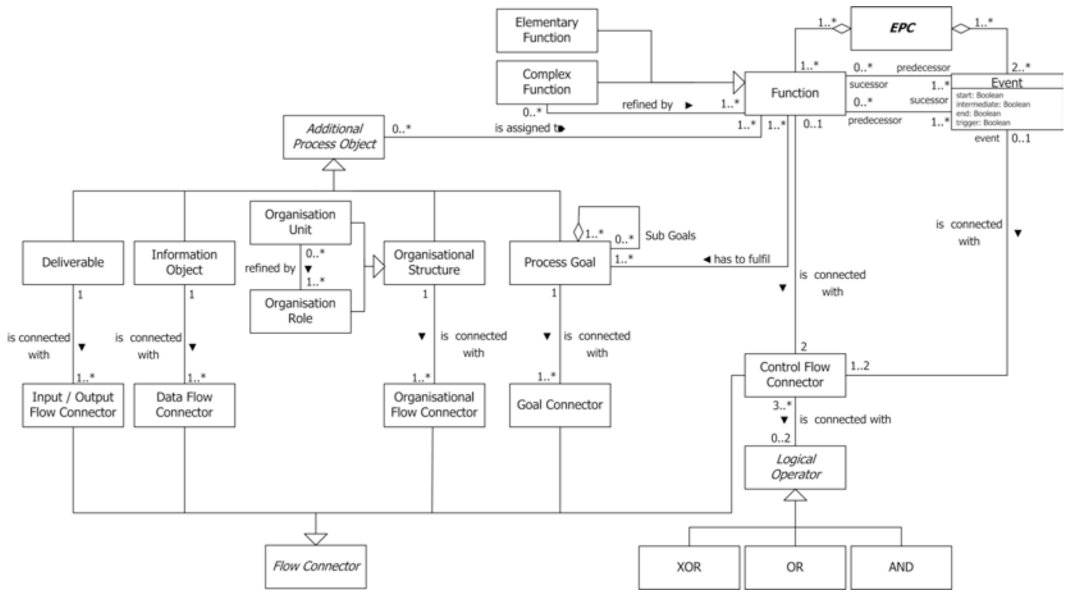


Figure 2.7: EPC metamodel

Example 2.2 Figure 2.8 shows the example business process of an EPC. As every EPC, the example starts and ends with an event. The EPC starts with the event *New Claim submitted*. The function *Record the Claim* starts after the first event. After the event *Claim recorded* the function *Calculate the Insurance Sum* begins. The organisational role Financial Expert is responsible for the functions *Record the Claim* and *Calculate the Insurance Sum*. After Calcualting the Insurance Sum, the path of the business process is split up into two alternative flows, depending if the insurance sum has a *Minor Amount* or a *Major Amount*. If the insurance sum has a minor amount, then the function *Contacting the Garage* starts. If the insurance sum has a major amount, then the functions *Contacting the Garage* and *Checking History of the Customer* starts concurrently. These two functions are connected with the next event *Results Collected* by an OR-Join. At that time the organisational role *Claim Administrator* is responsible for the business process. Due to the fact that either *Checking History of the Customer* or *Contacting the Garage* is executed, or that both functions are processed, the OR-Join is needed. If the results are collected, then the function *Examination of Results* starts processing, to decide if the claim is handled *Positive* or *Negative*. If the claim is handled positive, then the insurance company *Pays for the Damage*, otherwise not. In both situations the *Case is Closed*.

2.3.3 Business Process Modelling Notation (BPMN)

The BPMN was developed by the Business Process Management Initiative (BPMI) with the goal to provide a notation that is easily readable and understandable for all business users [OMG06a], who design, implement or monitor business processes including a transformation into an execution language, namely the Business Process Execution Language, (BPEL) [IBM03]. 2005 BPMI and OMG merged together, and BPMN is now maintained by the OMG.Thus the BPMN aims to bridge the gap between business process design and its implementation. The main concepts of BPMN are similar to UML 2 Activity Diagrams (AD) [OMG05c]. Also BPMN uses the concept of token. But in contrast to ADs, the BPMN has no specific metamodel, just a mapping to the Business Process Definition Metamodel [OMG07a]. BPDM provides a general process modeling metamodel that supports the BPMN notation.

We derived a simple MOF-based metamodel of BPMN based on the specification of BPMN [OMG06a] which is shown in figure 2.10. The metamodel shows all core elements [OMG06a] that are used to describe a business process with BPMN. The metamodel was developed according to the specification of BPMN. The BPMN metamodel consists of four different categories: Flow Objects, Connecting Objects, Swimlanes, and Artifacts. Figure 2.9 shows the graphical notation of the elements of BPMN.

The elements *Activity, Process, Sub-Process, Task* as well as *Events* and *Gateways* are *Flow Objects*, which define the behaviour of a business process. A process consists of one or more activities. The activity is the main part of a BPMN, and is specialised through sub-processes that consist of at least one task. An event is something that

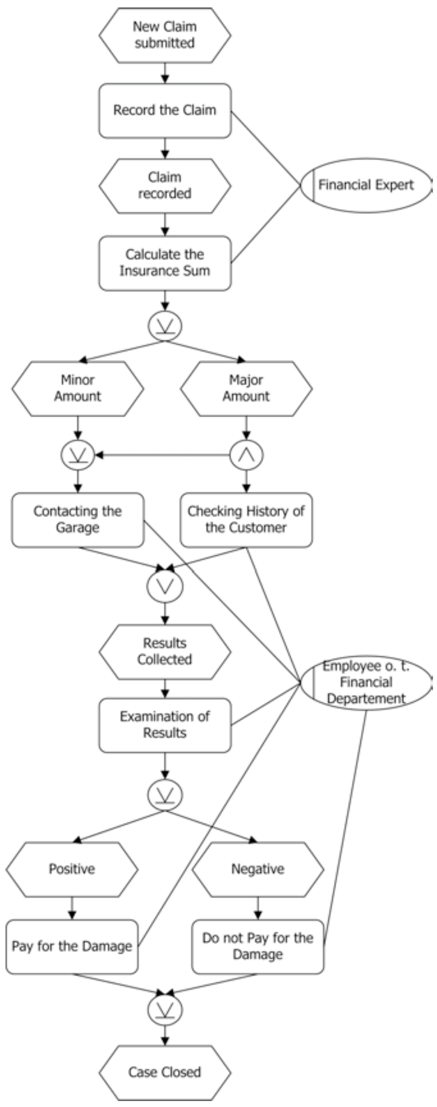


Figure 2.8: Example business process of an EPC

”happens” during the execution of a business process. There are three types of events, based on when they affect the flow: Start, Intermediate, and End. Also the *Time Event*, which can be a start or an intermediate event, is part of the metamodel because it is required for presenting the measure of time. It belongs to the complete set of elements, which displays a more extensive list of the business process concepts that could be depicted through BPMN. A gateway is used to control the divergence and convergence of a sequence flow. Markers within a gateway show the type of that flow object, it will determine between the logical operators XOR, OR, and AND, which stand for the Exclusive (XOR), Inclusive (OR) and Parallel (AND) gateway. Furthermore the type Complex indicates complex conditions and situations, e.g. that three paths out of five have to be chosen.

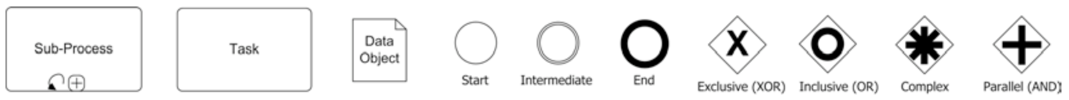


Figure 2.9: Elements of BPMN

The connecting objects *Sequence Flow, Message Flow* and *Association* describe the ways of connecting the flow objects to each other. A message flow can be connected to at most two activities, or occur between an activity and a pool, or between two pools to illustrate the exchange of messages. A sequence flow shows the order in which activities are performed in a process, and relates activities, gateways and events to each other. An association is used to associate information to activities, and associates a *Data Object* to a flow or connects it to an activity.

A *Pool* represents a participant in a process and belongs to the category of swimlanes, and it groups a set of activities for identifying activities that have some characteristic in common. A pool can be connected with other pools or activities by a message flow. A *Lane* is a sub-partition within a pool.

Example 2.3 Figure 2.11 describes the example business process with BPMN. The business process begins with a start event to execute the first task *Record the Claim*. *Calculate the Insurance Sum* and *Record the Claim* are part of the pool *Financial Claim Specialist*. After the task *Record the Claim* the pool *Claim Administrator* is responsible for the process. The exclusive gateway splits the flow, because the decision has to be made if the insurance sum has a minor amount or a major amount. if the insurance sum has a minor amount, then only the task *Contacting the Garage* is processed. But if the insurance sum has a major amount, then *Contacting the Garage* concurrently starts with the task *Checking History of the Customer*. An inclusive gateway combines the different pathes. The gateway conforms to the logical operator OR. After the task *Examination of Results* the decision has to be made if the insurance company *Pays for the Damage* or *Does Not Pay for the Damage*. After that decision the case is closed.

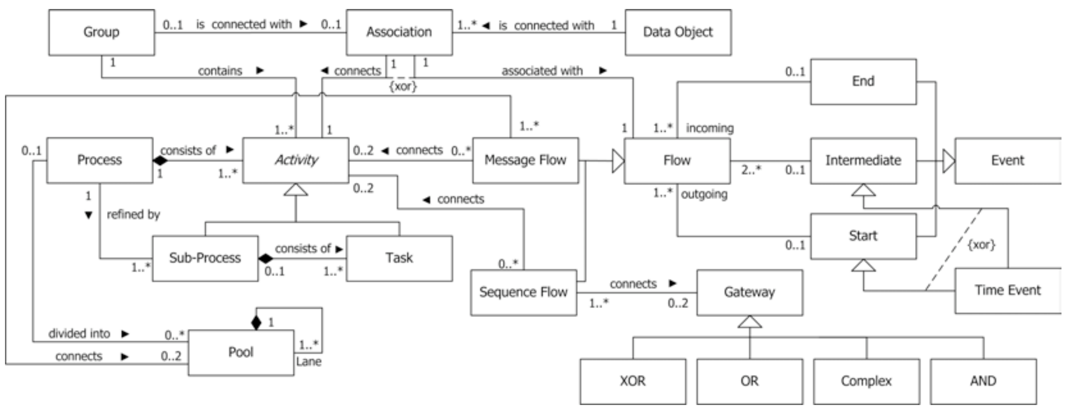


Figure 2.10: BPMN metamodel

2.3.4 Integrated DEFinition Method 3 (IDEF3)

IDEF3 [MMP^{+}95] is designed to model business processes and sequences of a system. It provides two views: the *process-centered* and the *object-centered view*. Figure 2.12 shows the main elements of IDEF3. The process-centered view models the process sequence with their temporal, causal, and logical relationships. The object-centered view describes objects and their changing states throughout a particular process.

The *process-centered strategy* mainly consists of *Units of Behaviour (UOB)*, *Links* and *Junctions*. The main elements are shown in figure 2.12. A UOB describes the activities or operations in a business process. If a UOB describes a complex activity of a business process, then it is possible to decompose it. It is graphically presented as a rectangle with current numbering and a reference number. A link represents the relationship between UOBs, and is graphically shown through an arrow. A junction illustrates the logical operators AND, OR, XOR. It is displayed as a little rectangle with with its boolean operator.

Furthermore it is possible to integrate *Referents* and *Notes* in an IDEF3 model. In IDEF3 a reference either refers to a previously defined UOB, or establish links between the process schematics and object schematics. A referent is either a *Call-and-Continue referent* or a *Call-and-Wait referent*. On the one hand side the IDEF3 element that makes the reference has to be initiated before a Call-and-Continue referent can progress to completion. On the other hand side a Call-and-Wait referent cannot start

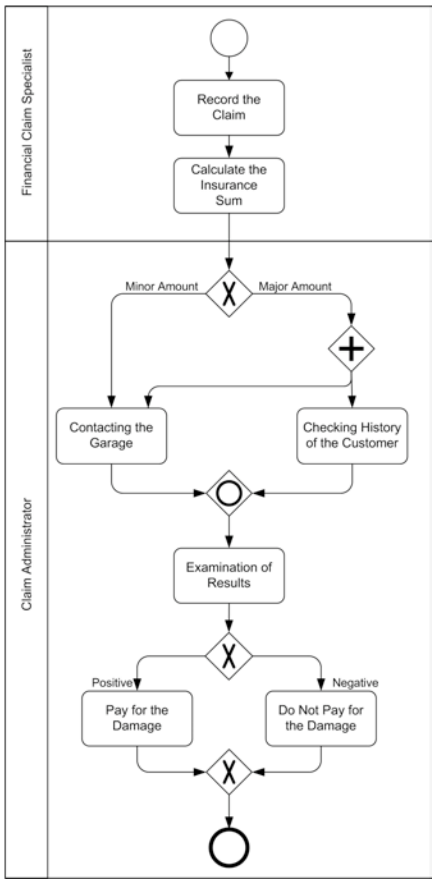


Figure 2.11: Example business process of a BPMN

until the IDEF3 element that references to that Call-and-Wait referent initiates and completes its tasks. A note allows to annotate additional information to a business process.

The three types of a link are the *Simple Precedence Link*, the *Constrained Precedence Link*, and the *Relational Link*. Precedence links express temporal precedence relations between UOBs. They are the most widely used link and are denoted by a solid arrow. *Constrained precendence links* add additional activation semantics to *simply precedence links*, that any instance of the source UOB must be followed by an instance of the destination UOB.

A junction splits or joins the pathes of an IDEF3 diagram. Multiple parallel subprocesses are indicated by the use of an *AND junction*. If a *synchronous AND junc-*

tion is used, then in the case of a fan-out AND junction the instances must all start simultaneously. In the case of a fan-in AND junction, the instances have to end simultaneously.

A fan-out *OR junction* indicates that there will be an instance of at least one of the UOBs. Similarly, a fan-out *XOR junction* indicates that there will be an instance of exactly one of the UOBs. If a *synchronous OR junction* is used, then those instances must all must start simultaneously. In the case of a non-synchronous OR junction an instance has not to wait for other instances to continue with further processing.

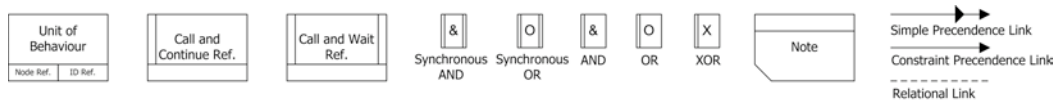


Figure 2.12: Symbols Used for IDEF3 Process Description Schematics

There are four different types of referents, namely *"UOB"*, *"Scenario"*, *"TS"* and *"Go-To"*. The type of a referent is labeled as a prefix, followed by the name of the referent. Referents also include a field to note a locator for declaring the referent type obviously.

The referent type "UOB" indicates that another instance of a previously defined UOB occurs at a specific point in the process. If this referent type is attached to an object state in an object schematic, it indicates that the referenced UOB sustains the object in the state.

The referent type "Scenario" specifies that the next happening in the process flow is an occurrence of an activation of the referenced Scenario. In that case all decompositions of the named Scenario are activated. If this referent type is attached to a transition arc in an object schematic, an activation of the referenced Scenario must start before the state transition is allowed.

The referent type "TS" has to be initiated during an activation of its associated UOB. They are connected through a simple connecting link. A Call-and-Continue "TS" referent attached to a transition arc between states indicates that the object must initiate a transition through the states of the referenced Transition Schematic before the state transition is allowed.

A "Go-To" referent type references another UOB. If a "Go-To" referent type is activated, then the business process continues with the referenced UOB. Go-to referents are always Call-and-Continue type referents.

The *object-centered strategy* consists of *Object States, Links, Relations* and *Junctions*, which are shown in figure 2.13. An object is of a certain kind, and is represented simply by a circle. The state of an object can be additionally annotated too. Relations describe the taxonomic relationship between objects. A *Transition* describes the change from an Object A to an Object B, which are connected through Links. A stronger connection between two objects are shown with a double headed arrow. Links and junctions have the same functionality like the process-centered strategy. Furthermore it is also possible to use for modelling referents and notes, like at the process-centered strategy. Referents are attached to transition links and objects at the object-centered strategy.

IDEF3 distinguishes between *firstorder objects* and *second orderobjects*. Individuals are referred as first-order objects. Properties and relations that hold among individuals are identifiable objects themselves. They are one level of abstraction above ordinary firstorder objects, because they are said to be of a higher logical type. Therefore they are classified as secondorder objects.

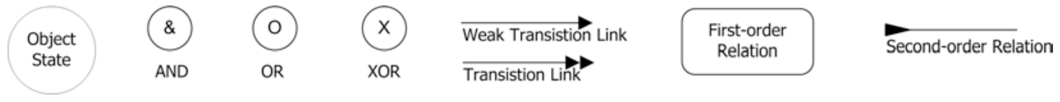


Figure 2.13: Symbols Used for IDEF3 Object Description Schematics

Example 2.4 Figure 2.14 describes the example business process in IDEF3 notation. Compared to BPMN or UML 2 AD, in IDEF3 it is not possible to show the organisational structure of a business process. The business process starts with the UOB *Record the Claim*. Afterwards the UOB *Calculate the Insurance Sum* begins. Now the flow is split up into several pathes. If the first path is chosen, then only the UOB *Contacting the Garage* is executed. If the other path is chosen, then *Contacting the Garage* as well as *Checking History of the Customer* are concurrently executed. An synchronous OR-join merges the pathes. The next UOB *Examination of Results* decides if the insurance company *Pays the Insurance Sum* or *Does Not Pay the Insurance Sum*. After that decision the case is closed.

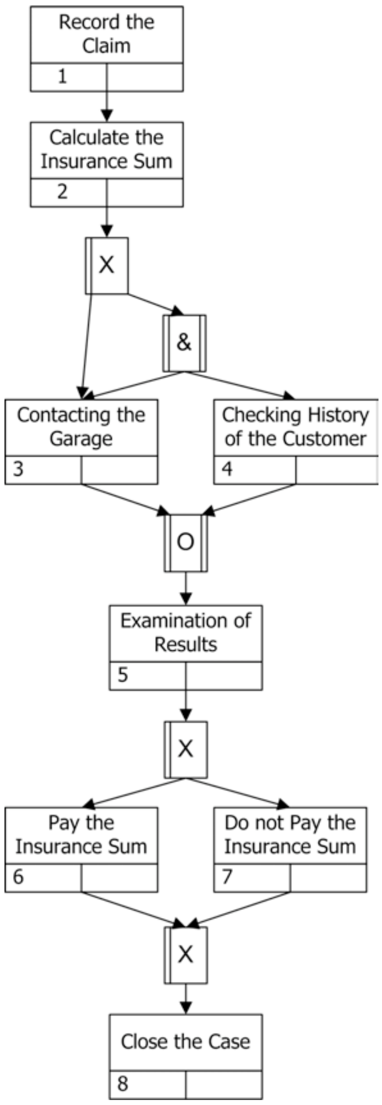


Figure 2.14: Example business process of an IDEF3

2.3.5 Petri Net

A Petri Net [Pet62] is designed for modelling, analysis and simulation of dynamic systems with concurrent and non-deterministic procedures. Petri Nets are utilised for modelling workflows.

Carl Adam Petri developed Petri Nets in his Ph.D. thesis ”Kommunikation mit

Automaten" [Pet62]. Originally, the work of Carl Adam Petri did not deal with business processes, but with deterministic distributed systems. Today many modelling techniques are based on Petri Nets, like the UML 2 AD. The petri nets were continously modified and adapted. Today three different forms of petri nets are distinguished: *condition/event net*, *place/transition net*, and *predicate/transition net*. Besides these three forms for example the *colored Petri Nets* [Jen92] and *relation Petri Nets* [Bau90] were developed. Every form of a Petri Net has its own rules for firing token. Furthermore we explain the condition/event net.

A Petri Net is a directed graph that mainly consists of four elements, namely *Places*, *Transition*, *Tokens* and *directed Arcs*. Transitions are interpreted as activities, actions or events which cause the change of state. Places represent possible states of the system. Transitions are graphically shown as rectangles, and states as circles. Tokens are placed within states, which is called marking. The state of a system is recorded through the positions of the different tokens in Petri Net. The directed arcs connect a transition with a place, or a place with a transition. It is not possible to connect places and transitions among each other. On the one hand side the places from which an arc runs to a transition are called the input places of the transition. On the other hand side the places to which arcs run from a transition are called the output places of the transition.

Every place has a specific weight, which symbolizes its capacity. If no weight is denoted, then the weight has the value one or unlimited. Every arc is assigned to a weight, which defines the consumption of tokens of an arc. If no weight is annotated to an arc, then the value of the weight is one. At the condition/event net the weight of places and arcs is always one.

A transition is enabled to fire, if two conditions are fulfilled. First, every input place must contain at least the value of tokens which are needed for a transition. Second, output arcs have to have at least enough capacity to accept all incoming tokens. When a transition fires, it uses the tokens from its input places, and performs processing tasks. Afterwards it places a specified number of tokens into each of its output places. Furthermore multiple transitions can be enabled at the same time.

Figure 2.15 shows a small example of the main elements of Petri Nets. The Petri Net consists of two places, and one transistion. When the first place fires, the token moves to the second place.

Example 2.5 Figure 2.16 describes an example petri net. For a better understand-

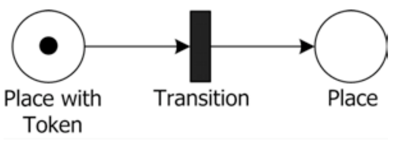


Figure 2.15: Small example of a Petri Net

ing textual labels are annotated to each place. Since Petri Nets are not able to show OR-Nodes, the business process is a little bit different then in the other languages. When the first transition fires, the token moves from *Record the Claim* to *Calculate the Insurance Sum*. The next transition T2 splits up the token either directly to *Contacting the Garage* or first to *Checking History of the Customer* and then to *Contacting the Garage*. Afterwards transition T4 fires the token to *Examination fo Results*. When transition T5 fires, the token moves to transition T6 either through the place *Pay the Insurance Sum* or *Do not Pay the Insurance Sum*. After that decision the case is closed in the last place.

2.3.6 Role Activity Diagram (RAD)

Role Activity Diagrams (RAD) were first described by Holt et al. [HRG83]. They are based on Petri Nets. The origin of the RAD [HRG83] lies in the modelling of coordination. Today, the RAD is used for modelling business processes [Mar95]. Figure 2.17 describes the graphical representation of the main elements of RAD.

RAD shows *Roles*, their *Activities* and *Interactions*, together with *External Events*. In RAD an external event, also called a trigger, initialises the begin of a business process. It is denoted as an arrow. A role is responsible for the performance of its activities. Roles are drawn as sets of boxes. If a role does not exist at the begin of a process, it has to be initiated. Cardinalities show the numerical relationship between role types. Interactions between roles are shown by a horizontal line linking two white boxes. An activity represents the task of a business process. It is shown as black boxes within a role. Activities are linked through vertical lines, which describe the different states of the role. A token shows the actual state of the process in RAD. Inverted triangles show the alternative choices in RAD. Parallel flows between activities are represented through normal triangles. For finishing a business process in RAD, the initiated role has to be terminated by the graphic representation of an ellipse.

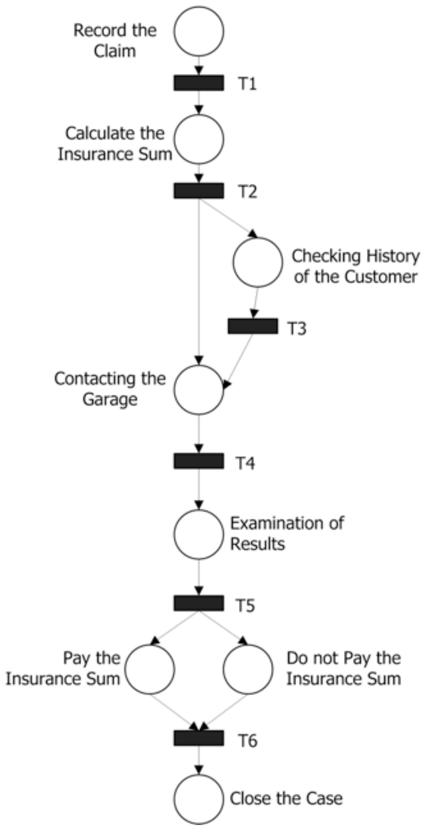


Figure 2.16: Example business process of a Petri Net

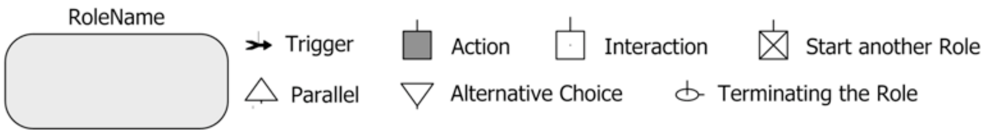


Figure 2.17: Elements of RAD

Example 2.6 The example business process is shown in figure 2.18. The business process consists of two roles, the *Financial Claim Specialist* and the *Claim Administrator*. The *Financial Claim Specialist* is initiated by the trigger *New Claim Received*. After the trigger the action *Record the Claim* starts processing. The *Financial Claim Specialist* starts by *Calculating the Insurance Sum* an interaction with the *Claim Administrator* to decide if the insurance sum has an minor amount or a major amount. If the insurance sum has a minor amount then the *Claim Administrator* starts the action

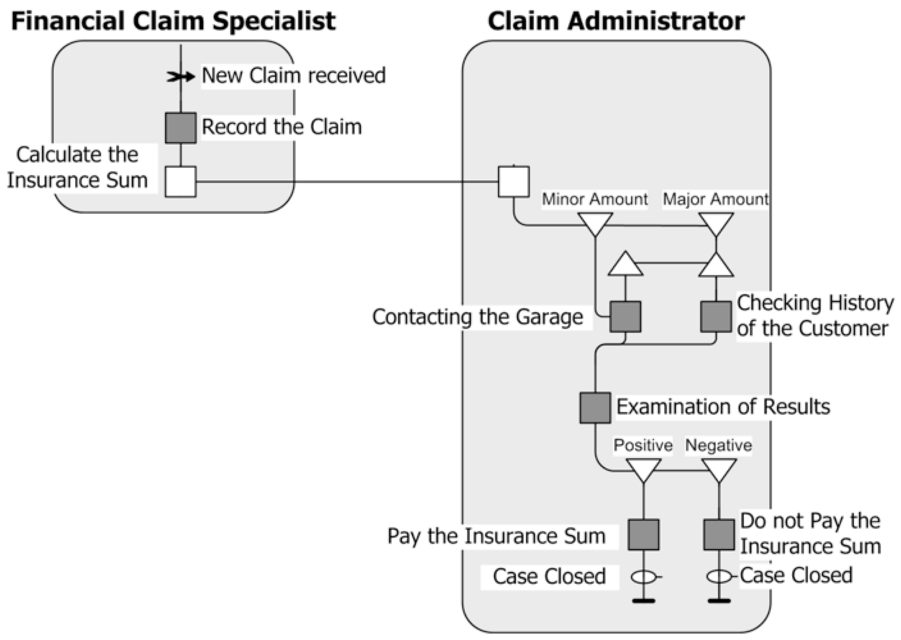


Figure 2.18: Example business process of a RAD

Contacting the Garage. If the insurance sum has a major amount, then the *Claim Administrator* concurrently initiates *Checking History of the Customer* and *Contacting the Garage*. After the action *Examination of Results* the *Claim Administrator* has to judge the claim positive or negative. If the decision is positive, then the *Claim Administrator Pays the Insurance Sum* to the customer. If the decision is negative, then the *Claim Administrator does not Pay the Insurance Sum* to the customer. After that decision in both cases the case is closed.

2.4 Evaluation

In this section, we evaluate six BPMLs based on the metamodel developed in section 2.2. The evaluation method is described in subsection 2.4.1, and the results are shown in subsection 2.4.2 in the tables 2.2, 2.3, 2.4 , 2.5, and 2.6. The rows represent the elements of the metamodel. The columns represent the different BPMLs.

Table 2.1: Meta-Model and Notation of BPMLs

BPML	AD	BPMN	EPC	IDEF3	Petri Nets	RAD
Metamodel	+	Mapping to BPDM	+	-	-	-
Notation	+	+	+	+	+	+

2.4.1 Evaluation Method

Because the majority of the BPMLs do not offer an explicit metamodel, we have not focused on the comparison of metamodel elements, but rather on notation elements and on concepts. Table 2.1 shows the BPMLs, which provide a metamodel and their own notation. The evaluation shows one assessment criterion with two possible symbols addressing one element of a BPML. The assessment criterion consists of two positions, seperated by a slash. The first position signals if a certain BPML offers a specific graphical notation element to explicitly symbolise a certain element of the generic meta-model. The second position shows, if the BPML provides a concept that somehow allows describing this meta-model element with a workaround.

The symbol ”+” characterises a success, otherwise it is denoted with a ”-”. In addition to the two symbols, the name of the concept representing the element is shown in table 2.1. For example, the UML AD does not offer a specific graphical notation element for a database table, but a DataStore Node could be utilised. The result of this evaluation is therefore ”-/+”.

2.4.2 Results of Evaluation

Since an accurate description is very often missing, it was difficult to evaluate the BPMLs. UML 2 AD, BPMN,BPDM and IDEF3 offer an official specification. On the one hand side sometimes elements have ambiguous meanings. For instance RAD describes the organisational as well as the informational perspective with one element. On the other hand side some BPMLs have complex definitions, while others are inaccurate and leave the usage of elements up to the interpretation of the user. For example it is impossible to show elements of the organisational perspective with Petri Nets.

At the beginning of the evaluation it was our original goal to compare the different BPMLs based on their metamodels with our generic metamodel. This was impossible because metamodels are not available for four out of six BPMLs like table 2.1 shows. Therefore we compared the BPMLs with our generic metamodel based on their elements and notation (see section 2.4.1).

Generally, the functional and the behavioural perspectives are very well represented in all BPMLs, while the organisational and informational perspectives are only partly supported.

The functional perspective is very well represented in all BPMLs (table 2.2). The languages differ by the fact, that they either describe the task of a business process with one single element, an activity, or with two explicit elements, a subprocess and an atomic activity.

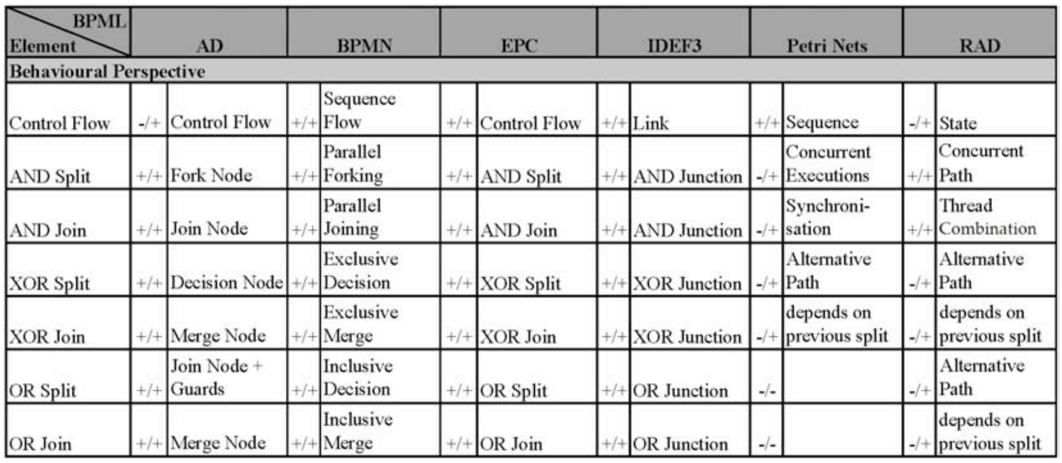
BPML / Element	AD		BPMN		EPC		IDEF3		Petri Nets		RAD	
Functional Perspective												
Activity	-/+		-/+		+/+	Function	+/+	Unit of Behaviour	-/-		+/+	Activity
SubProcess	+/+	Activity	+/+	Sub Process	-/+	Complex Function	-/+	Unit of Behaviour	-/+	Transition Hierarchy	-/+	Activity
Atomic Activity	+/+	Action	+/+	Task	-/+	Elementary Function	-/+	Unit of Behaviour	-/+	Transition	-/+	Activity

Table 2.2: Functional Perspective

Also the behavioral perspective is very well represented (table 2.3). Moreover the control flow as well as the control nodes are supported. The exceptions are petri nets, which do not support an OR node. Furthermore no language is able to present and N-out-of-M-Join, except the BPMN.

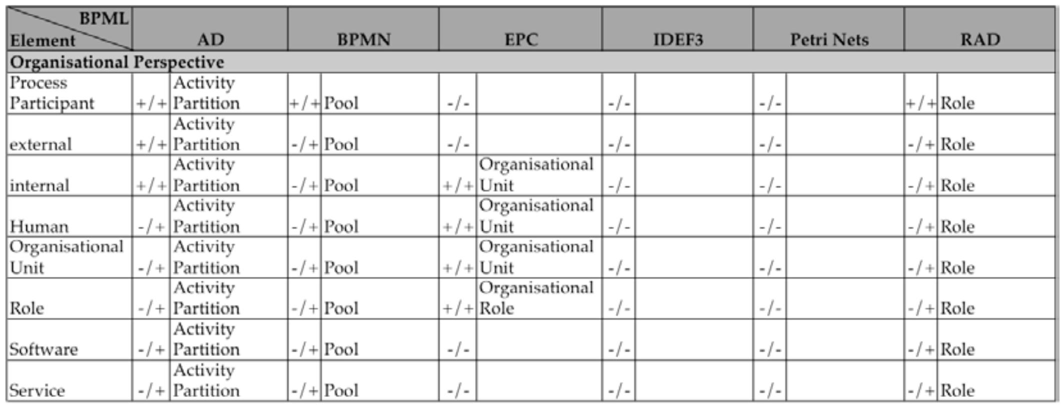
As table 2.4 demonstrates, the BPMLS partially provide the organisational perspective. Exceptions are IDEF 3 and Petri Nets, which have their origin in system and software engineering. All other BPMLs of this evaluation focus much more on the business process and include therefore a role concept. No BPML represents software in an explicit concept and only the AD explicitly shows whether a role belongs to the organisation or it is external. A lot of BPMLs utilise one concept to represent all types of process participants (e.g. AD, RAD, BPMN) and do not distinguish between the different types. This differentiation could be very helpful for BPMLs with a focus on process enactment.

The informational perspective is better developed for more recent BPMLs like AD, BPMN, and EPC (table 2.5). Only the EPC provides an explicit notation element



BPML / Element	AD		BPMN		EPC		IDEF3		Petri Nets		RAD	
Behavioural Perspective												
Control Flow	-/+	Control Flow	+/+	Sequence Flow	+/+	Control Flow	+/+	Link	+/+	Sequence	-/+	State
AND Split	+/+	Fork Node	+/+	Parallel Forking	+/+	AND Split	+/+	AND Junction	-/+	Concurrent Executions	+/+	Concurrent Path
AND Join	+/+	Join Node	+/+	Parallel Joining	+/+	AND Join	+/+	AND Junction	-/+	Synchroni-sation	+/+	Thread Combination
XOR Split	+/+	Decision Node	+/+	Exclusive Decision	+/+	XOR Split	+/+	XOR Junction	-/+	Alternative Path	-/+	Alternative Path
XOR Join	+/+	Merge Node	+/+	Exclusive Merge	+/+	XOR Join	+/+	XOR Junction	-/+	depends on previous split	-/+	depends on previous split
OR Split	+/+	Join Node + Guards	+/+	Inclusive Decision	+/+	OR Split	+/+	OR Junction	-/-		-/+	Alternative Path
OR Join	+/+	Merge Node	+/+	Inclusive Merge	+/+	OR Join	+/+	OR Junction	-/-		-/+	depends on previous split

Table 2.3: Behavioural Perspective

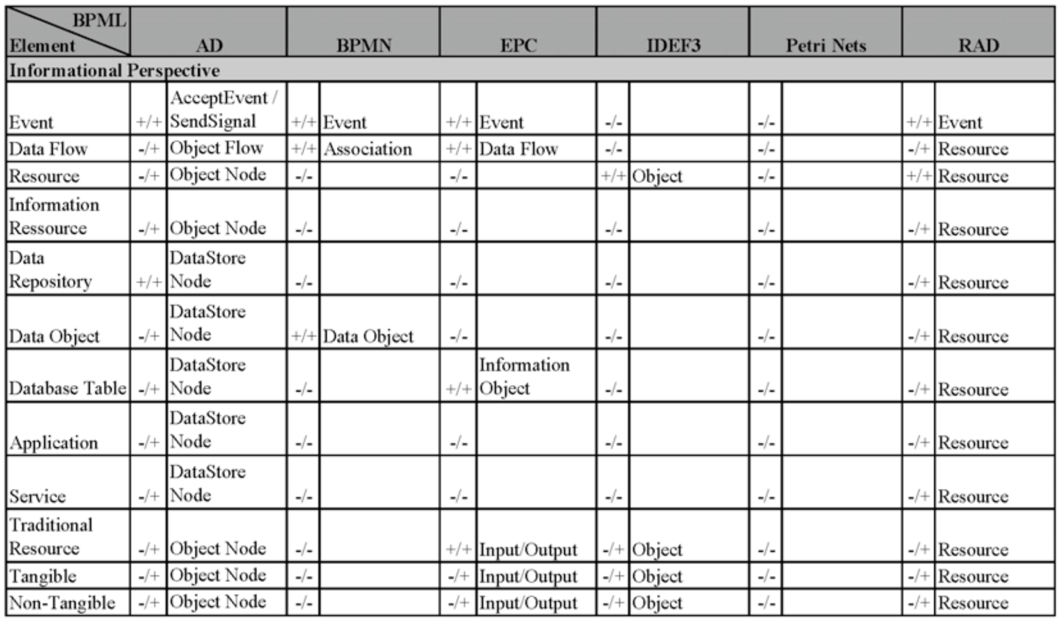


BPML / Element	AD		BPMN		EPC		IDEF3		Petri Nets		RAD	
Organisational Perspective												
Process Participant	+/+	Activity Partition	+/+	Pool	-/-		-/-		-/-		+/+	Role
external	+/+	Activity Partition	-/+	Pool	-/-		-/-		-/-		-/+	Role
internal	+/+	Activity Partition	-/+	Pool	+/+	Organisational Unit	-/-		-/-		-/+	Role
Human	-/+	Activity Partition	-/+	Pool	+/+	Organisational Unit	-/-		-/-		-/+	Role
Organisational Unit	-/+	Activity Partition	-/+	Pool	+/+	Organisational Unit	-/-		-/-		-/+	Role
Role	-/+	Activity Partition	-/+	Pool	+/+	Organisational Role	-/-		-/-		-/+	Role
Software	-/+	Activity Partition	-/+	Pool	-/-		-/-		-/-		-/+	Role
Service	-/+	Activity Partition	-/+	Pool	-/-		-/-		-/-		-/+	Role

Table 2.4: Organisational Perspective

for traditional resources and is therefore well suited for process analysis.

As table 2.6 shows, the main lack of the BPMLs is that the business process context perspective is not explicitly supported at all. Of course it is possible to present a Business Process, but it cannot be distinguished between a Core-, Support-, or Management Process. If the BPML allows to show a process participant, then it is also possible to present a Customer. If the BPMLs have an element to present some kind of a resource, then the languages are able to represent a deliverable. However,

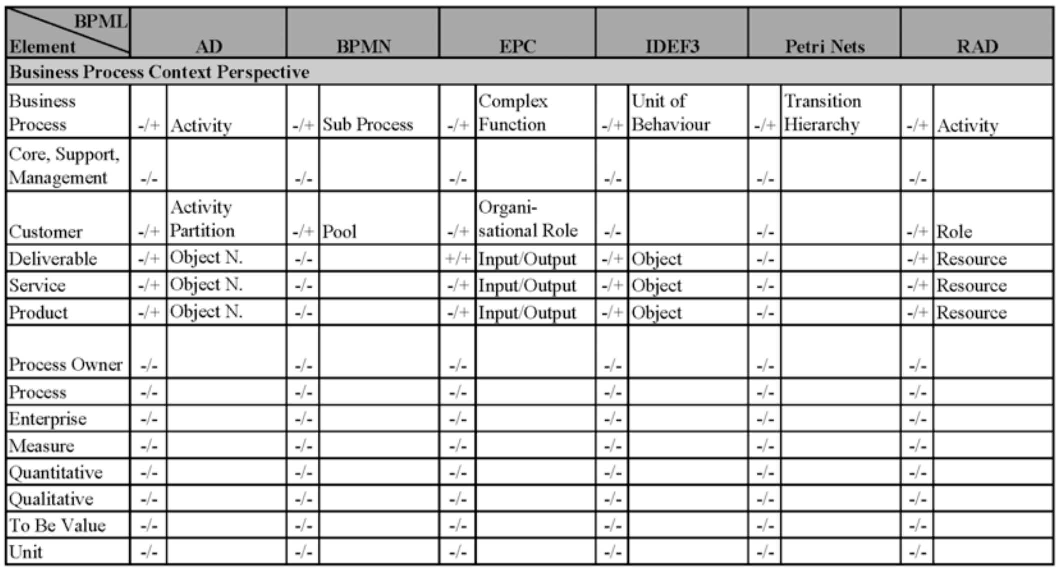


BPML / Element	AD		BPMN		EPC		IDEF3		Petri Nets		RAD	
Informational Perspective												
Event	+/+	AcceptEvent / SendSignal	+/+	Event	+/+	Event	-/-		-/-		+/+	Event
Data Flow	-/+	Object Flow	+/+	Association	+/+	Data Flow	-/-		-/-		-/+	Resource
Resource	-/+	Object Node	-/-		-/-		+/+	Object	-/-		+/+	Resource
Information Ressource	-/+	Object Node	-/-		-/-		-/-		-/-		-/+	Resource
Data Repository	+/+	DataStore Node	-/-		-/-		-/-		-/-		-/+	Resource
Data Object	-/+	DataStore Node	+/+	Data Object	-/-		-/-		-/-		-/+	Resource
Database Table	-/+	DataStore Node	-/-		+/+	Information Object	-/-		-/-		-/+	Resource
Application	-/+	DataStore Node	-/-		-/-		-/-		-/-		-/+	Resource
Service	-/+	DataStore Node	-/-		-/-		-/-		-/-		-/+	Resource
Traditional Resource	-/+	Object Node	-/-		+/+	Input/Output	-/+	Object	-/-		-/+	Resource
Tangible	-/+	Object Node	-/-		-/+	Input/Output	-/+	Object	-/-		-/+	Resource
Non-Tangible	-/+	Object Node	-/-		-/+	Input/Output	-/+	Object	-/-		-/+	Resource

Table 2.5: Informational Perspective

not any BPML is able to show the goals of a business process, or how they could be measured.

To improve the flexibility of business processes, there is a need to reduce the time between process modelling and the transformation into executable code. Therefore, future BPMLs must provide execution languages and in turn, offer more explicit notation elements on all perspectives.



BPML / Element	AD		BPMN		EPC		IDEF3		Petri Nets		RAD	
Business Process Context Perspective												
Business Process	-/+	Activity	-/+	Sub Process	-/+	Complex Function	-/+	Unit of Behaviour	-/+	Transition Hierarchy	-/+	Activity
Core, Support, Management	-/-		-/-		-/-		-/-		-/-		-/-	
Customer	-/+	Activity Partition	-/+	Pool	-/+	Organi-sational Role	-/-		-/-		-/+	Role
Deliverable	-/+	Object N.	-/-		+/+	Input/Output	-/+	Object	-/-		-/+	Resource
Service	-/+	Object N.	-/-		-/+	Input/Output	-/+	Object	-/-		-/+	Resource
Product	-/+	Object N.	-/-		-/+	Input/Output	-/+	Object	-/-		-/+	Resource
Process Owner	-/-		-/-		-/-		-/-		-/-		-/-	
Process	-/-		-/-		-/-		-/-		-/-		-/-	
Enterprise	-/-		-/-		-/-		-/-		-/-		-/-	
Measure	-/-		-/-		-/-		-/-		-/-		-/-	
Quantitative	-/-		-/-		-/-		-/-		-/-		-/-	
Qualitative	-/-		-/-		-/-		-/-		-/-		-/-	
To Be Value	-/-		-/-		-/-		-/-		-/-		-/-	
Unit	-/-		-/-		-/-		-/-		-/-		-/-	

Table 2.6: Business Process Context Perspective

3 Extending Business Process Modelling Languages with Performance Measures and Goals

Contents

3.1 Introduction

Business process performance measurement is an important topic in research and industry [Cas05]. In general the main focus of conceptual business process modelling languages (BPMLs) is organizing the flow with its tasks of a business process, but not the explicit modeling of process goals and their measures (see section 2.4) of a language. The UML 2 Activity Diagram, the Event-Driven Process Chain (EPC) as well as the Business Process Modeling Notation (BPMN) are designed for modelling business processes, but do not yet include any means for modelling process goals and their measures. To solve this gap, we have to extend these BPMLs with business process goals and performance measures. On the one hand side, UML 2 offers its own mechanism, the so called light-weight extension mechanism "UML Profile" to extend its metamodel. On the other hand side, EPC and BPMN do not have an extension mechanism. EPC has no metamodel which shows all views in one model, and BPMN is mapped to a general metamodel (see section 2.3 for more details). The goal of this section is to address these limitations by

- enhancing the expressiveness of EPC and BPMN by deriving metamodels for both, and by
- extending the metamodels of UML 2 AD, EPC as well as BPMN with business process goals and performance measures to make them conceptually visible.

Activity diagrams are a part of the behavioural models of UML 2 [OMG05c] and are used for modelling business processes as well as control flows in software as described before. Activity diagrams neither have quality nor quantity based elements to measure the performance of a business process. For instance, the modeller of a process has no concepts to express a maximum time limit for processing of a specific action - the basic element of activity diagrams - or a group of actions. UML profiles are an extension mechanism for building UML models for particular domains or purposes [OMG05c].

EPCs have become widely-used for business process modelling in continental Europe, in countries where SAP is a leading Enterprise Resource Planning (ERP) system. EPCs are inspired from Petri nets, incorporate role concepts and data models like ER models or UML class diagrams. The BPMN is wide spread in the US and in countries where US companies dominate the ERP system market.

BPMN was developed by the Business Process Management Initiative (BPMI) with the goal to provide a notation that is easily readable and understandable for all business users [OMG06a], who design, implement or monitor business processes. Thus the BPMN aims to bridge the gap between business process design and its implementation. The language is now being controlled by the Object Management Group since the two organizations merged in 2005.

Since UML 2 AD, EPC as well as BPMN are not able to represent business process goals and their measures, we have to extend these three languages with these concepts. For that challenge different mechanisms will be used. At UML 2 Activity Diagrams a UML profile is created, and at the EPC we introduce a new view, as well as at BPMN we establish a new category.

The Unified Modeling Language provides three different extension mechanisms that maybe used in conjunction with the Activity Diagram. The heavy-weight extension offers the possibility to extend and adapt the metamodel of UML, whereas with the light-weight extension it is only possible to extend but not to modify the metamodel. The latter mechanism is called a *profile*. It is also possible to create a new metamodel and use it within the UML-framework. We use the light weight extension to extend the UML 2 AD for providing the interoperability of tool support, since almost all newer UML modelling tools support UML profiles.

In contrast to UML 2 AD, BPMN has no specific metamodel, just a mapping to the Business Process Definition Metamodel [OMG07a]. BPDM provides a general process modeling metamodel that supports the BPMN notation. As explained in subsection 2.3 the EPC consists of different views. The EPC provides for each view a metamodel, but not an integrated metamodel that connects the related views together.

We derive a metamodel for the EPC and the BPMN based on the Meta-Object Facility (MOF), the OMG's meta-metamodel [OMG05a]. We extend the metamodels of BPMN, EPC and UML Activity Diagram with business process goals and performance measures, and thus, provide the following contributions:

- We integrate business process goals intro three well-known BPMLs. We want to provide a concrete syntax for the different goals of a business process, since the goals of a business process which are already available in process theory.
- Performance measures quantify business process goals, and thus help to evaluate the process design and the operating process. The extended UML 2 AD, EPC and BPMN conceptually visualize the evaluation criteria for a business process.

The remainder of this section is organized as follows:

- Subsection 3.2 describes the role of goals and performance measures in the business process.
- Subsection 3.3 presents a UML 2 profile for integrating business process goals and performance measures into UML 2 activity diagrams. The profile provides an explicit illustration of the performance measures time, cost, and quality. The UML profile and its mapping is tested by an example business process.
- Subsection 3.4 and 3.5 presents the metamodels with its extension to integrate business process goals and performance measures into EPC and BPMN. The extension of both languages provides an explicit illustration of the goals a business process must achieve. Furthermore the performance measures time, cost, and quality are integrated in EPC and BPMN, because without measur-

ing the process goals it is not possible to assess wether a goal is fulfilled or not.

- The extensions of BPMN and EPC are tested with an example business process shown in subsection 3.4 and 3.5.

3.2 The Role of Goals and Measures in the Business Process

With business process reengineering Davenport, Hammer and Champy created a new discipline at the beginning of the 1990ies and provided the business motivation for business process modelling. So far, in the business process modelling community attention has only been given to the modelling of certain aspects of processes (e.g. roles, activities, interactions). These theoretical aspects are captured in several business process modelling languages (BPMLs), for example, in the Business Process Modelling Notation [OMG06a], the Event-driven Process Chain [Sch99], the UML 2 Activity Diagram [OMG05c], etc. In 1997 Kueng and Kawalek argued already that little attention is paid to the value of making goals explicit [KK97]. Today there are currently quite a lot of conceptual BPMLs available, but they still do not provide modelling technics for business process goals and performance measures (see section 2.4).

A business process is defined as a "group of tasks that together create a result of value to a customer" [Ham96]. "Its purpose is to offer each customer the right product or service, e.g., the right deliverable, with a high degree of performance measured against cost, longevity, service and quality" [JEJ94]. Thus, while they lack the visibility in conceptual BPMLs, process goals and performance measures are available in process theory.

According to [KK97] the modelling of goals is a critical step in the creation of useful process models, for the following reasons:

- We need to be able to state what we want to achieve so that we are then able to define the necessary activities which a business process should encompass (i.e., goals are used to structure the design).
- A clear understanding of goals is essential in the management of selecting the best design alternative (i.e., goals are used to evaluate the design).
- A clear expression of goals makes it easier to comprehend the organisational changes that must accompany a business process redesign (i.e., goals help the

modeller to better understand the broader implication of design, beyond those of the business process itself).

For all the reasons described above, we capture goals and represent them graphically in conceptual BPMLs, namely UML 2 AD, BPMN and EPC. Furthermore, Kueng and Kawalek in [KK97] recommend defining to which extent the process goals are fulfilled. The achievement of goals must be measured either by qualitative or quantitative measures. Measures aim at reaching a to-be-value, also called target value. They are very important for business process improvement. Harrington stated "Measurements are the key. If you cannot measure it, you cannot control it. If you cannot control it, you cannot manage it. If you cannot manage it, you cannot improve it." [Har91]. In order to support all stages of Harrington's statement, we need to integrate performance measures into conceptual BPMLs.

As a first step according to the missing concepts found out in the evaluation in section 2, we capture goals as well as measures and represent them graphically in conceptual BPMLs.

The metamodel of the UML 2 AD, EPC and the BPMN will be extended by a small generic metamodel of goals and performance measures shown in Figure 3.1. It contains two core concepts, namely *Measure* and *Process Goal*. While these two concepts do not appear as notation elements in BPMN as well as UML 2 AD, the process goal is a part of EPC. Often it does not appear in the graphical notation of a business process modelled with EPCs, and there are no measures available for quantifying a goal.

A *process goal* describes the specific intension of a business process and is quantified by at least one *measure*. Furthermore the *goal* can be refined by one or more *sub goals*. A *measure* is an abstract metaclass, and can be classified and implemented as *Quality, Cost* or *Cycle Time*. A *measure* is responsible for the concrete quantification of different *goals* as well as for measuring the performance of a business process.

Quality has the aim to measure the quality of a business process, which can be expressed e.g., by a low number of complaints or a high customer satisfaction, described in Fig. 3.1 by the attributes *maxComplaints* as well as *avgComplaints*. The attribute *maxComplaints* shows the total number of complaints, and the attribute *avgComplaints* shows the average allowed number of complaints measured for instance during the time period of a month.

Cost represents the expenses a business process requires for instance for its exe-

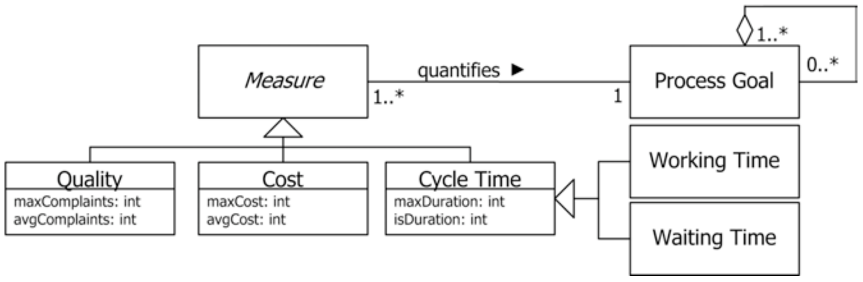


Figure 3.1: Generic metamodel of goals and performance measures

cution. Its attributes *maxCost* and *avgCost* are necessary for comparing the total and monthly average cost of a certain process. The performance measures of *quality* and *cost* are in contrast to the measures of the *cycle time* often more focused on the type level of a process, since the required data is often not available on instance level.

The measure *cycle time* presents a time based measure and defines the processing duration of a business process instance, or part of it. *Cycle Time* can be specialised as *Working Time* or *Waiting Time*. *Working time* presents the actual time a business process instance is being executed by a role. *Waiting time* shows the time the process instance is waiting for further processing. Moreover, *cycle time* has two attributes *maxDuration* and *isDuration* for representing the target value and the actual value of the process duration or a part of it.

3.3 UML 2 Profile

In this section the extended metamodel for activity diagrams for the UML 2 profile with business process goals and performance measures will be described. Activity diagrams are a part of the behavioural set of UML 2 diagrams. They are used for modelling business processes and control flows. An activity diagram specifies the control and data flow between different tasks, called actions, which are essential for the realisation of an activity. The activity diagram currently does not allow to represent business process goals and performance measures. Thus, it is not possible to show, e.g., time restrictions of the business process, its cost or quality requirements. UML offers a possibility to extend and adapt its metamodel to a specific area of application through the creation of profiles. This mechanism is called a light-weight extension. UML profiles are UML packages of a stereotype ≪profile≫. A *profile* extend a metamodel or another profile [OMG05c] while preserving the syntax and semantic of existing UML elements. It adds elements which extend existing metaclasses. UML profiles consist of stereotypes, constraints and tagged values.

A *stereotype* is a model element defined by its name and by the base class(es) to which it is assigned. Base classes usually are metaclasses from the UML metamodel, for instance the metaclass *Class*, but can also be stereotypes from another profile. A stereotype can have its own notation, e.g. a special icon.

Constraints are applied to stereotypes in order to indicate restrictions. They specify pre- or post conditions, invariants, etc., and must comply with the restrictions of the base class [OMG05c]. Constraints can be expressed in any language, such as programming languages or natural language. We use the Object Constraint Language (OCL) [OMG05b] in our profile, as it is more precise than natural language or pseudocode, and widely used in UML profiles.

Tagged values are additional meta-attributes assigned to a stereotype, specified as name-value pairs. They have a name and a type and can be used to attach arbitrary information to model elements.

Figure 3.2 illustrates a section of the UML metamodel for activity diagrams and its extension with stereotypes for representing business process goals and their performance measures. The UML profile consists of four different stereotypes, namely ≪Process Goal≫, ≪Measure≫, ≪Alert≫ and ≪Organisational Structure≫. The stereotype ≪Process Goal≫ describes the specific intension of a business process and is quantified by at least one ≪Measure≫. It extends the metaclass *Activity*, meaning that a ≪Process Goal≫ is described at activity level.

The stereotype ≪Measure≫ can be classified and implemented as ≪Quality≫, ≪Cost≫ and ≪Cycle Time≫ and extends the metaclasses *Activity Partition, Structured Activity Node*, and *Control Flow*. This means that the stereotype ≪Measure≫ can be described in three different ways. It is the modeller's role to choose the most suitable way to best describe a measure for a certain purpose, or for a user or user group. The modeller has the choice to go for one, or a combination of two or all three possibilities.

A measure positioned in an activity partition quantifies the section of the process that is covered by the role. According to the OMG [OMG05c], an activity partition identifies actions that have some characteristics in common. For example, if activity nodes have to be performed within a specific period of time, then they are grouped within an activity partition labelled with the stereotype ≪Cycle Time≫. It is also possible to nest the stereotypes. A stereotyped structured activity node labelled with ≪Working Time≫ can be nested in an activity partition, e.g., extending ≪Cycle Time≫.

A structured activity node has the function to group elements of an activity, in order to structure the activity [OMG05c]. A measure located in a structured activity node quantifies the section of the process that is covered. For instance, a structured activity node that is extended with the stereotype ≪Cycle Time≫ has to finish the processing of its actions within a certain period of time.

A measure based on the control flow quantifies the cycle time, cost or quality between two actions. The OMG [OMG05c] defines a control flow as an edge that starts an activity node after the previous one is finished. For example, a control flow that is extended with the stereotype ≪Cycle Time≫ and connects two activity nodes, means that the stereotype measures a period of time the token requires from the activity node at the beginning of the edge to the activity node at the end of the edge.

The stereotype ≪Measure≫ is responsible for the concrete quantification of different goals as well as for measuring the performance of a business process. If the process is not performed according to the ≪Measure≫, an ≪Alert≫ is triggered.

The stereotypes ≪Quality≫, ≪Cost≫ and ≪Cycle Time≫ add more detail to the stereotype ≪Measure≫ and classify it. The stereotype ≪Quality≫ has the aim to measure the quality of a business process, which can be expressed e.g., by a low number of complaints or a high customer satisfaction.

The stereotype ≪Cost≫ represents the expenses associated to a business process e.g., the cost of its execution. Its tagged values are necessary to compute e.g. average values like the total and monthly average cost of a certain process. The performance measures of ≪Quality≫ and ≪Cost≫ are in contrast to the measures of the ≪Cycle Time≫ often more focused on the type level of a process, as the required data is often not available on instance level.

The stereotype ≪Cycle Time≫ presents a time based measure and defines the duration a business process instance, or part of it requires from the beginning until the end. The stereotype ≪Cycle Time≫ can be specialised as ≪Working Time≫ or ≪Waiting Time≫. ≪Working Time≫ presents the actual time a business process

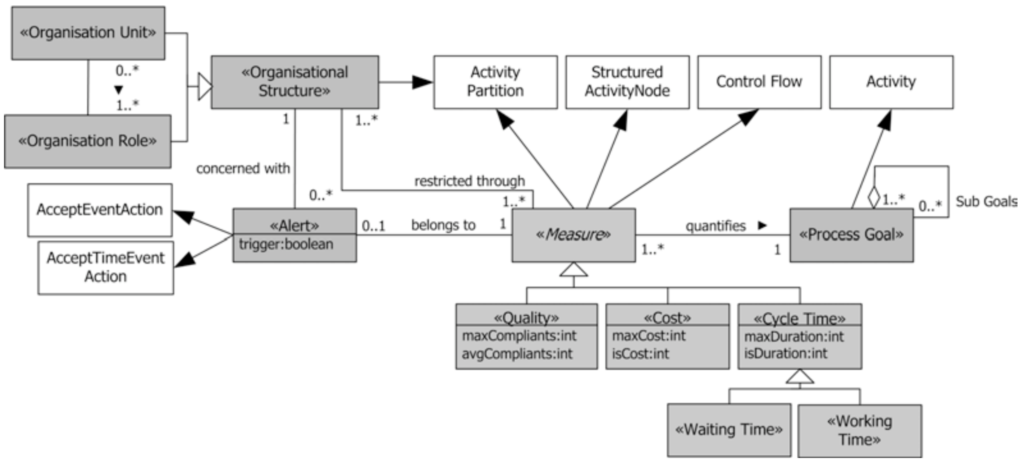


Figure 3.2: Extended metamodel of the activity diagram for the UML 2 profile with business process goals and performance measures

instance is being executed by a role. ≪Waiting Time≫ shows the time the process instance is waiting for further processing. Moreover, ≪Cycle Time≫ has two tagged values, for representing the target value and the actual value of the process duration or a part of it which is computed by an operation of the stereotype.

The stereotype ≪Organisational Structure≫ describes the different roles that perform certain actions within an activity diagram, namely ≪Organisational Unit≫ and ≪Organisational Role≫. If an action or a group of actions is not executed within its performance measures, then the stereotype ≪Alert≫ will be triggered. Furthermore an ≪Organisational Structure≫ is concerned with at least one ≪Alert≫.

The ≪Alert≫ stereotype has two metaclasses, from which it is derived, one for time based measures, namely AcceptTimeEventAction, and one for non-time based measures, namely AcceptEventAction. An ≪Alert≫ belongs to exactly one ≪Measure≫ and one ≪Organisational Structure≫.

3.3.1 Constraints

Table 3.1 and table 3.2 show the different constraints in OCL with explanations in natural language for the stereotype ≪Measure≫ and ≪Alert≫ that are necessary for performance measurement.

Table 3.1: OCL Constraints of ≪Measure≫

Name	**Measure**
Base class	Activity Partition OR Structural Activity Node OR Control Flow
Description	A measure is an abstract metaclass, and can be classified and implemented as Quality, Cost or Cycle Time. Furthermore it is responsible for the concrete quantification of different goals as well as for measuring the performance of a business process.
Constraints	When a measure is a cycle time based measure, then the possible alert has the type of an AcceptTimeEventAction. Else the possible alert has the type of an AcceptEventAction. `context Measure inv:` `if Measure.oclIsKindOf(CycleTime) then` `Alert.oclIsKindOf(AcceptTimeEventAction)` `else Alert.oclIsKindOf(AcceptEventAction)` `endif`

The stereotype ≪Measure≫ and its restrictions is shown in table 3.1.
If ≪Measure≫ is a cycle based measure, then the metaclass of ≪Alert≫ has to be an *AcceptTimeEventAction*. Otherwise the metaclass for ≪Alert≫ has to be an *AcceptEventAction*.

Table 3.2 explains the stereotype ≪Alert≫. An ≪Alert≫ has to be triggered when three different causes happen independent from each other. First, if the time duration is higher then the given value of a process. Second, if the average cost is higher then the maximum cost. Third, if the average number of complaints is higher then the constant number of complaints.

3.3.2 Applying the UML 2 Profile to an Example Business Process

We demonstrate the practical applicability of the extension of UML 2 Activity Diagrams with business process goals and performance measures in Figure 3.3 with the example business process of an insurance company: the Processing of Automobile Claims business process (Fig. 3.3). At first the business process of the insurance company will be described.

The overall goal of the processing of automobile claims business process is to fulfil *high customer satisfaction, short process duration* and *low processing costs*. The business process has three process participants, *Financial Claim Specialist*, *Claim Administrator*,

Table 3.2: OCL Constraints of ≪Alert≫

Name	**Alert**
Base class	AcceptEventAction OR AcceptTimeEventAction
Description	An Alert is triggered, if an action or a group of actions is not executed within its performance measures.

Constraints	If the actual value of the duration is higher then the given value of the duration, an alert will be generated.
	`context Alert inv:`
	`if cycleTime.isDuration > cycleTime.maxDuration then)`
	`Alert.trigger = true else Alert.trigger = false`
	`endif`
	
	If the average cost is higher then the maximum cost, then an alert will be generated.
	`context Alert inv:`
	`if Cost.allInstances()->forAll(avgCost > constantCost then)`
	`Alert.trigger = true else Alert.trigger = false`
	`endif`
	
	If the average number of complaints is higher then the constant number of compliants, an alert will be generated.
	`context Alert inv:`
	`if Quality.allInstances()->forAll(avgCompliants > constantCompliants) then`
	`Alert.trigger=true else Alert.trigger=false`
	`endif`

and *Claim Manager*. At the beginning of the process the *Financial Claim Specialist* is responsible for *Record the Claim* and *Calculate the Insurance Sum*. *Financial Claim Specialist* has to execute these two business process tasks within one day.

After a Waiting Time of two days maximum, the *Claim Administrator* has to follow up with the process. If the insurance sum has a major amount, then the claim administrator has to do *Check History of the Customer* in the other case no action is required. After starting to *Contact the Garage* for the reparation, the *Examination of*

Results has to begin. If the examination is positive, then the insurance has to *Pay for the Damage*, and the case is closed.

The process has to fulfill three performance measures, namely *Cost, Cycle Time,* and *Quality*. The average processing costs per month have to be 15 dollar maximum and the number of complaints should not exceed five percent. In our example, if the *Cycle Time* exceeds four days, then the *Claim Manager* receives an alert, and gets a report about that specific case.

We refine the activity diagram by including a set of stereotypes, based on the various types of actions specified in the metamodels of actions in the UML superstructure in chapter 11 [OMG05c]. Furthermore the work of Bordbar et. al [BS04] serves as a foundation for the creation of the stereotypes. Figure 3.3 shows that a business process based on the UML 2 profile can be grasped at a glance. The extensions of the UML activity diagram better illustrate the requirements of a certain business process and enhance the expressiveness of a model.

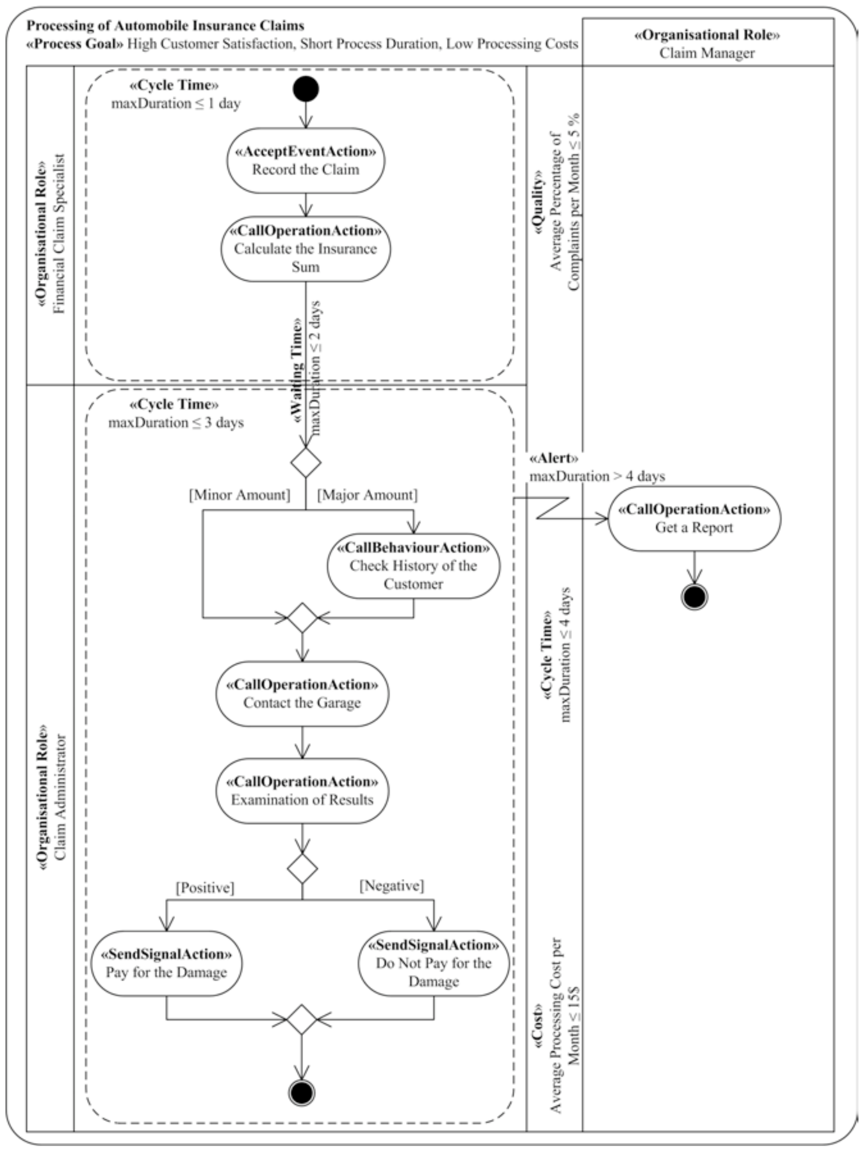

Processing of Automobile Insurance Claims
«Process Goal» High Customer Satisfaction, Short Process Duration, Low Processing Costs
«Organisational Role» Claim Manager
«Organisational Role» Financial Claim Specialist
«Cycle Time» maxDuration ≤ 1 day
«AcceptEventAction» Record the Claim
«CallOperationAction» Calculate the Insurance Sum
«Waiting Time» maxDuration ≤ 2 days
«Quality» Average Percentage of Complaints per Month ≤ 5 %
«Organisational Role» Claim Administrator
«Cycle Time» maxDuration ≤ 3 days
[Minor Amount]
[Major Amount]
«CallBehaviourAction» Check History of the Customer
«Alert» maxDuration > 4 days
«CallOperationAction» Get a Report
«CallOperationAction» Contact the Garage
«Cycle Time» maxDuration ≤ 4 days
«CallOperationAction» Examination of Results
[Positive]
[Negative]
«SendSignalAction» Pay for the Damage
«SendSignalAction» Do Not Pay for the Damage
«Cost» Average Processing Cost per Month ≤ 15$

Figure 3.3: Example business process based on the UML 2 profile for business process goals and performance measures

3.4 The extended EPC Metamodel

The metamodel of EPC is extended by introducing a new view, the so called performance measure view. It is shown with the performance measure elements highlighted in grey in figure 3.4.

A *Process Goal* can have several *Sub Goals*. Each *Goal* has at least one *Measure* and is connected with one or more *Measure Flow Connectors*. *Measure* and *Process Goal* are specialisations of *Additional Process Object*. Furthermore a *Measure* is specialised by the subclasses *Cost, Quality* and *Cycle Time*. A *Function* can be related to *Additional Process Objects*. This means a *Measure* is assigned to at least one *Function*.

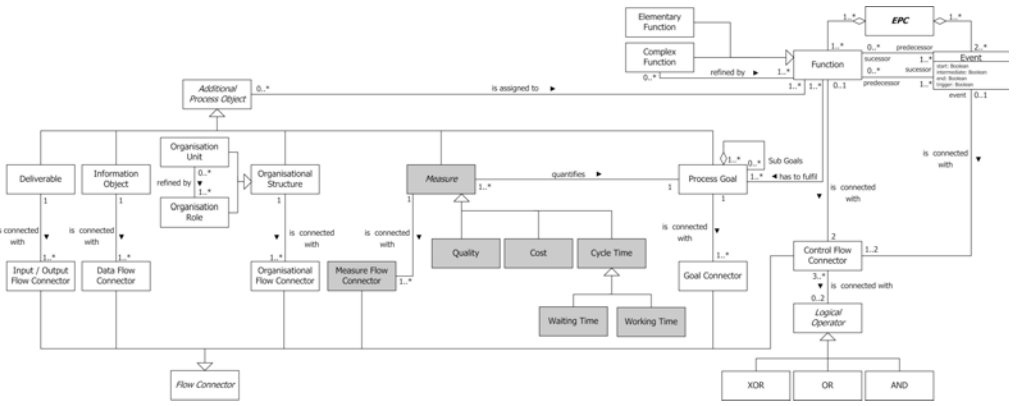


Figure 3.4: Extended EPC metamodel with performance measures

The relationship between goals and measures is illustrated in a so called goal measure tree in figure 3.5. The goal measure tree is related to the example business process of an insurance claim in the figures 3.6, 3.7, and 3.8. Its main process goal is *good process performance*. This goal has three sub-goals: *low processing costs, short process duration*, and high *customer satisfaction*. Furthermore each goal is refined by measures. The goal low processing costs is fulfilled, when the *average processing costs per month* are under 15 Euros. The measure cycle time indicates that the process duration has to be less than four days. Moreover the goal *high customer satisfaction* is achieved, if the *average percentage of complaints* per month is less than five percent.

3.4.1 Example Business Process of an EPC

We demonstrate the practical applicability of the extension of the EPC with business process goals and performance measures in figure 3.4 with the business process of an insurance company: the Processing of Automobile Claims business process (figures 3.6, 3.7 and 3.8). The whole business process is explained in subsection

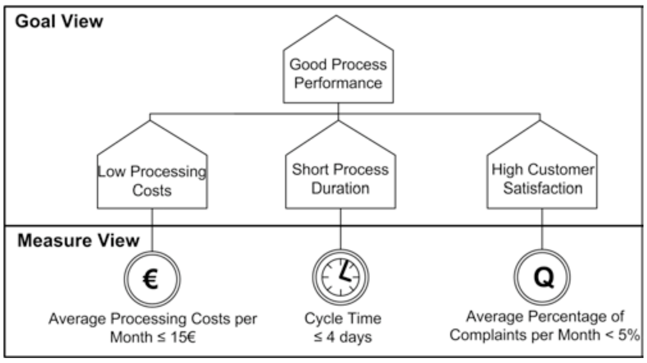

Goal View
Good Process Performance
Low Processing Costs
Short Process Duration
High Customer Satisfaction
Measure View
€
Average Processing Costs per Month ≤ 15€
Cycle Time ≤ 4 days
Q
Average Percentage of Complaints per Month < 5%

Figure 3.5: Goal Measure Tree

3.3.2.

The business process shown with EPC is decomposed into three hierarchical levels to improve the structure and clarity. The main difference in the graphical notation of the extension of UML 2 AD, BPMN, and EPC is that EPC uses new graphical notation elements for presenting the performance measures, while the other two languages uses no graphical notation elements and integrates them into the existing elements.

At the first hierarchy level, the process goals and the performance measures cost, cycle time and quality with the core business process *Processing of Automobile Insurance Claims* are shown in figure 3.6.

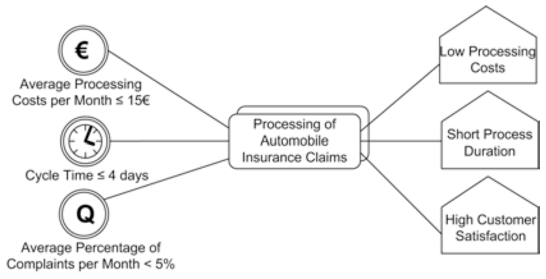

€
Average Processing Costs per Month ≤ 15€
Cycle Time ≤ 4 days
Q
Average Percentage of Complaints per Month < 5%
Processing of Automobile Insurance Claims
Low Processing Costs
Short Process Duration
High Customer Satisfaction

Figure 3.6: First hierarchy level of the example business process

At the second hierarchy level the business process is split up into two subprocesses, namely *Assertion of the Claim* and *Compensation of the Claim*. The organisational roles *Financial Claim Specialist* and *Claim Administrator* which are responsible for the execution of the business process are presented figure 3.7. Furthermore the cycle time of the first hierarchy level is split up into to a subset of two cycle times, where each cycle time belong to one subprocess.

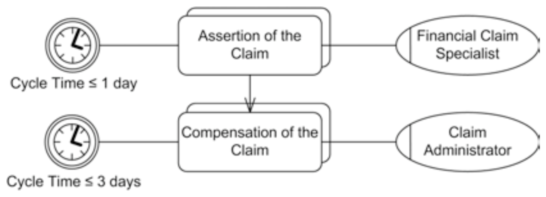


Figure 3.7: Second hierarchy level of the example business process

The third hierarchy level presents the fine granular business process of EPC. Furthermore the different organisational roles as well as the maximum waiting time of two days the organisational role of the *Claim Administrator* has to follow up with the process.

3.5 The extended BPMN Metamodel

Figure 3.9 shows the metamodel of BPMN which is extended by performance measures as a new category (in grey). According to the specification [OMG06a], it is not allowed to change the basic shape of the defined graphical elements and markers. Therefore the extensions are marked with the term "is presented through" in the metamodel, to sign that an extended metaclass is graphically described through a core element of BPMN.

The *Organisational Structure* explicitly describes *Organisational Units* and *Roles* within a business process. F or example this could be the department or an employee of a company. They are presented through a pool, because they are a concrete specification of a pool and so far also part of the category swimlanes. An organisational unit has one or more roles, and a role belongs to at most one unit. The metamodel extended with the new introduced category of performance measures are highlighted in grey in Figure 3.9. A *Measure* is distinguished between a measure on *Type Level* or *Instance Level*. *Cost* and *Quality* belong to the type level, and cycle time to instance level. Cost and quality are in contrast to cycle time more focused on the type level of a process, as the required data is often not available on instance level. The instance level of BPMN can be executed with a mapping to BPEL through a workflow engine according to the specification [OMG06a]. EPC was not

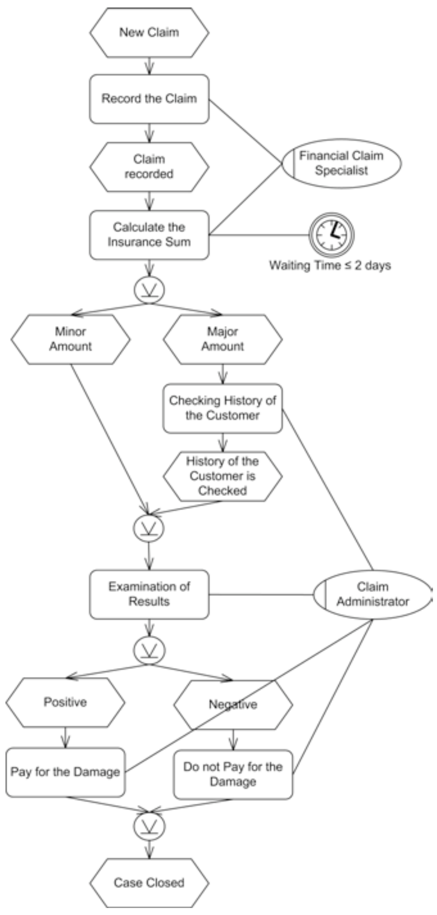


Figure 3.8: Third hierarchy level of the example business process

designed to be executable, therefore the language does not need a distinction in its metamodel between type or instance level. A measure is represented by a pool, because an organisational structure has to act on measures. If the measure is cycle time, then it is represented through a time event. Furthermore an organisational structure can be triggered by an event alert, if an action or a group of actions is not executed within its performance measures.

3.5.1 Example Business Process of a BPMN

The example business processes in the figures 3.10, 3.11 and 3.12 show the business process goals and performance measures graphically integrated in BPMN. The

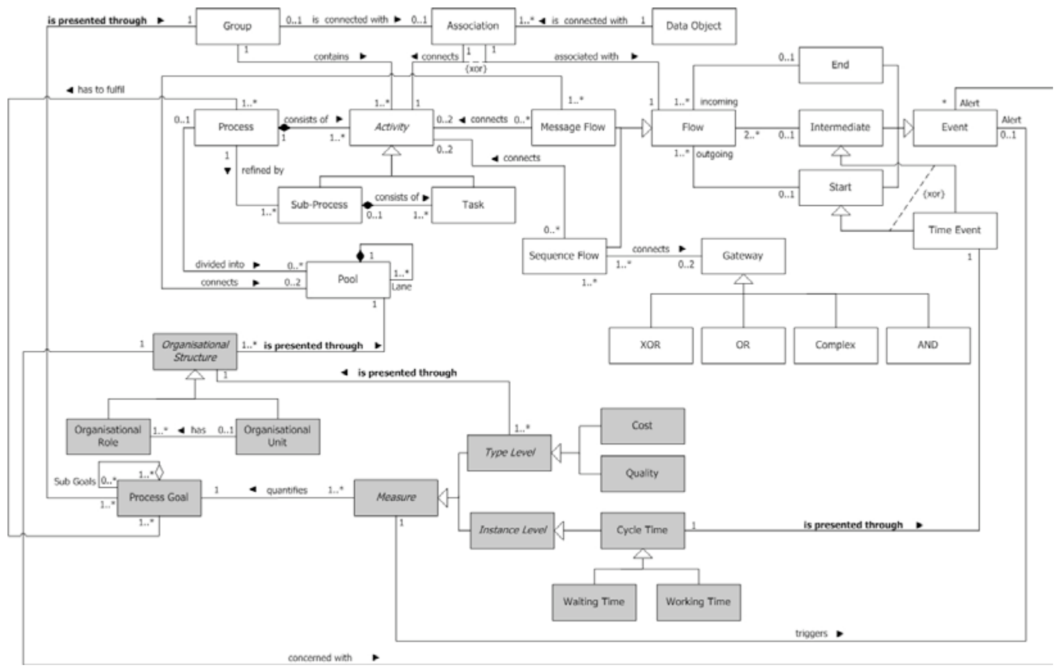


Figure 3.9: Extended BPMN metamodel with performance measures

whole business process is explained in detail in subsection 3.3.2.

The business process shown with BPMN is decomposed into three hierarchical levels to improve the structure and clarity. In BPMN, extensions to notation elements can be made by means of new markers or indicators associated with the current graphical elements. It is recommended to use the existing graphical notation elements, and to keep away from changing them. In the example in figure 3.10 we introduce an additional label for a pool, namely the ”Organisational Role”. It corresponds to the homonymous metaclass in the metamodel in figure 3.9.

The overall goal of the collapsed sub-process in BPMN with the label *Process of Automobile Insurance Claims* is to fulfil the process goals and the performance measures cost, cycle time and quality. Figure 3.10 shows the first hierarchy level of the business process. Furthermore in BPMN it is possible to introduce alerts in a diagram with time events[OMG06a]. In our example, if the cycle time is over four days, then the *Claim Manager* receives an alert, and gets a report about that specific case.

At the second hierarchy level the business process is split up into two subpro-

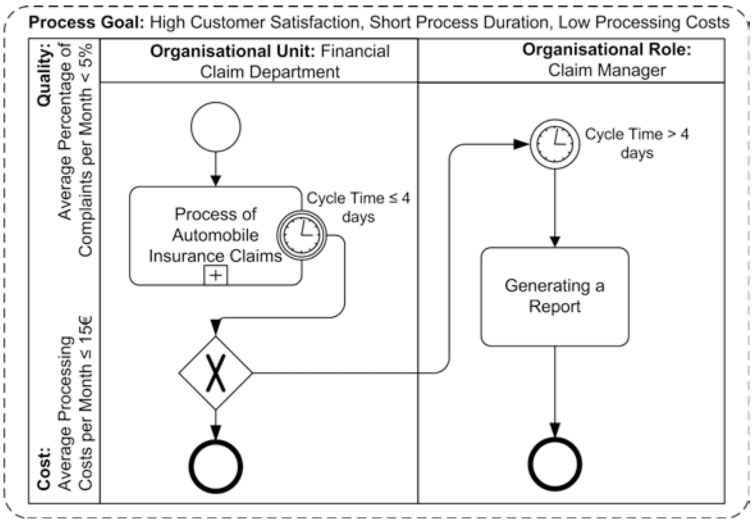

Process Goal: High Customer Satisfaction, Short Process Duration, Low Processing Costs
Quality: Average Percentage of Complaints per Month < 5%
Cost: Average Processing Costs per Month ≤ 15€
Organisational Unit: Financial Claim Department
Organisational Role: Claim Manager
Process of Automobile Insurance Claims
Cycle Time ≤ 4 days
Cycle Time > 4 days
Generating a Report

Figure 3.10: First hierarchy level of the example business process

cesses, namely *Assertion of the Claim* and *Compensation of the Claim*. The organisational roles *Financial Claim Specialist* and *Claim Administrator* which are responsible for the execution of the business process are also presented in figure 3.11. Furthermore the cycle time of the first hierarchy level is split up into to a subset of two cycle times, where each cycle time belong to one subprocess.

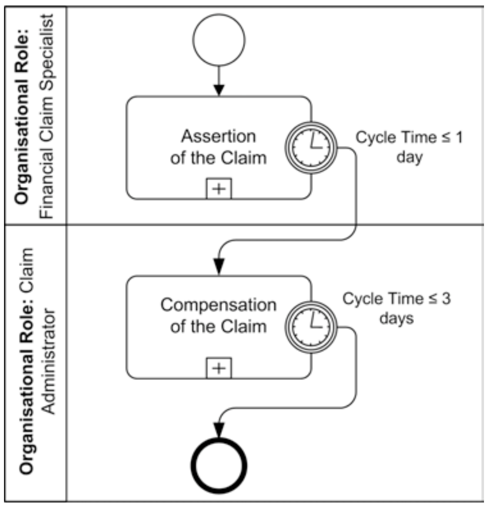

Organisational Role: Financial Claim Specialist
Organisational Role: Claim Administrator
Assertion of the Claim
Cycle Time ≤ 1 day
Compensation of the Claim
Cycle Time ≤ 3 days

Figure 3.11: Second hierarchy level of the example business process

The third hierarchy level presents in figure 3.12 the fine granular business process of BPMN. Furthermore the different organisational roles as well as the maximum waiting time of two days the organisational role of the *Claim Administrator* has to follow up with the process.

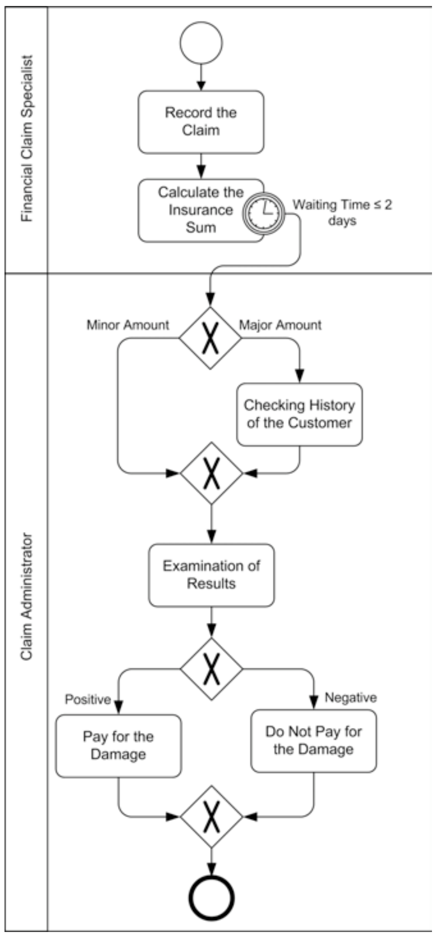


Figure 3.12: Third hierarchy level of the example business process

4 Mapping the UML 2 Activity Diagram to BPEL

Contents

The Business Process Execution Language (BPEL) is a language for specifying business process behaviour based on Web Services. Section 4 defines a mapping between the UML profile for performance measures and goals and BPEL for establishing a basis to transform a specific business process modelling language and its conceptually described performance measures to an execution language as well as to monitor the process instances continuously.

This described approach consists of two main steps. First, the UML 2 profile for performance measures and goals is annoted with stereotyped actions, and mapped to BPEL which is shown by a graphically described example. The performance measures cost and quality are not mapped to BPEL, because our approach focusses on the instance level of a business process and not on the type level. Furthermore cost and quality cannot be shown in BPEL, because their are not part of BPEL. Second, the mapping of the UML Profile to BPEL is defined as a UML Profile in eclipse. Thus we provide a foundation for transforming the UML profile to BPEL code with eclipse.

The remainder of this section is organized as follows:

- Subsection 4.1 describes the base constructs of BPEL and WSDL.
- Subsection 4.2 uses the described base constructs of BPEL and WSDL in subsection 4.1 for specifying an example business process with BPEL and WSDL.
- Subsection 4.3 defines the mapping of the UML 2 profile for performance measures and goals annoted with stereotyped actions to BPEL. Simply the time measures are mapped to BPEL, because the measures cost and quality are not part of BPEL.
- Subsection 4.4 defines the mapping of the UML 2 profile to BPEL designed in eclipse. This is the foundation for transforming a BPML to BPEL code, as described in subsection 4.5.

4.1 Introduction to WSDL and BPEL

4.1.1 Business Process Execution Language (BPEL)

The Business Process Execution Language (BPEL) is an XML-based language for describing business processes. In this thesis WS-BPEL 2.0 [Oas07] is used, which standardization process was finished in April 2007. The activities of a business process defined with BPEL is implemented by web services.

BPEL supports the description of *abstract* and *executable processes*. The orchestration of executable processes are deployed by a workflow engine. The choreography of abstract processes describe the behavioral interface of a process, and are used for hiding the internal behaviour of a process, e.g. from a business partner.

BPEL is used for illustrating the orchestration of web services. An orchestration is defined as an executable business process that describes a flow from a single perspective and under control of a single endpoint. This means that BPEL controls the participating web services of a business process, for instance to order the arrival of messages. BPEL does not support the direct interaction with humans, because BPEL processes only communicate with web services that are described by the Web Service Description Language (WSDL).

The Web Service Description Language (WSDL) [W3C01] defines a programming language-, protocol-, and platform-independent XML specification for describing

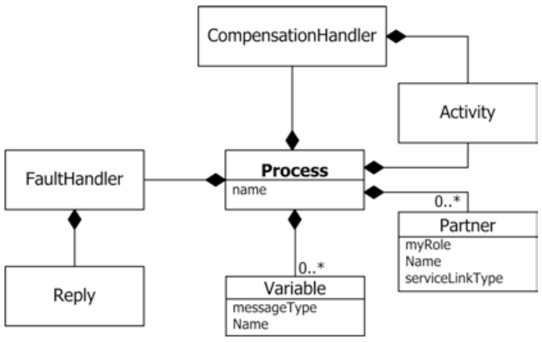


Figure 4.1: Process metamodel of BPEL

web services for exchanging messages. For this purpose WSDL describes the function, datatypes and protocols of a web service.

The overall structure of BPEL consists of partner/partner links, variables, correlation sets, fault handlers, event handlers, compensation handlers. Figure 4.1 describes the BPEL 2.0 process metamodel. Since no official metamodel of BPEL exist, Dubray [Jea05] created an reversed enigneered metamodel of BPEL, which conforms to BPEL 1.1. We use the BPEL metamodel of Dubray for adapting it on BPEL 2.0.

Partner links are parties that interact with the business process. *Variables* describe data that is used by the business process. *Correlation* sets are set of properties shared by messages in a correlated group. *Fault handlers* are activities that must be performed in response to faults. *Event handlers* are invoked concurrently if the corresponding event occurs. *Compensation handlers* are a wrapper for a compensation activity.

Figure 4.2 describes the BPEL 2.0 activity metamodel. *Basic activities* are atomic activities for describing a process with BPEL. <*assign*> updates the values of variables or partner links with new data. An <*invoke*> is a synchronous (request/response) or asynchronous (one way) call of a web service. <*receive*> waits for a message to arrive. If the message requires a response, it is send back by a <*reply*>. <*throw*> signals a fault from inside the business process, which is processed by fault handlers. <*exit*> quits the behaviour of a business process instance. <*wait*> stays for a given time period or until a certain time has passed. <*empty*> is a no-operation activity, that means it is used to specify that no action is executed. This activity is

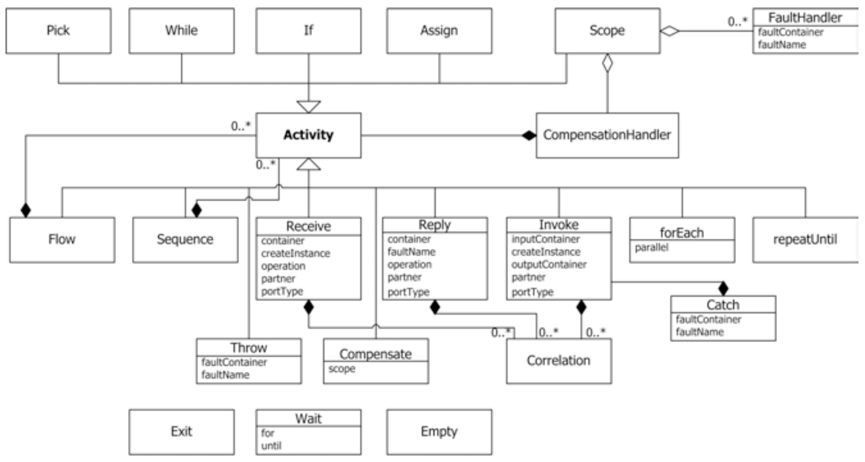


Figure 4.2: Activity metamodel of BPEL

used for fault handlers that consume a fault without acting on it. Other use cases for the empty activity include synchronization points in a flow, or placeholders for activities that are to be added later.

Structured activities contain other activities for defining the recursive composition of complex processes. <*sequence*> executes the activities one after another. <*while*> indicates that an activity is to be repeated until a certain stop criteria is met. <*switch*> selects one branch out of serveral activity branches. <*flow*> executes activities concurrently or in any different order. Dependencies between activities are defined by <*links*>. <*pick*> blocks and waits for a suitable message to arrive or for starting a time-out alarm. <*scope*> handles a set of activities for describing a nested activity. A scope may be associated with a fault handler, an event handler or a compensation handler.

4.1.2 Web Service Description Language (WSDL)

The Web Service Description Language (WSDL) defines a protocol and platform independent XML specification for describing web services to exchange messages. WSDL describes the functions, datatypes and protocols of a web service.

A web service defined by WSDL consists of six main XML elements, and distinguishes between *abstract* and *concrete* definitions. To the abstract definitions belong the elements types, messages, and port types. The elements binding, ports, and services belong the concrete definitions.

Types define data types, that are needed for exchanging messages. *Messages* define the transmitted data. Every message defines a link to datatypes. *Port types* distinguish between four different types of operations. If the web service gets an input message from the client without sending a response, then it is called *one-way* message. A *request-response* message consists of two steps. Firstly, the web service receives an input message from the client. Secondly, the web service sends an output message to the client. During a *notification* the web service sends an output message to the client, but does not receive an input in return. When the web service also expects an answer from the client, then it is called *solicit-response*. A *binding* sets the concrete protocol and data format for the messages, which are declared by a specific port type. A port specifies an address for a binding, where a certain binding is located. This is usually done by a URI.

4.2 Describing an example business process with WSDL and BPEL

This subsection describe the example business process of an insurance claim with WSDL and BPEL. Listing 4.1 defines the types of the business process. The messages of the example business process are described in listing 4.2. The types are defined within the <types> tags. According to the XML schema definition, simple or complex data types are used. For example, in listing 4.1 at line number 12 describes "ClaimInfo" as well as its complex type with the element "ClaimInfo". Listing 4.3 describe the port types. This subsection only describe the elements of the abstract definitions, because the concrete definitions are not relevant for this work. Listing 4.4 describe the partner link types, which are a specific extension of WSDL. define the schema with WSDL of the BPEL process in the listings 4.5, and 4.6.

Listing 4.1: First part of WSDL code of the insurance claim example

```
1: <?xml version="1.0"?>
2: <definitions name="Customer_FinancialClaimSpecialist"
3:  targetNamespace="http://eclipse.org/bpel/sample"
4:  xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
5:  xmlns="http://schemas.xmlsoap.org/wsdl/"
6:  xmlns:ecl="http://eclipse.org/bpel/sample">
7: <types>
8:         <schema attributeFormDefault="unqualified"
9:                 elementFormDefault="qualified"
10:                targetNamespace="http://eclipse.org/bpel/sample"
11:                xmlns="http://www.w3.org/2001/XMLSchema">
```

```
12:          <element name=”claimInfo”>
13:              <complexType>
14:                  <sequence>
15:                      <element name=”sendInfo” type=”string” />
16:                  </sequence>
17:              </complexType>
18:          </element>
19:          <element name=”calculatedClaimInfo”>
20:              <complexType>
21:                  <sequence>
22:                      <element name=”sendCalculatedInfo” type=”string” />
23:                  </sequence>
24:              </complexType>
25:           </element>
26:           <element name=”ClaimInTimeInfo”>
27:               <complexType>
28:                   <sequence>
29:                       <element name=”inTime” type=”string” />
30:                   </sequence>
31:               </complexType>
32:           </element>
33:           <element name=”RepairPropertyofCustomer”>
34:               <complexType>
35:                   <sequence>
36:                       <element name=”RepairinTime” type=”string” />
37:                   </sequence>
38:               </complexType>
39:           </element>
40:           <element name=”claimInfoBack”>
41:               <complexType>
42:                   <sequence>
43:                       <element name=”sendInfoBack” type=”string” />
44:                   </sequence>
45:               </complexType>
46:           </element>
47:      </schema>
48:</types>
```

A message element in listing 4.2 combines different parameters to a group, to offer them to operations which are defined in listing 4.3. For instance at line number 1 the message “ClaimMessage” is defined. It consists of one part only, namely “claimInfo”. “ClaimInfo” is associated to the type “tns:claimInfo” which corresponds to the defined element in line number 12 at listing 4.1.

Listing 4.2: Second part of WSDL code of the insurance claim example

```
1:  <message name=”ClaimMessage”>
2:      <part name=”claimInfo” element=”tns:claimInfo”/>
3:  </message>
4:  <message name=”CalculatedClaimMessage”>
5:      <part name=”calculatedClaimInfo” element=”tns:CalculatedClaimMessage”/>
6:  </message>
7:  <message name=”PassClaimInTimeMessage”>
8:      <part name=”ClaimInTimeInfo” element=”tns:PassClaimInTimeMessage”/>
9:  </message>
10: <message name=”repairPropertyMessage”>
11:     <part name=”RepairPropertyofCustomer” element=”tns:repairPropertyMessage”/>
12: </message>
13: <message name=”ClaimMessageBack”>
14:     <part name=”claimInfoBack” element=”tns:claimInfoBack”/>
15: </message>
16: <message name=”Insurance_ClaimResponseMessage”>
17:     <part name=”payload” element=”tns:Insurance_ClaimResponse”/>
18: </message>
```

Port types define the operations that are performed by web services. In listing 4.3 at line number 1 the port type "FinancialClaimSpecialist_PT" defines the operation "sendClaimInfo" with its input message "tns:ClaimMessage" which is defined in listing 4.2 at line number 1. Listing 4.3 defines only one way operations. This means a port type does not wait for an output message.

Listing 4.3: Third part of WSDL code of the insurance claim example

```
1:  <portType name="FinancialClaimSpecialist_PT">
2:     <operation name="sendClaimInfo">
3:        <input message="tns:ClaimMessage"/>
4:     </operation>
5:  </portType>
6:  <portType name="ClaimAdministratorPT">
7:     <operation name="sendCalculatedClaim">
8:        <input message="tns:CalculatedClaimMessage"/>
9:     </operation>
10: </portType>
11: <portType name="ClaimAdministratorPT">
12:    <operation name="passClaimInTime">
13:       <input message="tns:PassClaimInTimeMessage"/>
14:    </operation>
15: </portType>
16: <portType name="Garage_PT">
17:    <operation name="repairProperty">
18:       <input message="tns:repairPropertyMessage"/>
19:    </operation>
20: </portType>
21: <portType name="Customer_PT">
22:    <operation name="sendClaimInfoBack">
23:       <input message="tns:ClaimMessageBack"/>
24:    </operation>
25: </portType>
```

Partner link types define the roles in a relationship between two services. Furthermore each role is refers to a port type. For instance in listing 4.4 at line number 1 the partner link type "Customer_FinancialClaimSpecialist" defines the roles "Customer" and "FinancialClaimSpecialist". "Customer" refers to the port type "CustomerPT", and "FinancialClaimSpecialist" refers to the port type "FinancialClaimSpecialist_PT". The partner link types and their referenced port types are described in listing 4.3 at line number 21 and 1. For instance the partner link type "ClaimAdministrator_FinancialClaimSpecialist" in listing 4.4 at line number 5 corresponds to the partner link type used at the partnerLink with the name "ClaimAdministrator_FinancialClaimSpecialist" in listing 4.5 at line number 14.

Listing 4.4: Forth part of WSDL code of the insurance claim example

```
1:  <plnk:partnerLinkType name="Customer_FinancialClaimSpecialist">
2:          <plnk:role name="Customer" portType="CustomerPT"/>
3:          <plnk:role name="FinancialClaimSpecialist" portType="FinancialClaimSpecialist_PT"/>
4:       </plnk:partnerLinkType>
5:       <plnk:partnerLinkType name="ClaimAdministrator_FinancialClaimSpecialist">
6:          <plnk:role name="ClaimAdministrator" portType="ClaimAdministratorPT"/>
7:          <plnk:role name="FinancialClaimSpecialist" portType="FinancialClaimSpecialist_PT"/>
```

```
    </plnk:partnerLinkType>
        <plnk:partnerLinkType name="Garage_ClaimAdministrator">
        <plnk:role name="Garage" portType="Garage_PT"/>
        <plnk:role name="ClaimAdministrator" portType="ClaimAdministratorPT"/>
    </plnk:partnerLinkType>
        <plnk:partnerLinkType name="Customer_ClaimAdministrator">
        <plnk:role name="Customer" portType="Customer_PT"/>
        <plnk:role name="ClaimAdministrator" portType="ClaimAdministratorPT"/>
    </plnk:partnerLinkType>
        <plnk:partnerLinkType name="ClaimManager_ClaimAdministrator_">
        <plnk:role name="ClaimAdministrator" portType="ClaimAdministratorPT"
        <plnk:role name="ClaimManager" portType="ClaimManager_PT"/>/>
    </plnk:partnerLinkType>
</definitions>
```

<partnerLinks> define how a process communicates with its partners. Each partner may capture different roles to other partners. For instance in listing 4.5 at line number 10 the <partnerLink> "Customer_FinancialClaimSpecialist" consists of the roles "FinancialClaimSpecialist" as well as "Customer", and is described by the <partnerLinkType> "tns:Customer_FinancialClaimSpecialist". "FinancialClaimSpecialist" defines the role within a business process, namely <myRole>. "Customer" describes the function of the external partner, namely <partnerRole>.

Listing 4.5 contains three <myRole> tags and two <partnerRole> tags. The <myRole> tags are "Claim Administrator", "Claim Manager" and "FinancialClaimSpecialist". These thee roles describe the perspective of the flow of the business process inside the insurance company. "Customer" and "Garage" describe the <partnerRole> tags. These roles illustrate the external partners of the business process. Moreover the partner link types in listing 4.4 describe the types in WSDL which are associated to the partner links in BPEL in listing 4.5.

Variables are used for the exchange of messages. Every variable must be defined by a data type, namely WSDL message type, XML Schema simple type, or XML schema element. In listing 4.5 at line number 36 the variable "Claim" is defined by the element "tns:claimInfo" which corresponds to the illustrated element in listing 4.1 at line number 12. In listing 4.5 each variable element corresponds to the described elements in listing 4.1.

In listing 4.5 the event handler in line number 53 reacts to events that occur while the business process executes. <onAlarm> specifies that if the claim needs for processing more then four days, then <throw> generates a fault. Afterwards the claim manager invokes the claim administrator to create a report. After creating a report about the case, the claim administrator sends a report back to the claim manager.

Listing 4.5: First part of BPEL code of the insurance claim example

```
<?xml version="1.0" encoding="UTF-8"?>
<bpws:process exitOnStandardFault="yes" name="Insurance_Claim2"
        suppressJoinFailure="yes"
        targetNamespace="http://eclipse.org/bpel/sample"
        xmlns:bpws="http://docs.oasis-open.org/wsbpel/2.0/process/executable"
        xmlns:tns="http://eclipse.org/bpel/sample">
<bpws:import importType="http://schemas.xmlsoap.org/wsdl/"
        location="Insurance_Claim.wsdl" namespace="http://eclipse.org/bpel/sample"/>
  <bpws:partnerLinks>
          <bpws:partnerLink myRole="FinancialClaimSpecialist"
             name="Customer_FinancialClaimSpecialist"
             partnerLinkType="tns:Customer_FinancialClaimSpecialist"
             partnerRole="Customer"/>
          <bpws:partnerLink myRole="FinancialClaimSpecialist"
             name="ClaimAdministrator_FinancialClaimSpecialist"
             partnerLinkType="tns:ClaimAdministrator_FinancialClaimSpecialist"
             partnerRole="ClaimAdministrator"/>
          <bpws:partnerLink myRole="ClaimAdministrator"
             name="Garage_ClaimAdministrator"
             partnerLinkType="tns:Garage_ClaimAdministrator"
             partnerRole="Garage"/>
          <bpws:partnerLink myRole="ClaimAdministrator"
             name="Customer_ClaimAdministrator"
             partnerLinkType="tns:Customer_ClaimAdministrator"
             partnerRole="Customer"/>
          <bpws:partnerLink myRole="ClaimAdministrator"
             name="Customer_ClaimAdministrator"
             partnerLinkType="tns:Customer_ClaimAdministrator"
             partnerRole="Customer"/>
          <bpws:partnerLink myRole="ClaimAdministrator"
             name="ClaimManager_ClaimAdministrator"
             partnerLinkType="ClaimManager_ClaimAdministrator"
             partnerRole="ClaimManager">
  </bpws:partnerLinks>
  <bpws:variables>
       <bpws:variable element="tns:claimInfo"
             name="Claim"/>
       <bpws:variable element="tns:calculatedClaimInfo"
             name="passedClaim"/>
       <bpws:variable element="tns:claimInfo"
             name="ClaimInTime"/>
       <bpws:variable element="tns:claimInfoBack"
             name="Letter2Customer"/>
       <bpws:variable element="tns:claimInfo"
             name="DataOfCMS"/>
       <bpws:variable element="tns:RepairPropertyofCustomer"
             name="damagedProperty"/>
       <bpws:variable element="tns:RepairPropertyofCustomer"
             name="fixedProperty"/>
       <bpws:variable element="tns:claimInfo"
             name="Report"/>
       </bpws:variables>
       <bpws:eventHandlers>
           <bpws:onAlarm for="'P4DT'">
               <bpws:throw faultName="Timeout" faultVariable="Fault" />
               <bpws:invoke name="CreateAReport"
                    inputVariable="ClaimNotFinished"
                    operation="finishClaim"
                    outputVariable="Report"
                    partnerLink="Manager_Administrator"/>
               <bpws:receive name="Receive_ReportGenerated"
                    operation="finishClaim"
                    partnerLink="Manager_Administrator"
                    variable="Report"/>
           <bpws:/onAlarm>
       </bpws:eventHandlers>
```

In listing 4.6 the BPEL process starts with the <receive> tag *record the claim*. The financial claim specialist waits for *receiving the claim* from the customer. After receiving the claim, the financial claim specialist invokes the service *calculate the insurance sum* for passing the calculated claim to the claim administrator. After the calculation of the insurance sum is successful accomplished by the claim administrator, the claim administrator sends an answer back to the claim specialist, which is shown at line number 20. Furthermore if the damage is above a specific amount, then the *history of the customer* has to be checked, by updating the claim files with data of the *customer management system (CMS)*. After that the claim administrator invokes the garage for *repairing the property* of the customer. When the reparation is done, the garage informs the claim administrator in line number 39. If the claim is enabled to *pay the insurance sum* to the customer, then the claim administrator replies an appriate answer back to the customer. Also the customer receives a letter from the claim administrator if the insurance sum will not be paid. In both cases the process is closed afterwards.

Listing 4.6: Second part of BPEL ode of the insurance claim example

```
1:  <bpws:sequence name="Sequence">
2:          <bpws:receive name="Receive_RecordTheClaim"
3:             operation="sendClaimInfo"
4:             partnerLink="Customer_FinancialClaimSpecialist"
5:             variable="Claim"/>
6:      <bpws:invoke inputVariable="Claim"
7:             name="Invoke_CalculateTheInsurance"
8:             operation="sendCalculatedClaim"
9:             outputVariable="passedClaim"
10:            partnerLink="ClaimAdministrator_FinancialClaimSpecialist"/>
11:     <bpws:receive name="reply_InsuranceSumCalculated"
12:            operation="sendClaimInfo"
13:            partnerLink="FinancialClaimSpecialist_ClaimAdministrator"
14:            variable="Claim"/>
15:         <bpws:if name="If">
16:                <bpws:assign name="Assign_CheckHistoryOfTheCustomer" validate="no">
17:                     <bpws:copy>
18:                          <bpws:from variable="DataOfCMS"/>
19:                          <bpws:to variable="Claim"/>
20:                     </bpws:copy>
21:                </bpws:assign>
22:                <bpws:condition><![CDATA[damageIsTooHigh=true()]]></bpws:condition>
23:                <bpws:else/>
24:         </bpws:if>
25:     <bpws:invoke inputVariable="damagedProperty"
26:            name="Invoke_ContactTheGarage"
27:            operation="repairProperty"
28:            outputVariable="fixedProperty"
29:            partnerLink="Garage_ClaimAdministrator"/>
30:     <bpws:receive name="reply_InsurancedPropertyFixed"
31:            operation="repairProperty"
32:            partnerLink="Garage_ClaimAdministrator"
33:            variable="fixedProperty"/>
34:         <bpws:assign name="Assign_ExaminationOfResults" validate="no">
35:                <bpws:copy>
36:                     <bpws:from variable="Claim"/>
37:                     <bpws:to variable="DataOfCMS"/>
```

```
            </bpws:copy>
        </bpws:assign>
        <bpws:if name="If1">
          <bpws:reply name="Reply_DoPayForTheDamage"
                  operation="sendClaimInfoBack"
                  partnerLink="Customer_ClaimAdministrator"
                  variable="Letter2Customer"/>
          <bpws:condition><![CDATA[CaseIsJustified=true()]]></bpws:condition>
          <bpws:else>
              <bpws:reply name="Reply_DontPayForTheDamage"
                      operation="sendClaimInfoBack"
                      partnerLink="Customer_ClaimAdministrator"
                      variable="Letter2Customer"/>
          </bpws:else>
    </bpws:if>
  </bpws:sequence>
</bpws:process>
```

4.3 Mapping the UML 2 Activity Diagrams with time measures to BPEL

The Business Process Execution Language (BPEL) is a language for specifying business process behaviour based on Web Services [Oas07]. The UML profile will be mapped onto BPEL, in order to transform a specific business process modelling language and its conceptually described performance measures to an execution language. Figure 4.3 shows the extended UML 2 Activity Diagram for BPEL based on [BS04].

Bordbar et al. present in [BS04] a transformation of the UML 2 Activity Diagram to BPEL to show the behavioural aspects of web services. This approach is used for mapping the stereotyped actions in figure 3.3 to the BPEL tags *<receive>, <invoke>, <reply>, and <assign>* in figure 4.3. Furthermore, ≪Alert≫ is mapped by using the BPEL tag *<onAlarm>*, as well as
≪Organisational Unit≫ and ≪Organisational Role≫ by using the *<partnerLink>* tag. For sake of simplicity the ≪Cycle Time≫, ≪Waiting Time≫, and ≪Working Time≫ are based on one metaclass, namely
≪AcceptTimeEventAction≫. We do not map the performance measures cycle time, cost and quality to BPEL tags. Cycle time is represented indirectly by <onAlarm>, because an alert and the cycle time have the same time restriction for executing the business process. Furthermore cost and quality are concepts that are not provided by BPEL.

As explained in subsection 4.1, partner links are parties that interact with the business process. A <partnerLink> consists of the attributes *myRole* and *partner-*

Role. The attribute myRole defines the role within a business process, which calls and answers to other business partners. A partnerRole describes the function of the external partner. Since the≪Organisational Unit≫ and ≪Organisational Role≫ are mapped to <partnerLink> tags, and the two stereotypes extend the metaclass activity partition, the <partnerLink> tag is graphically described as an activity partition. To be more precise, the ≪Organisational Unit≫ and ≪Organisational Role≫ are mapped to the attribute myRole, since a business process described with UML 2 AD usually shows only one business partner that interacts with an action, and not an interorganisational view of two or more partners. The the first partition name labels myRole, and the second partition name labels the partnerRole. For a better understanding the activity partitions in figure 4.3 are not shown as swimlanes, but are placed above the activity name according to the specification of UML. The arrow within the action marks if another partner is called (incoming arrow), or if an aswer of a partner is expected (outgoing arrow). If an action has no arrow, then it has to execute a local operation. The partners which are part of the business process are the *claim manager (CM), the claim administrator (CA), financial claim specialist (FCS), customer (C)*, and *garage (G)*.

The example business process consists of a main part, and an *<eventHandler>* which is attached to the business process. An <eventHandler> reacts to events that occur while the business process executes, and consists at least one *<onMessage>* or *<onAlarm>*. For describing the time factor a business process has for executing its tasks, <onAlarm> activity is mapped to ≪AcceptTimeEventAction≫ of UML 2 AD. An alarm event goes off when the specified time or duration has been reached. The *for* attribute specifies the duration after which the event will be triggered. The alternative attribute *until* describes a specific point in time when the alarm will be fired. The clock for the duration starts at the point in time when the associated process starts. For the sake of simplicity we do not integrate the whole <eventHandler> tag with the <onMessage> elements into the diagram. If the overall process execution exceeds four days, then a <onAlarm> activity is activated, which is mapped to the ≪AcceptTimeEventAction≫ in UML 2 AD. Afterwards the <throw> activity generates a fault, which is mapped to the *Interruptible Activity Region*. Afterwards *claim manager* invokes the *claim administrator* for creating a report of the delay. Afterwards the *claim administrator* sends an answer back to the *claim manager* that a report is generated.

Table 4.1: Mapping relations between the UML 2 profile and BPEL

UML Base Class	**UML Stereotype**	**BPEL Tag**
Activity		<process>
StructuredActivityNode		<scope>
Action	≪AcceptEventAction≫	<receive>
Action	≪CallOperationAction≫	<invoke>
Action	≪CallBehaviourAction≫	<assign>
Action	≪SendSignalAction≫	<reply>
ActivityFinalNode		<exit>
InterruptableActivityRegion		<throw>
AcceptTimeEventAction	≪Alert≫	<onAlarm>
ControlFlow		<sequence>
Decision/MergeNode		<If> activity
Fork/JoinNode		<flow> activity
ActivityPartition	≪Organisational Unit≫	<partnerlink>
ActivityPartition	≪Organisational Role≫	<partnerlink>

In a UML 2 Activity Diagram a control flow connects the different action together for describing the process order. For describing a pre-defined order in BPEL, the *Sequence activity* is used. A *<sequence>* tag contains all other tags that have to be proceed in a sequential order.

In UML 2 AD for describing different alternative pathes the decision and merge nodes are used. For describing the same process logic in BPEL the nodes are mapped to the *If activity*. The fork and join nodes describe actions that are executed in parallel. These nodes are mapped to the *<Flow>* activity for describing the same in BPEL.

Table 4.1 shows the mapping relations between the stereotypes of the UML 2 Profile and the BPEL tags, which are used in Figure 4.3. In figure 4.3 the base for the graphical notation of the mapping is a UML 2 Activity Diagram. For the sake of clarity not every BPEL tag is graphically assigned in 4.3. For instance the <sequence> is implicitly described by the control flow that connects the different actions.

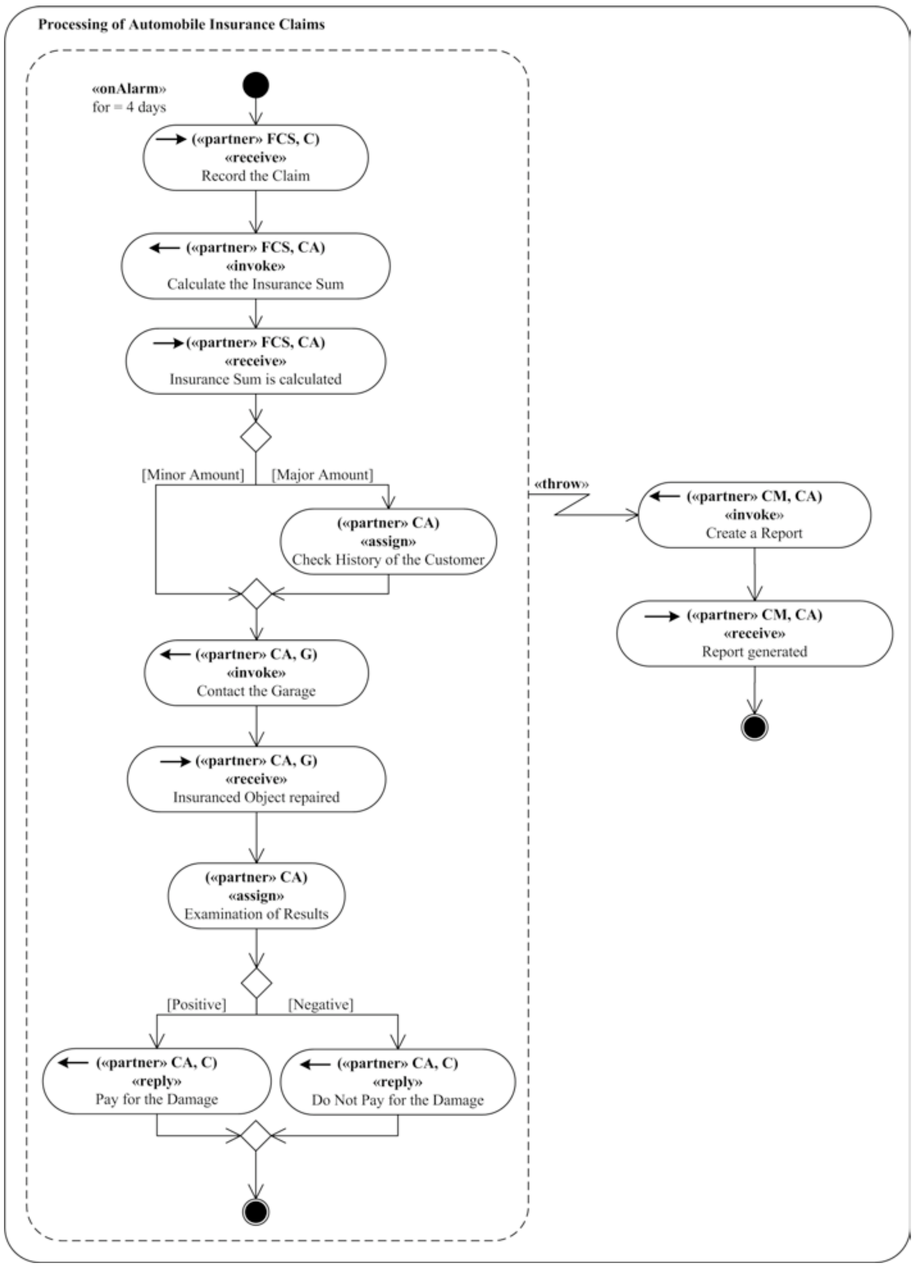


Figure 4.3: Example business process based on the UML 2 profile for business process goals and performance measures and the mapping to BPEL

4.4 Designing a UML Profile for BPEL with Eclipse

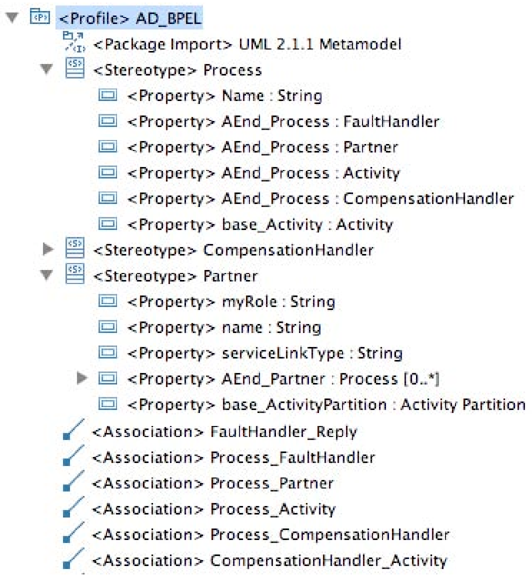
Eclipse is a well known open source framework for developing software, and support multiple platforms. As a result of its structure which is based on plug-ins, it is used for different development challenges. Also a plug-in has been developed for creating UML models as well as UML profiles based on the UML 2 metamodel. The following subsection 4.4.1 and 4.4.2 define a UML model based on the example business process of handling an insurance claim, and a UML profile based on the mapping table 4.3.

4.4.1 A UML 2 Activity Diagram designed with Eclipse

Figure 4.4 shows a part of the UML 2 profile for BPEL designed in eclipse. It describes the stereotypes ≪Process≫, ≪compensationHandler≫, and ≪Partner≫, which are based on the mapping table 4.3. The UML profile consists of its stereotypes, extensions, and the relationships between them. The stereotype ≪Process≫ consists of several properties and extends the metaclass activity. A property labels either the end of an association, or the attribute of a stereotype. Properties with the prefix "AEnd" show the end of an association. In the case of the ≪Process≫, it has associations to ≪FaultHandler≫, ≪Partner≫, and ≪Activity≫, and a property which defines the name of the stereotype. The stereotype ≪Partner≫ has five properties. *myRole, name,* and *serviceLinkType* define attributes. AEnd_Partner describes the association relationship to ≪Process≫, and the property *base_ActivityPartition* illustrates the metaclass of ≪Partner≫.

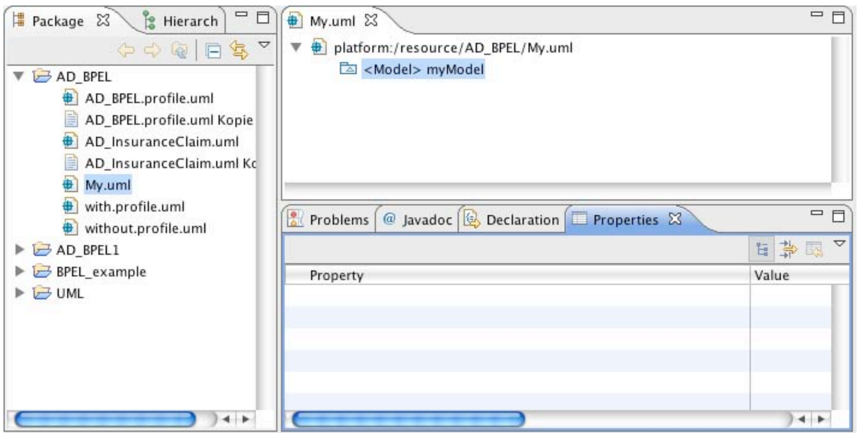
For designing a UML 2 Activity Diagram with the eclipse framework, Eclipse 3.X, EMF 2.X and the UML 2 plug-in have to be installed. EMF is the abbreviation for the Eclipse Modeling Framework. EMF is an open-source framework based on Java for the automatic generation of code based on structured models. As figure shows 4.5, a UML model designed in Eclipse consists of *three main views*: the package view (left), the model view (right), and the property view (at the bottom). The *package view* describes the different projects and models of UML. The *model view* shows the model in a tree view or textual view. The *property view* defines the different options of a model element.

Figure 4.6 describes the UML model of an Insurance Claim, shown in Eclipse and as a graphical representation. For a better understanding the labels of the *Control*



```
<Profile> AD_BPEL
  <Package Import> UML 2.1.1 Metamodel
  <Stereotype> Process
    <Property> Name : String
    <Property> AEnd_Process : FaultHandler
    <Property> AEnd_Process : Partner
    <Property> AEnd_Process : Activity
    <Property> AEnd_Process : CompensationHandler
    <Property> base_Activity : Activity
  <Stereotype> CompensationHandler
  <Stereotype> Partner
    <Property> myRole : String
    <Property> name : String
    <Property> serviceLinkType : String
    <Property> AEnd_Partner : Process [0..*]
    <Property> base_ActivityPartition : Activity Partition
  <Association> FaultHandler_Reply
  <Association> Process_FaultHandler
  <Association> Process_Partner
  <Association> Process_Activity
  <Association> Process_CompensationHandler
  <Association> CompensationHandler_Activity
```

Figure 4.4: Stereotypes of the UML profile shown with eclipse



```
Package  Hierarch
AD_BPEL
  AD_BPEL.profile.uml
  AD_BPEL.profile.uml Kopie
  AD_InsuranceClaim.uml
  AD_InsuranceClaim.uml Kc
  My.uml
  with.profile.uml
  without.profile.uml
AD_BPEL1
BPEL_example
UML

My.uml
platform:/resource/AD_BPEL/My.uml
  <Model> myModel

Problems  Javadoc  Declaration  Properties
Property          Value
```

Figure 4.5: Screenshot of the eclipse views

Flows in Eclipse are also annotated to the graphical representation. The process starts with element *Initial Node*. Furthermore the *Accept Event Action* Record the Claim and the *Call Operation Action* Calculate the Insurance Sum as well as *Accept Event Action* Insurance Sum is Calculated and the different *Control Flows* that connect the different elements together are part of the *Activity Partition* Financial Claim

Specialist.

Now two decision have to be made as figure 4.6 shows: first, whether the history of the customer has to be checked, and second, whether the customer receives the insurance sum from the insurance company. The first decision depends on the *Guards* annotated to the *Control Flows* c4a and c4b whether the insurance sum has a minor amount or a major amount. The second decision depends on the *Guards* annotated to the *Control Flows* c9a and c9b whether the *Call Behaviour Action* Check History of the Customer is judged positive or negative. After the *Send Signal Actions* Pay or Do not Pay for the Damage the case is closed. These elements are part of the *Activity Partition* Claim Administrator.

Furthermore, if the duration of the execution of the process exceeds 4 days, then the *Interruptible Activity Region* stops the process. In that case, the *Activity Partition* Financial Claim Manager gets a report, and the business process ends.

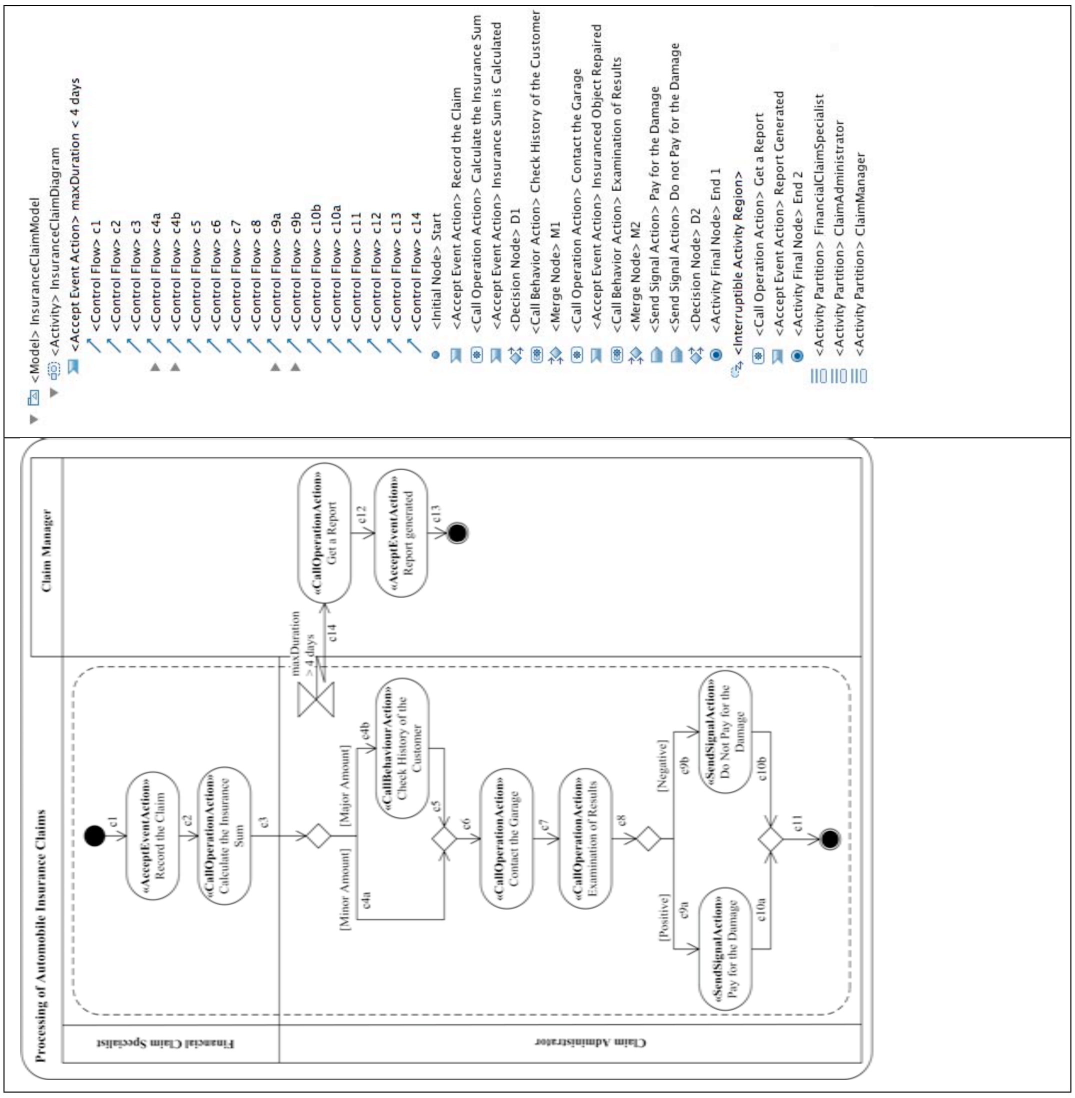
4.4.2 Applying the UML Profile onto a UML 2 Activity Diagram

For applying a UML 2 Profile on a UML 2 Model in Eclipse, several steps have to be made. All these steps are shown in figure 4.7. First, the profile has to be defined, for referencing it to the meta-metamodel Ecore. Second, in order to make use of the profile, the profile is loaded in the workspace of the example model of the Insurance Claim. Third, the profile must be applied to the UML model, for relating the stereotypes that are defined in the profile to the UML model.

4.5 Conceptually desribed Transformation of a BPML to BPEL code

In the sense of Model Driven Engineering (MDE) [OMG06b], the transformation of the Platform Independent Model (PIM), e.g. a UML profile to the Platform Specific Model (PSM), e.g. BPEL code has to be the next step in our work. Now the proper model transformation language has to be taken to find the way from the conceptual level to the implementation level. The most well-known transformation approaches [JK06] are the Query/Views/Transformation (QVT) approach [OMG07b], and the ATLAS Transformation Language (ATL) [Bez05].

ATL is available as a plug-in in eclipse, and is therefore chosen to discourse the



```
<Model> InsuranceClaimModel
  <Activity> InsuranceClaimDiagram
    <Accept Event Action> maxDuration < 4 days
    <Control Flow> c1
    <Control Flow> c2
    <Control Flow> c3
    <Control Flow> c4a
    <Control Flow> c4b
    <Control Flow> c5
    <Control Flow> c6
    <Control Flow> c7
    <Control Flow> c8
    <Control Flow> c9a
    <Control Flow> c9b
    <Control Flow> c10b
    <Control Flow> c10a
    <Control Flow> c11
    <Control Flow> c12
    <Control Flow> c13
    <Control Flow> c14
    <Initial Node> Start
    <Accept Event Action> Record the Claim
    <Call Operation Action> Calculate the Insurance Sum
    <Accept Event Action> Insurance Sum is Calculated
    <Decision Node> D1
    <Call Behavior Action> Check History of the Customer
    <Merge Node> M1
    <Call Operation Action> Contact the Garage
    <Accept Event Action> Insuranced Object Repaired
    <Call Behavior Action> Examination of Results
    <Merge Node> M2
    <Send Signal Action> Pay for the Damage
    <Send Signal Action> Do not Pay for the Damage
    <Decision Node> D2
    <Activity Final Node> End 1
    <Interruptible Activity Region>
    <Call Operation Action> Get a Report
    <Accept Event Action> Report Generated
    <Activity Final Node> End 2
    <Activity Partition> FinancialClaimSpecialist
    <Activity Partition> ClaimAdministrator
    <Activity Partition> ClaimManager
```

Processing of Automobile Insurance Claims
Financial Claim Specialist
Claim Administrator
Claim Manager
«AcceptEventAction» Record the Claim
«CallOperationAction» Calculate the Insurance Sum
[Minor Amount]
[Major Amount]
«CallBehaviourAction» Check History of the Customer
«CallOperationAction» Contact the Garage
«CallOperationAction» Examination of Results
[Positive]
[Negative]
«SendSignalAction» Pay for the Damage
«SendSignalAction» Do Not Pay for the Damage
maxDuration >4 days
«CallOperationAction» Get a Report
«AcceptEventAction» Report generated
c1
c2
c3
c4a
c4b
c5
c6
c7
c8
c9a
c9b
c10a
c10b
c11
c12
c13
c14

Figure 4.6: Example UML 2 Activity Diagram, left a graphical representation, right designed in eclipse

outlook of this section. The designed UML profile in eclipse in this section can be used for creating an ecore complaint *extended UML Metamodel*, as figure 4.8 shows at M2. Models designed with ecore in eclipse are comparable to metamodels in UML. At M2 the *Weaving Model* establish the mapping links between the *extended UML Metamodel* and a *BPEL Metamodel*, which also has to be created in ecore. The *Weaving Model* is based on a *Weaving Metamodel*, which itself is based on MOF, like

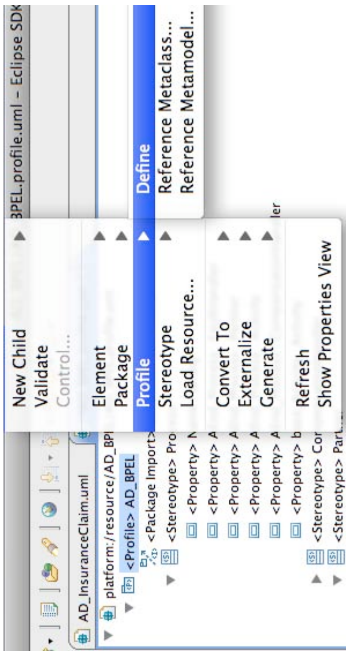
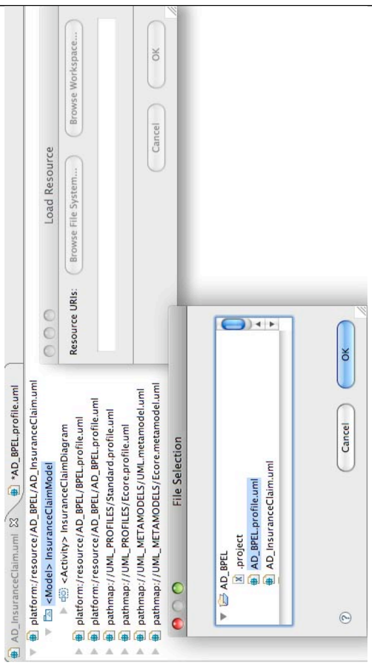
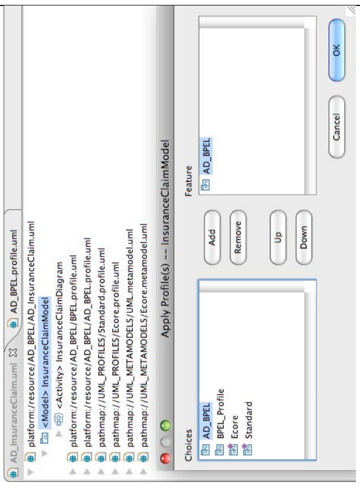
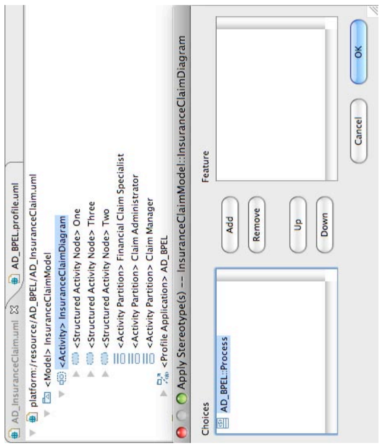
 BPEL.profile.uml – Eclipse SDK New Child Validate Control... Element Package Profile Stereotype Load Resource... Convert To Externalize Generate Refresh Show Properties View Define Reference Metaclass... Reference Metamodel... AD_InsuranceClaim.uml platform:/resource/AD_BPEL <Profile> AD_BPEL <Package Import> <Stereotype> <Property>	 AD_InsuranceClaim.uml *AD_BPEL.profile.uml platform:/resource/AD_BPEL/AD_InsuranceClaim.uml <Model> InsuranceClaimModel <Activity> InsuranceClaimDiagram platform:/resource/AD_BPEL/BPEL.profile.uml platform:/resource/AD_BPEL/AD_BPEL.profile.uml pathmap://UML_PROFILES/Standard.profile.uml pathmap://UML_PROFILES/Ecore.profile.uml pathmap://UML_METAMODELS/UML.metamodel.uml pathmap://UML_METAMODELS/Ecore.metamodel.uml Load Resource Resource URIs: Browse File System... Browse Workspace... Cancel OK File Selection AD_BPEL .project AD_BPEL.profile.uml AD_InsuranceClaim.uml Cancel OK
First step: defining the UML Profile	Second step: loading the UML Profile into the workspace of the model
 AD_InsuranceClaim.uml AD_BPEL.profile.uml platform:/resource/AD_BPEL/AD_InsuranceClaim.uml <Model> InsuranceClaimModel <Activity> InsuranceClaimDiagram platform:/resource/AD_BPEL/BPEL.profile.uml platform:/resource/AD_BPEL/AD_BPEL.profile.uml pathmap://UML_PROFILES/Standard.profile.uml pathmap://UML_PROFILES/Ecore.profile.uml pathmap://UML_METAMODELS/UML.metamodel.uml pathmap://UML_METAMODELS/Ecore.metamodel.uml Apply Profile(s) -- InsuranceClaimModel Choices AD_BPEL BPEL_Profile Ecore Standard Add Remove Up Down Feature AD_BPEL Cancel OK	 AD_InsuranceClaim.uml AD_BPEL.profile.uml platform:/resource/AD_BPEL/AD_InsuranceClaim.uml <Model> InsuranceClaimModel <Activity> InsuranceClaimDiagram <Structured Activity Node> One <Structured Activity Node> Three <Structured Activity Node> Two <Activity Partition> Financial Claim Specialist <Activity Partition> Claim Administrator <Activity Partition> Claim Manager <Profile Application> AD_BPEL Apply Stereotype(s) -- InsuranceClaimModel::InsuranceClaimDiagram Choices AD_BPEL::Process Add Remove Up Down Feature Cancel OK
Third step: Applying the profile onto the model	Forth step: Applying the stereotypes onto the model

Figure 4.7: Applying the UML Profile onto a UML 2 Model

extended UML Metamodel and the *BPEL Metamodel*. With a generator implemented by Wimmer et al. [WSS+07], *ATL Code* is derived from the *Weaving Model*, which uses a UML Model as an input, and generates a BPEL Model as an output. With *MOF Scripts* the *BPEL Model* is transformed to *XML Code*. MOF Script was submitted to the OMG process for MOF Model to Text Transformation, and is developed as an eclipse plug-in. Figure 4.8 shows a graphical representation of the conceptual explained transformation.

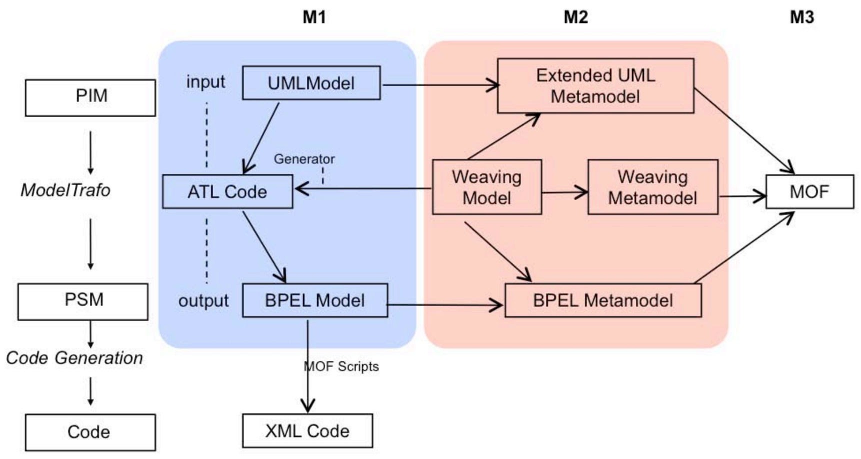


Figure 4.8: Transforming a BPML to BPEL code

5 Linking Business Processes with Software Elements and Variabilities

Contents

5.1 Introduction

Although business processes are often the starting point for software development and define requirements for software systems, linking business processes with software elements is inadequately supported in conceptual modelling. On the on hand side it is not possible to show software requirements or components which have an influence on a BPML. On the other hand side it is not possible to show the variabilities of a BPML.

In order to integrate software requirements and components into a BPML, we describe how the UML 2 profile for EPCs is linked with software systems. We connect the profile with UML 2 elements. On the one hand side we describe the software requirements of a process. On the other hand side we present software components that are provided to successfully realise a process. The extension is used as the starting point for achieving a business-goal oriented software development, but also for a better description of a business process and its supporting software systems.

A variability model defines the variability of a software product line and is used during the different life cycle stages of software product lines [PBvdL05]. Variability modelling is a domain specific modelling technique, that is becoming more and more integrated into traditional software engineering. Variability models also have an impact on processes, because variabilities may change the process flow. Consider, a car engine manufacturing process. The decision wether the variability manufacture a diesel engine or a petrol engine is chosen, changes the process flow. Unfortunately, variability models that describe such cases, are not integrated into popular modelling frameworks, like the Unified Modelling Language.

The UML 2 profile for EPCs [KL06] represents a mapping from EPCs to UML 2 activity diagrams and aims at providing business process models to software developers in a well known notation. In this section, the linking process focuses on the software requirements of a business process and the software components that are necessary to successfully implement and execute the process. The contribution of the linking process is:

- Business process models that are used as a starting point for software development support the achievement of a business goal-oriented software development.
- By linking business process with software systems a better description of a business process and its supporting software systems is provided.
- Business process models in general and the UML 2 profile for EPCs in particular, are utilised to elicit requirements for a new software system, but also for checking whether the functions of an existing software system match the requirements of a new business process.
- We provide a UML 2 profile comprising stereotypes representing software components. The close relationship between these profiles is a further step towards bridging the gap between business process engineering and software engineering.

Variability Models are designed for modelling variabilities of a software. Unfortunately they are not part of a well-known modelling framework for a higher usability, like the Unified Modelling Language. To address this limitation, we provide a UML 2 profile for variability models. Furthermore we show the dependency from the UML profile to activity diagrams to visualize the relationship between variability models and process models. This profile and its mapping are tested with example business processes. The goals of this section are:

- to provide variability models in a UML notation allowing software developers to use any UML tool

- to show the dependency between variability models and business processes to visualize the relationship between structural models and behavioural models in all stages of the software developing process

UML profiles are an extension mechanism for building UML models for particular domains or purposes [OMG05c]. We utilize this well-defined way to develop a UML profile for variability models and describe its dependencies onto activity diagrams, showing the impact of variabilities on the process flow and thus, provide the following contributions:

- The profile provides variability models in UML notation for software developers. Software systems are very complex and require variabilities for defining their process logic e.g. by customization. The profile represents variability requirements to software developers or process engineers in a semi-formal and well-known modelling notation.
- The UML profile for variability models offers the possibility to create, present and edit variability models with existing UML modelling tools, if they support UML profiles.
- The UML profile and its dependencies onto activity diagrams makes the relationship between variabilities and processes visible.

The remainder of this section is organized as follows:

- Subsection 5.2 describes the approach to connect software requirements and components to the UML 2 Profile for EPC.
- Subsection 5.3 presents a UML 2 profile for variabilities. Furthermore the subsection discusses the dependency onto business processes.

5.2 Linking Business Processes with Software Elements

In this section, we describe the details of how a business process, represented as a UML 2 profile for EPCs [KL06] is used in software development. Therefore, we connect business process activities, in our profile stereotyped actions, called ≪*elementary function*≫, with the UML 2 elements *component* and *use case* by using *dependencies*. Use cases are suitable for defining software requirements, while components represent the modular structure of a software system.

The Object Management Group (OMG) [OMG05c] describes use cases as a collection of actions, which stand for a specific behaviour. According to the OMG [OMG05c], a component covers physical and logical modelling aspects; this means that a component is a modular part of a system. On the one hand side, use cases represent software in an abstract way, which means that no concrete implementation stands behind them. They describe the requirements of a software. On the other hand side, components describe a software system or part of it.

We use dependencies to connect stereotyped actions with use cases and components, because with dependencies it is possible to connect UML 2 elements from behavioural diagrams with elements from structural diagrams. By contrast, associations cannot connect stereotyped actions with use cases and components. The reason is that the meta-class property represents the association end and belongs to the structural models, while the UML 2 Activity Diagram which is the foundation for the UML profile for EPCs belongs to the behavioural models. Generally, in UML 2 it is impossible to link the two different modelling types with associations. A dependency allows to show that one or more elements, called client(s), are dependent on one or more elements, called supplier(s) [OMG05c]. This means that a modification of the supplier may impact the client. For instance, the client might need the model element of the supplier for its specification or implementation. The graphical notation of a dependency is a dashed arrow. The model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier). The arrow may be labelled with an optional stereotype and an optional name. If no stereotype is labelled, the dependency makes no declaration about its semantic.

5.2.1 Example business processes with software requirements and components

Figure 5.1 shows a dependency relationship between an elementary function and a use case as well as between an elementary function and a component. The left hand side of the figure shows that the elementary function *Internet Buying* has the role of the client, and the use case *Credit Card Payment* represents the supplier. *Internet Buying* needs *Credit Card Payment* for its execution. This means that *Internet Buying* needs the realisation of the software requirement *Credit Card Payment* for further processing. We address the following utilization scenarios for this type of diagram: Process actors directly access the process diagram, by the help of use cases and know how they need to interact with the system. Moreover the process actors raise

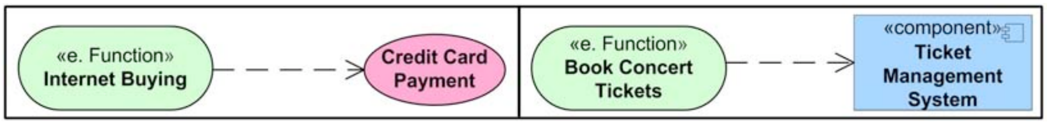


Figure 5.1: Dependency Relationship with a Use Case and a Component

basic system requirements from a process perspective. The designer of a business process has now the possibility to check which tasks of a specific business process depend on a specific software requirement.

A dependency between an elementary function and a component is shown at the right hand side of figure 5.1. The component represents the supplier, and the elementary function the client. The elementary function *Register Participant* needs the component *Participant Management System* for its full execution. The utilization scenario for the relationship with a component is intended to describe the interaction between stereotyped actions and concrete software systems, either existing ones or to be developed ones. The example visualizes that a certain software application is required by a stereotyped action.

Figure 5.2 presents the *processing of automobile insurance claims* business process and its dependencies with stereotyped actions and use cases as well as software components. The process starts with the stereotyped action *record the claim*, which requires the component *claim management system* as well as the use cases *check policy* and *formulate claim description* for its execution. Furthermore, the *claim management system* is also needed by the stereotyped actions *calculate the insurance sum* and *examination of results*.

The ≪Organisation Role≫ *Financial Claim Expert* is responsible for the execution of *record the claim* and calculate the insurance sum. The component *claim management system* saves, destroys, and archives the different insurance claims. To start the process, the stereotyped action *record the claim* needs access to the *claim management system* as well as to the the software requirements *check policy* and *formulate claim description*. After the claim is recorded, the software requirement *proof of documents* is necessary for *calculate the insurance sum*. Now the ≪Organisation Role≫ *Financial Clerk* has to decide wether the insurance company pays for the damage or not. If the insurance sum represents a major amount, then the stereotyped action *checking history of the customer* is necessary. This action also requires the customer relationship management (CRM) system. All relevant data from the customers of the insurance

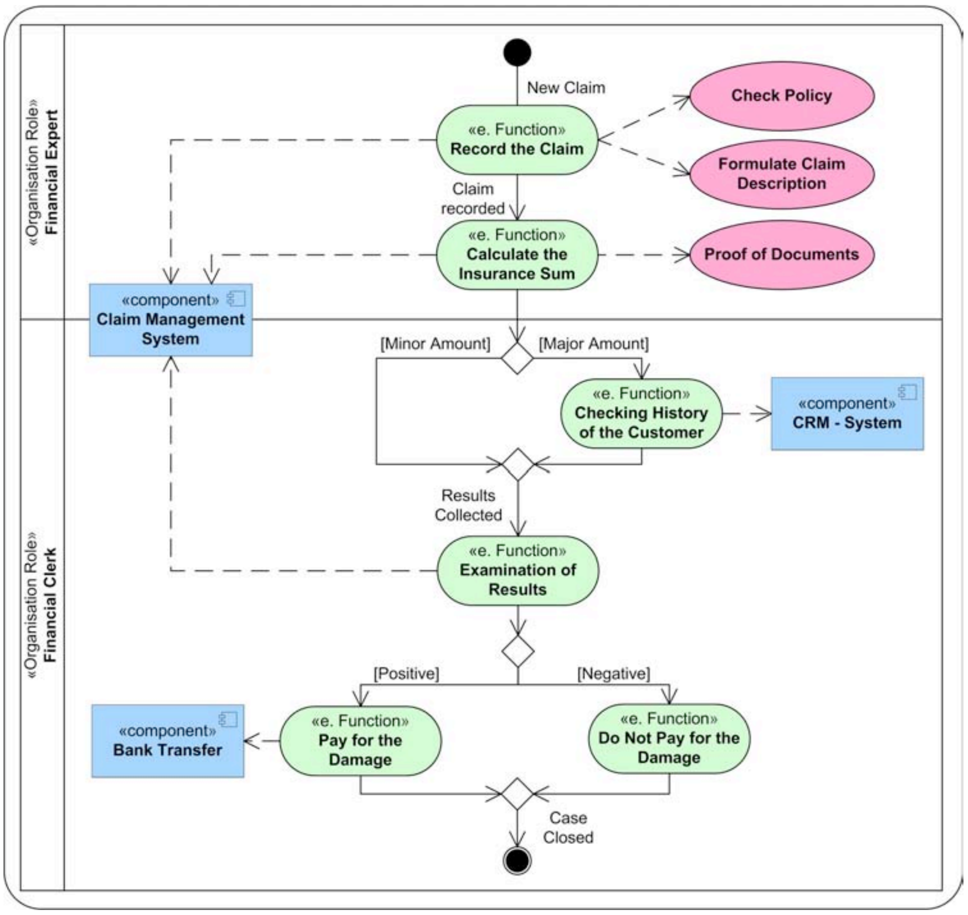


Figure 5.2: Processing of Automobile Insurance Claims

company are saved at the CRM system. After checking the history of the customer, the stereotyped action *examination of results* needs all relevant files of the claim from the *claim management system* for the decision wether the company pays for the damage or not. Finally, if the action *examination of results* is positive, the bank transfer component is used by the stereotyped action *to pay for the damage*, otherwise the action *do not pay for the damage* is processed. The process ends with a closed case. The first part of the process shows how software requirements are integrated into the process model. The second part of the process illustrates which stereotyped action depends on a specific software component for its execution.

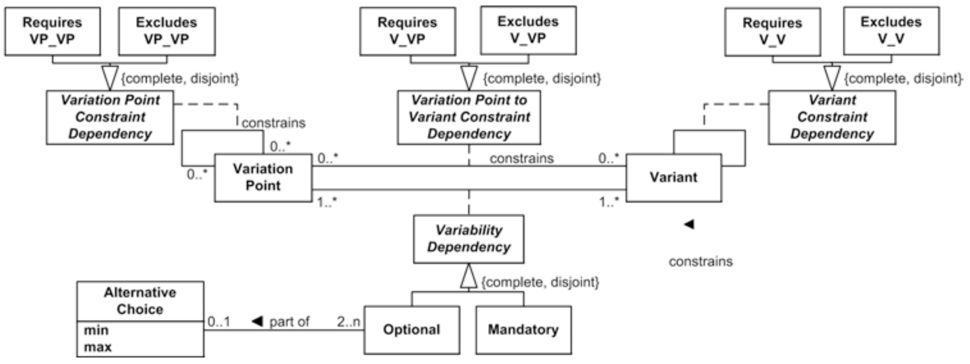


Figure 5.3: Variability metamodel

5.3 UML Profile for Variability Models and their Interdependencies with Business Processes

5.3.1 Variability Modelling

A variability model shows the different variation points and variants of a software product line. Variability models are based on a metamodel developed by Pohl et al. [PBvdL05]. This metamodel is not MOF-compliant, because it has association classes which are not part of MOF. This is shown in figure 5.3. We adapt the metamodel of Pohl et al. [PBvdL05] by changing the association classes and its associations with binary relationships and classes to make it MOF-compliant according to Hitz et al.[HKRK05]. The MOF-compliant metamodel could be for instance easily integrated into UML, which is based on MOF. The adapted metamodel is shown in figure 5.4.

A variability model consists of the variabilities *variation point*, *variant*, and the relationships between them. A variation point is a representation of a variable item of the real world or a variable property of such an item. It has a *variability dependency* relationship with at least one variant. A variant is a representation of a particular instance of a variation point. It may constrain it by a constraint dependency. Furthermore it has at least one variability dependency to a variation point. A variability dependency is an abstract class. We distinguish between optional or mandatory dependencies. If a variation point has a *mandatory* dependency to a variant, the variant has to be chosen when the variation point is selected. An *optional* variability

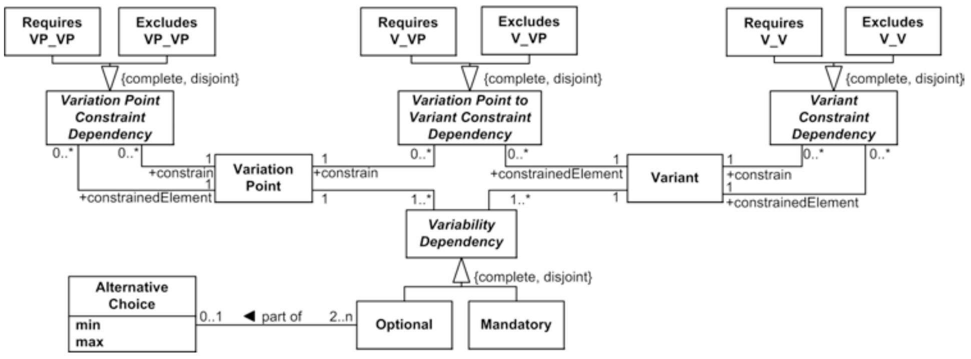


Figure 5.4: MOF-compliant metamodel of the variability metamodel

dependency indicates that none, one or more variants may be selected. Moreover, it may be refined by an *alternative choice*, which declares the range between a variation point and its variants. The *constraint dependency* distinguishes an *requires* and an *excludes* dependency between variants, variation points and variants to variation points. An *requires* constraint dependency indicates that a variability is dependent on another variability. An *excludes* constraint dependency defines that a variability has to eliminate another variability if it is selected.

5.3.2 UML Profile for Variability Models

In this subsection the UML 2 profile for Variability Models is described. As stated in subsection 5.3.1, variability modelling is a domain specific modelling technique, that has no MOF-compliant metamodel for integration in a modelling framework, and has no tool support yet. The Meta-Object-Facility (MOF) is the four-layered metamodelling architecture of the OMG [OMG05a]. If a modelling technique is based on a MOF-compliant metamodel, then a UML Profile may be easily created, because UML offers a mechanism to extend and adapt its metamodel to a specific area of application. A profile extends a metamodel or another profile [OMG05c] while preserving the syntax and semantic of existing UML elements. It adds elements which extend existing classes, and consists of stereotypes, constraints and tagged values. A stereotype is a model element defined by its name and by the base class(es) to which it is assigned. Base classes are usually metaclasses from the UML metamodel, but may also be stereotypes from another profile. In this profile the different stereotypes are based on the metaclass "Class". Furthermore, a stereotype may have its own notation, e.g., a special icon.

5.3.2.1 Extended metamodel with variabilities

A UML 2 class diagram describes the structure of a system that needs to be designed. It shows the main static properties as well as the relationship among each other. However it has no mechanism showing the variabilities of a system. Therefore, a subset of the metamodel for class diagrams is used as metaclasses to define a UML 2 profile for Variability Models introducing variability concepts in UML. Figure 5.5 illustrates a section of the UML metamodel for Classes and its extension with stereotypes for representing variation points, variants, and their relationships among each other. The triangle at associations marks the direction of reading a relationship between the metaclasses. The stereotypes are shown as grey rectangles. Furthermore we use the icons of the variability model of Pohl et al. [PBvdL05] for presenting the UML profile graphically.

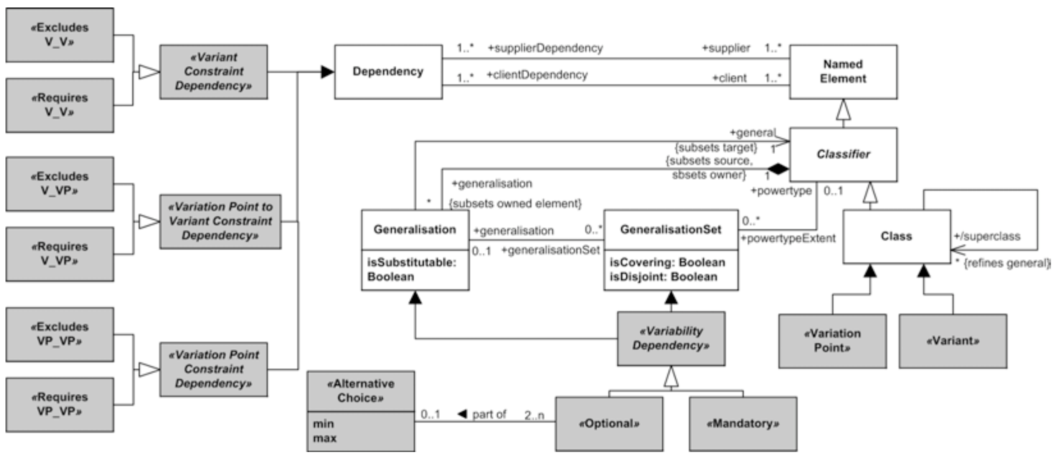


Figure 5.5: Extended UML Class Metamodel with Variabilities

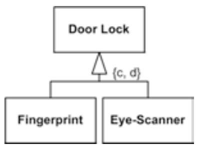
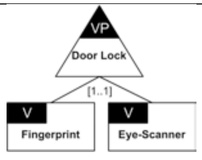
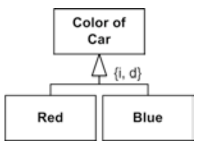
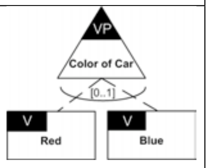
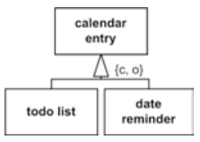
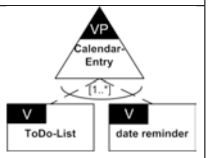
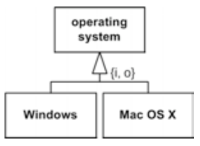
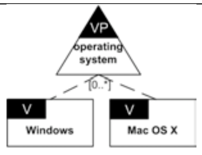
5.3.2.2 Description of stereotypes

In a variability model a ≪*variation point*≫ marks out the different sets of options of a model. The OMG defines a class as "a set of objects that share the same specifications of features, constraints, and semantics. The purpose of a class is to specify a classification of objects and to specify the features that characterize the structure and behavior of those objects." [OMG05c]. A class is the appropriate metaclass for describing the stereotype ≪variation point≫ and its characteristics, because a variation point describes common properties of instances. Furthermore a subclass is a child from another class in a generalization relationship. A subclass inherits the structure, relationship and behaviour of its superclass and may add to it. The stereotype ≪*variant*≫ is a representation of a particular instance, and thus it represents an extension of the subclass of a class.

The different variability dependencies of a variability model, namely ≪*mandatory*≫, ≪*optional*≫, and ≪*alternative choice*≫ extend the metaclasses *generalisation* and *generalisationSet* to describe these different types of stereotypes. Table 5.1 shows the different characteristics of variability dependencies with their multiplicities. Moreover, table 5.1 shows how the variabilities are described with the appropriate *generalisationSet* corresponding to the UML profile including examples for each type. A *generalisation* is defined as a taxonomic relationship between a more general classifier and a more specific classifier, for example from a class to its superclass [OMG05c]. Each instance of the specific classifier is also an indirect instance of the general classifier. Thus, the specific classifier inherits the features of the more general classifier. A *generalisationSet* defines a specific set of generalisation relationships. The metaclass *generalisationSet* describes how a general classifier (or superclass) may be divided into specific subtypes. Furthermore it has two metaattributes with boolean values, namely *isCovering* and *isDisjoint*. If *isCovering* is true, the generalisation set is complete, otherwise it is incomplete. If *isDisjoint* is true, the generalisation set is disjoint, otherwise it is overlapping. When *isCovering* is true, every instance of an superclass has to be an instance of at least one of its subclasses at the same time. If *isDisjoint* is true, every instance of a superclass corresponds only to one subclass. The overall multiplicity for the couple {complete, disjoint} is 1. This is equal to the multiplicity of the mandatory relationship. This means if a variant has a mandatory relationship with a variation point, the variation point has to select the variant.

The example in table 5.1 shows that a door lock can only be opened if a fingerprint and an eye-scan are accomplished. If isCovering is false, then a superclass can have more instances that do not correspond to the declared subclasses and the overall multiplicity for {incomplete, disjoint} is 0 to 1. This means that for example the color of a car may be either red or blue, or may have a different color. The alter-

Table 5.1: variability dependency and generalisation set

var. dep.	mult.	generalisation set	Class Diagram	UML Profile
Mandatory	1	{complete, disjoint}		
Alternative	0..1	{incomplete, disjoint}		
Alternative	1..*	{complete, overlapping}		
Optional	0..*	{incomplete, overlapping}		

native choice expresses this fact, because in such a relationship a group of variants may but does not need to be a part of the business process. If isDisjoint is false, an instance of a superclass may have more then one related subclasses. The combination {complete, overlapping} is on a par with the alternative choice with the multiplicity 1 to *. This means a variation point has to select at least one variant. For example, a calendar entry can be a todo list, a date reminder or both. The optional variability dependency defines that a variation point may select none, one or more variants, which conforms to the multiplicity 0..*. The generalisation couple {complete, overlapping} matches to the optional variability. An example is that an operating system installed on a computer may be Win XP or Mac OS X, both of them, or a different one.

The variability constraint dependencies ≪requires≫ and ≪excludes≫ between the stereotypes ≪variation point≫ and ≪variation point≫, ≪variation point≫ and ≪variant≫, as well as ≪variant≫ and ≪variant≫ are defined by the metaclass dependency. A dependency denotes that an element, called client, is dependent on another element, called supplier. The client depends on the model element of the supplier, e.g. for its specification or implementation. Dependencies are shown as dashed arrows, whereas the model element at the tail of the arrow (the client) depends on the model element at the arrowhead (the supplier). The ≪requires≫ stereotype defines that a client needs the consideration of a supplier. The ≪excludes≫ stereotype declares that a client excepts the consideration of a supplier.

5.3.3 Constraints

The tables 5.2, 5.3, 5.4, 5.5, and 5.6 show the different constraints in OCL with explanations in natural language for the stereotypes ≪Variation Point≫, ≪Variant≫, ≪Mandatory≫, ≪Optional≫, ≪Alternative Choice≫, ≪Requires≫ and ≪Excludes≫.

The stereotypes ≪Variation Point≫ and ≪Variant≫ are described in table 5.2. Both stereotypes are based on the metaclass *Class*. However, both stereotypes need different characteristics of their baseclass. ≪Variation Point≫ reflects a superclass, while a ≪Variant≫ represents a subclass, because a ≪Variant≫ is a specialisation of a ≪Variation Point≫.

≪Mandatory≫, ≪Optional≫ and ≪Alternative Choice≫ describe the different relationships between a ≪Variation Point≫ and a ≪Variant≫. The baseclass of these three stereotypes is *GeneralisationSet*. The baseclass has two attributes, namely *isCovering* and *isDisjoint* with boolean value. If a variation point and a variant are connected by a mandatory relationship (see table 5.3), then the attributes have the value complete and disjoint according to the UML 2 Superstructure [OMG05c].

An optional variability dependency states that a variant can but does not need to be a part of a variability model. The dependency is described by the couple {incomplete, overlapping}.

An alternative choice as shown in table 5.5 offers the possibility to define the minimum and the maximum number of optional variants to be selected from a given group of variants. The couple {incomplete, disjoint} describes the multiplicity 1, and the couple {complete, overlapping} the multiplicity 1 to many.

≪Requires≫ and ≪Excludes≫ describe the different meanings of their baseclass *Dependency*. If ≪Requires≫ is chosen, a variability is required. In contrary, if ≪Excludes≫ is chosen, then a variability is excluded.

Table 5.2: Stereotype Definitions for ≪Variation Point≫ and ≪Variant≫

Name	**Variation Point**
Base class	Class
Description	A variation point is a representation of a variable item of the real world.

Constraints	A variation point must have a superclass as a baseclass which is a role of a class.
	`context Variation Point inv:`
	`class.superclass=true implies variation point`

Name	**Variant**
Base class	Class
Description	A Variant is a representation of a particular instance of the stereotype variation point.

Constraints	A variant must not have a superclass as a baseclass which is a role of a class.
	`context Variant inv:`
	`class.superclass=false implies variant`

Table 5.3: Stereotype Definition for ≪Mandatory≫

Name	**Mandatory**
Base class	GeneralisationSet
Description	A variant must be selected for an business process if and only if the associated variation point is part of the business process.

Constraints	If the variability dependency is mandatory, then metaattributes isCovering and isDisjoint of the metaclass GeneralisationSet are true.
	`context Variability Dependency inv:`
	`if self.Mandatory=true then`
	`self.GeneralisationSet.isCovering=true and`
	`GeneralisationSet.isDisjoint=true`

5.3.4 Showing the dependency between Variability Models and UML 2 Activity Diagrams

Variability models show the different variabilities of a software. Activity Diagrams are a part of the behavioural set of UML 2 diagrams, and are used for modelling

Table 5.4: Stereotype Definition for ≪Optional≫

Name	**Optional**
Base class	GeneralisationSet
Description	A variant can but does not need to be part of a business process.
Constraints	If the variability dependency is optional, then metaattributes isCovering and isDisjoint of the metaclass GeneralisationSet are false. `context Variability Dependency inv:` `if self.Optional=true then` `self.GeneralisationSet.isCovering=false and` `isDisjoint=false`

business processes as well as for describing control flows in software.

On the one hand side variability models show the different variabilities of a software, and on the other hand side UML 2 Activity Diagrams show the control and data flow between different tasks according to the software system an activity diagram has to describe. It follows that the two modelling techniques focus on similar concepts from different perspectives. Accordingly, a mapping between these metamodels to find out in which way they are related to each other is needed. Table 5.7 shows the related elements, with the variability metamodel on the left hand side and the activity metamodel on the right hand side.

A *variant* is mapped to an action, because both describe the atomic tasks of their metamodel. Due the fact that an activity partition is a kind of activity group for identifying actions that have some characteristic in common, it is mapped to a *variation point*, because it describes common properties of variants.

The *optional* variability dependency defines that a variation point may select none, one or more variants. Therefore this variability dependency maps to a fork node which splits one incoming flow in several concurrent outgoing flows and a join node, which synchronizes the several incoming flows. Additional guards on the flows indicate wether none, one, or more paths are selected. The additional guards are needed in our mapping, because if in an activity diagram no path is selected, the business process is terminated. An user must define an additional control flow to overcome this gap.

The mapping of the *alternative choice* to an appropriate element of the activity diagram depends on the range of the variability dependency. If the range is 1 to

Table 5.5: Stereotype Definition for ≪Alternative Choice≫

Name	**Alternative Choice**
Base class	GeneralisationSet
Description	The alternative choice groups a set of variants that are related through an optional variability dependency to the same variation point and defines the range for the amount of optional variants to be selected for this group.
Tagged V.	min, max Type: UML::Datatypes::Integer Multiplicity: 1 Description: Min and max stands for the range of a relationship between variation points and variants.

Constraints	If the variability dependency is an Alternative Choice and the range is or smaller then 1, then the metaattribute isCovering is false and isDisjoint is true of the metaclass GeneralisationSet. Else if the tagged values of the alternative choice are min = 1 and max = n, then then the metaattribute isCovering is true and isDisjoint is false of the metaclass GeneralisationSet.
	`context Variability Dependency inv:`
	`if self.Alternative Choice=true and size()=<1`
	`then self.GeneralisationSet.isCovering=false and`
	`isDisjoint=true`
	`else if self.AlternativeChoice=true and`
	`self.AlternativeChoice.min=1 and`
	`self.Alternative.max=n then`
	`self.GeneralisationSet.isCovering=true and`
	`GeneralisationSet.isDisjoint=false`

*, then the *alternative choice* is mapped to a fork node starting concurrent flows, and closed by a join node for sychronisation of the incoming flows. Furthermore additional guards on the flows specify if one or more paths will be selected. If the range is 0 to 1, then the alternative choice is mapped to a decision node with guareds ensuring that at most one outgoing edge will be selected. Furthermore, a merge node brings together the alternate flows.

The *requires* and the *excludes* constraint dependency are both mapped to the con-

Table 5.6: Stereotype Definitions for ≪Requires≫, and ≪Excludes≫

Name	**Requires**
Base class	Dependency
Description	The selection of a variability requires the selection of another variability.
Name	**Excludes**
Base class	Dependency
Description	The selection of a variability excludes the consideration of another variability.

Table 5.7: Dependencies between Variability Models and UML 2 Activity Diagrams

Left MM Class	*Right MM Class*
Variation Point	Activity Partition
Variant	Action
Mandatory	Fork Node - Join Node
Optional	Fork Node - Join Node
Alternative Choice [1..*]	Fork Node - Join Node
Alternative Choice [0..1]	Decision Node - Merge Node
requires	Control Flow
excludes	Control Flow - Decision Node

trol flow, independent from the fact if the dependency is between variants, variation points, or variants and variation points, because these elements are mapped as described above. Furthermore the *excludes* constraint dependency needs for its mapping to activity diagrams a decision node that the action that should be avoided cannot be executed.

5.3.5 Applying the UML Profile to an Example Business Process

The UML profile for variability models and the mapping to activity diagrams is shown in figure 5.6 with an example business process for an easier understanding of our approach. This example business process is part of the inventory process of a real international logistic company.

The variation point *inventory accomplishment* distinguishes between two types of inventory, *permanent inventory* and *periodical inventory*. An inventory accomplishment may be a permanent or a periodical inventory or both of them, when the two procedures overlap. Our example process focuses on the variant of the periodical inventory. It requires the variation *point base of inventory*, which has a mandatory relationship to the variant *inventory requirements*. The inventory requirements need the variation points *inventory records*, *behaviour of logistics execution* and *generation of appointment* for further processing. For the generation of appointment the *number of positions* of the inventory goods may, but need not be used. The behaviour of the logistics execution depends on wether a *printing device* or an *electronic device* or both of them is required. The variant printing device needs a special type of an *inventory record*, namely *list*, and the variant electronic device needs *radio* for further processing.

As the activity diagram example on the left hand side of figure 5.6 shows, every variation point of the right hand side of figure 5.6 has its corresponding activity partition, and every variant has its corresponding action. The whole business process is covered by the activity partition inventory accomplishment, which corresponds to the root variation point of the variability model. The activity diagram starts with the action *creating periodical inventory work order*, followed by *examining inventory requirements*. Both are part of the activity partition *base of the inventory*, because the action *creating periodical inventory work order* requires the *base of the inventory*. If it is selected, then the *inventory requirements* continues with further processing. After that one may choose whether the *input* of the *number of positions* of the inventory goods is needed or not, which is part of the *generation of appointment*. Afterwards the process continues with the decision wether the *behaviour of logistics execution process* proceeds with the *processing the inventory charge*, with *printing devices*, or *electronic devices* or both of them. The process ends with the *processing of the inventory records*. If the inventory was made with radio support, the action *acquire of manual or electronic inventory records* executes. Else the action *acquire of manual inventory records* finishes the business process. But if the inventory was made with radio and list support, then both actions execute.

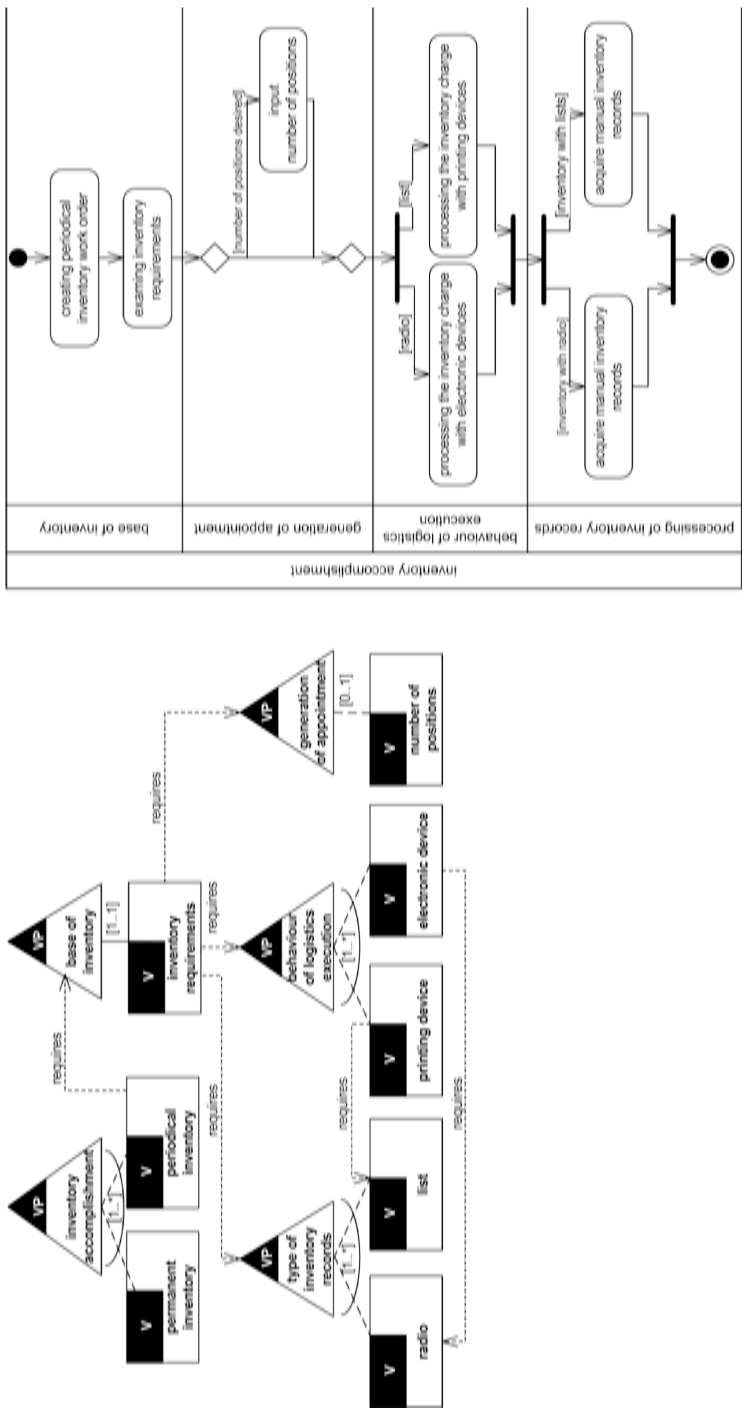

inventory accomplishment
base of inventory
generation of appointment
behaviour of logistics execution
processing of inventory records
creating periodical inventory work order
examing inventory requirements
[number of positions desired]
input number of positions
[list]
[radio]
processing the inventory change with printing devices
processing the inventory change with electronic devices
[inventory with lists]
[inventory with radio]
acquire manual inventory records
VP
V
permanent inventory
periodical inventory
[1..1]
[1..*]
[0..1]
requires
inventory requirements
number of positions
type of inventory records
radio
list
printing device
electronic device

Figure 5.6: Example of a UML Profile for Variability Models and the Mapping to Business Processes

6 Related Work

6.1 Introduction

Contents

There are a lot of approaches in the wide area of business process modelling languages. Their main goal is always extending and improving existing languages. Before an extension is possible, an evaluation is needed for knowing which parts of a BPML have the potential for improvement. There exist many frameworks with different perspectives for evaluating a language. On the one hand side the authors develop a metamodel and compare the languages with this metamodel. On the other hand, these frameworks are based on metamodels. On the other hand, they are based on a pattern approach. A summary of the most relevant evaluation concepts is presented in subsection 6.2.

Business process performance measurement is an important topic in research and industry, but the main focus of conceptual BPMLs is organizing the flow with its tasks of a business process, but not the explicit modeling of process goals and their measures (see subsection 2.4) of a language. An excerpt of approaches to integrate business process goals and their measures in a BPMLs is shown in subsection 6.3.

Although business processes are often the starting point for software development and define requirements for software systems, linking business processes with software elements is inadequately supported in conceptual modelling. On the on hand side it is not possible to show software requirements or components which have an influence on a BPML. On the other hand side it is not possible to show the variabilities of a BPML. A section of relevant approaches that want to solve these gaps is presented in subsection 6.5.

6.2 Evaluation

A number of publications on the evaluation of BPMLs is available. They evaluate a very limited number of BPMLs. The evaluation concepts are mainly based on metamodels representing a very technical perspective. We address these limitations with a comprehensive metamodel and the evaluation of six state-of-the-art BPMLs.

Söderström et al. developed a generic metamodel for comparing BPMLs in [SAJ$^+$02]. The metamodel shows technical concepts of business processes, and captures a definition and an execution level similar to workflow management systems. Events and control nodes are defined in detail, but roles and resources are described at a very high level. The paper compares only three different BPMLs: the EPC, the UML 1.3 State Diagram and the Business Modelling Language, the BPML of a commercial tool.

The authors in [SAJ$^+$02] declared that the increasing interest in process engineering and application integration has resulted in the appearance of various new process modelling languages. To solve the problems of understanding and comparing the high amount of different languages, Söderström et al. developed a generic metamodel. The framework is useful for several purposes, for instance for translating between languages or verifying that certain aspects of an language have been considered. Three different process modelling languages have been compared for validating the framework. These Languages are the Business Modelling Language (BML), the Event-driven Process Chains (EPC) and UML State Diagrams.

The framework consists of a general process metamodel, an analysis of the event concept, and a classification of concepts according to the interrogative pronouns: what, how, why, who, when, and where.

The main elements of the framework are activity, state, event, and time point. They are defined as follows: An activity is a performance of a task and can change a state. A time point is an instant in time, and not further decomposable. An event is a occurrence. The focus lies on event, it connects states and activities in time. The authors discusses three different event types, namely time point events vs. time duration events, pre-activity events vs post-activity events and state change events vs. no state change events. Furthermore Söderström et al. argued that different types of events are used in different BPMLs.

The metamodel is divided in two ways. On the one hand side the metamodel distinguishes between a type and an instance level. On the other hand side the metamodel elements are separated through the interrogative pronouns. On the type level, we find activity, resource, role and logical dependencies between activities. On the instance level, we find event, state, actor, and temporal dependencies between events.

Compared to our generic metamodel, the metamodel of Söderström et al. is not presented in a well known modelling language, like UML 2 class diagrams. For describing their metamodel the authors used an own modeling notation which is not clearly defined in their work. Therefore the taxonomic relationships between the metaclasses are not obvious. Furthermore events and control nodes are defined in detail, but roles and resources are described at a very high level.

Lin et al. analysed 10 BPMLs in [LYP02] and derived eight generic concepts: activity, resource, behaviour, event, information, relation, agent and entity. This bottom-up approach requires more detail for an evaluation of BPMLs, as the basic concepts are represented in all BPMLs.

The authors proposes a generic structure for modeling business processes. Lin et al. compares various BPM (Business Process Modeling) methods in order to elicit generic features from these BPM methods. Furthermore the authors developed a generic BPM method, which utilizes essential components of process representation and incorporate other functions. The generic structure offers two main features. At first, it represents a business process in various concerns and multiple layers of abstraction. Second, it lowers the barriers between process representation and model analysis by embedding verification and validation with the model. Lin et al. examined 10 BPMLs for detecting the main BPM methods for those languages. The languages were IDEF0, IDEF1, IDEF1X, IDEF3, RAD, REAL (acronym for resources, events, agents and locations), Dynamic Modeling, Object- Oriented Modeling (OO), AI and MAIS (multi-agent information system). The essential concepts the authors identified are: activity, behaviour, resource, relation, agent, information, entity, event, verification and validation.

The authors compared the BPML in two ways. On the one hand side they compared if the languages fulfill the perspectives of Curtis [CKO92]. Furthermore Lin et al. extended the perspectives with verfication and modeling procedures. On the other hand side the methods of the languages were compared against the perspectives of Curtis.

This bottom-up approach requires more detail for an evaluation of BPMLs, as the basic concepts are represented in all BPMLs. Therefore the languages are examined on a high level. Furthermore the authors did not examine well known languages, like UML 2 AD or EPC.

UML 2 Activity Diagrams are evaluated by **Wohed et al.** based on workflow control flow patterns in [WvdAD+05]. These patterns are very detailed by nature and focus on the execution of business processes. Concepts that target business users, like traditional resources or goals are not addressed at all.

Wohed et al. examined the Activity Diagrams notation for collecting patterns. The patterns were used for evaluating control flow capabilities of BPMLs. The analysis tries to find out the strengths and weakness of control flow specifications in Activity Diagrams. Furthermore the pattern-based analysis detect some of the ambiguities in the current UML 2.0 specification.

For providing a fine-grained analysis the authors have chosen a specialised evaluation framework, namely the workflow patterns of Aalst et al. In this analysis YAWL (Yet Another Workflow Language) is used as a reference realisation of the patterns (where appropriate). YAWL provides a reference formalisation for the control-flow patterns.

The first seven control-flow patterns, namely Sequence, Parallel Split, Synchronisation, Exclusive Choice, Simple Merge, Multiple Choice, and Multiple Merge are directly supported in UML AD. The first five of these patterns are supported by basically all process modelling and description languages and they correspond to control-flow constructs defined by the Workflow Management Coalition [Wor98]. Furthermore the Deferred Choice, Cancellation Patterns, Structural Patterns, Arbitrary Cycles and Implicit Termination are captured in UML 2 AD. The Synchronising Merge pattern and the Milestone pattern are not supported. Furthermore Multiple Instances Patterns, Interleaved Parallel Routing and Discriminator are indirect supported.

The pattern analysis of Wohed et al. is very detailed by nature and focuses on the execution of business processes. Concepts that target business users, like traditional resources or goals are not addressed at all.

Mendling et al. address the heterogeneity of business process interchange formats in [MNN04]. The authors argued that a commonly accepted interchange format is needed to import business process into different tools.

The work of Mendling et al. identified the superset of high-level concepts covered in metamodels of various proposals for interchange formats. From the analysis of fifteen specifications they gathered the following list of thirteen high-level metamodel concepts: Task I/O, Task Address, Quality Attributes, Task Protocol, Control Flow, Data Handling, Instance Identity, Roles, Events, Exceptions, Transactions, Graphic Position, and Statistical Data.

The thirteen metamodel concepts are compared with fifteen different business process modelling interchange format proposals. A couple of these interchange formats are BPEL4WS, BPMN, BPSS (Business Process Specification Schema) and WSCI (Web Service Choreography Interface). The interchange formats are used in at least four different areas of application, namely Composition, Choreography, Business Analysis, Formal Analysis.

The authors proposed their classification as a framework for comparing the completeness of BPM interchange formats. It serves as a first step towards a reference model for BPM that unifies the different perspectives. Though the metamodel elements are very technical (e.g. instance, identity, events, exceptions, transactions, task address) and have a strong focus on the execution of business processes.

6.3 Performance Measures in Business Processes

The related work discusses the different approaches that focus on different aspects of the quantification of performance measures and business process goals.

Aguilar et al. [ARGP06] developed a set of measures to evaluate the structural complexity of business process models on a conceptual level. The authors use the Business Process Modeling Notation (BPMN) [OMG06a] for their evaluation. The base measures consist of counting the business process models significant elements. The work is based on the proposal FMESP (Framework for the Modeling and Evaluation of Software Processes). The proposal defines a set of measures to evaluate software process models at two levels. The first level models the scope to evaluate the overall structural complexity of the model. The second level models the core elements scope, to evaluate the concrete complexity of fundamentals elements such as activities, roles or work products. In FMESP the main goal is to choose a set of indicators that are useful in the maintenance of software process models by the evaluation of their structural complexity. In order to achieve that goal the authors adapted and extended FMESP to business process models, which are represented with BPMN. Furthermore when all measures are defined, it is possible to evaluate the structural complexity of business process models developed with BPMN. The evaluation of performance measure like time or costs is not important for their work, the focus lies on measuring the core elements of BPMN.

The approach of **Vitolins** [Vit04] is based on metamodeling according to the Meta Object Facility (MOF) [OMG05a]. The author provides a metamodel with typical process measures for UML 2 activity diagram-like notation. The work of Vitolins is defined by different abstraction layers, which define aspects of business process measures. The M1 layer is represented by an activity diagram extended by measures as well as a class diagram defining the measure view of the same example. M2 presents the metamodel which shows common possibilities and features for business process models. At M3 the meta-metamodel outlines a universal framework for measure definition. Furthermore the metamodel at M2 can be used as an instance of M3. The author argued that the metamodel can be used as a unified framework for the development of comprehensive business modelling and measurement tools. As a contrast to our work, the author annotates cost and time to each action separately as a note. Therefore the activity diagrams looses its clarity and explicitness. Furthermore there are no considerations to integrate the performance measures more tightly into the activity diagrams.

Nurcan et al. [NEK+05] adopted a goal-perspective, the map-driven process modelling approach, to master the complexity of process modelling. The map-driven process modelling approach defines a business process in terms of gaols and strategies of reaching these goals. The map representation system conforms to goal models in the fact that it recognises the concept of a goal but departs from those by introducing the concept of strategy to attain a goal. Each strategy can be refined by representing it in goal-strategy terms of the next level of details. Furthermore the authors adopted the two levels hierarchical spiral process model. The reason for that is to support the propagation of the intentional changes onto operational ones, since a business and its supporting system change in a concurrent way. The intentional spiral deals with the production of the business process models using the map formalism. The operational spiral deals with the specifications of the supporting systems. The work of Nurcan et al. is situated on a higher level than our work. While Nurcan et al. adopted a common approach, namely the map-driven process modelling approach for mastering the complexity of process modelling, we integrate business process goals and their measures in concrete BPMLs.

Neiger et al. [NC04] focus on the problem that business process management frameworks are able to represent various aspects of the business process, but they do not meet the requirements of goal-oriented business process modelling. In a previous work the authors examined the i* framework, the "value focused thinking" (VFT) framework and EPC to derive the requirements of goal-oriented business process modelling of these modelling technics. A conceptual model that describes theses requirements at an intuitive level was proposed in [NC03]. In this work the authors introduce a formal syntax for describing the EPC and the VFT goal model. As a next step, the authors establish links recorded in the formal syntax between EPCs and its additional goals with the VFT framework to address the gaps in the existing frameworks and tools. Compared to our work, Neiger et al. focus on business process goals, without looking at the measurement of the goals.

Anderson et al. [ABJP05] developed a formal definition of goal-oriented business process patterns for making a formal comparison of business processes. The paper analyzes widespread modelling technics to discover which of them are able to build patterns for comparison. Based on this analysis the state-flow modeling technique is chosen as a foundation for defining business process patterns. In this approach a business process described as a trajectory in a specially constructed multidimensional state space, and not as sequence of activities. Therefore a pattern is defined as a three component entity: state space, goal, and valid movements in state space. This approach is very high level, because the authors focus on business processes, and not on a specific business process modeling language.

6.4 Mapping BPMLs to BPEL

Bordbar et al. [BS04] present a transformation of the UML 2 Activity Diagram to BPEL. The authors show the behavioural aspects of web services by using a metamodel based on MOF [OMG05a] for BPEL. The work presents model transformation by example. The UML Activity Diagram is the source metamodel, and the target model is BPEL. First, an UML 2 activity diagram example of a Stock Quote Process is shown. Second, the authors refined the example diagram with stereotyped actions from the UML 2 superstructure to define the diagram more precisely. Third, the stereotyped actions are mapped to its conforming BPEL tags. Since the example did not include all BPEL tags, the mapping is not completed. The mapping is made manually by a table and the transformation rules are expressed in OCL. This work used OCL as a transformation language adapted from [KWB03] at a time where no standard language for writing transformation definition existed. We use in section 4 the mapping from the stereotyped actions to the BPEL-tags as a basis for our transformation.

Gardner et al. [GAGI03] show a UML Profile for Automated Business Processes which enables BPEL processes to be modelled using an existing UML tool. The authors introduced a UML profile which supports modelling with a set of semantic constructs that correspond to those in the Business Process Execution Language for Web Services. Furthermore the authors describe a mapping to BPEL to automatically generate web service artefacts (BPEL, WSDL, XSD)from a UML model meeting the profile. The approach enables service-oriented BPEL4WS com- ponents to be incorporated into an overall system design utilizing existing software engineering practices. The mapping from UML to BPEL4WS permits a model-driven development approach in which BPEL4WS executable processes can be automatically generated from UML models. This work is out of date, because of the old UML version 1.4 and BPEL 1.0.

6.5 Linking Software Systems with Business Processes

There are quite a lot of conceptual Business Process Modelling Languages (BPMLs) and UML profiles for business process modelling available. They focus primarily on the sequential flow of the business process and do not integrate software elements (see subsection 2.4).

Tyndale et al. [BSWS02] focus in their work onto the COMBINE Project. This project has the goal to improve software development productivity by providing a holistic approach to component-based development of Enterprise systems. The authors want to reach their goal by identifying a coherent minimum set of modelling concepts and discovering a way of representing these in UML. A business model in the COMBINE project consists of seven work products. Context statement, vision statement, risk analysis belong belong to the informal work products. Goal model, business process and role model, business resource model as well as business component model are a set of linked rigorous models expressible in UML 1.4. The paper describes a conceptual business metamodel which contains a minimum set of metaclasses to describe the seven work products and their relations. Based on that metamodel, Tyndale et al. develop a UML profile. It shows the integration of business processes and software development from an industry perspective by mapping business concepts to software artefacts.

The UML profile for Business Modelling in [SVC^{+}01] of **Sinogas et al.** is focusing on the integration of business processes into software development. The profile maps between business concepts and software artefacts. This profile is located at the level M2, and it is an end user specific profile. The UML Profile for Business Modeling present in UML 1.4 uses actors and introduces the stereotypes worker, case worker, internal worker and entity. Therefore, the profile describes the process flow in a very a detailed way, and in addition it adds processes, goals, and resources. The global metamodel which serves as a foundation for the profile shows the relation of the model elements. It is possible to decompose processes, goals and resources. The process stereotype is also connected to an activity state to allow the modeler to express parts of the business process in a UML Activity Graph. Compared to our approach the UML profile from Sinogas et is presented from a high level, since it lacks in integrating more advanced concepts like customers, process types or process owners. Furthermore a possibility to measure the business process is not integrated in the profile.

There are a lot of approaches in the global area of variability modelling available, and a few of them will be presented now.

Rosemann et al. [RvdA07] want to solve the problem that most Enterprise Systems (ES) solutions provide reference models that describe the functionality and structure of the system, but they do not capture the potential configuration alternatives. The authors define reference models as reusable conceptual models that depict recommended structures and processes. Rosemann et al. proposed in their work configurable Event-Driven Process Chains as an extended reference modelling language. This approach allows to capture the core configuration patterns of Dreiling et al. [DRvdA^{+}07] by introducing configurable elements into the Event-Driven Process Chains. Furthermore the approach of Rosemann et al. allows that the proposed extensions may be easily adapted to other modelling techniques (e.g. UML or Petri Nets). The difference between our approach and the approach of Rosemann et al. is, that we do not want do describe the configurability of a certain business process modelling language, because we want to show how the configurability of a software process can be mapped to its business process, which describes the control flow of the software.

Clauss [Cla01] address to the problem that on the one hand side modeling variabilities is a key element in developing product families and product lines, but there is no uniform and consistent notation for modeling variabilities and commonalities available. On the other hand side UML is able to model a wide range of software systems, but it is not useful for modeling groups of related systems which are for product families. The author introduces a UML extension to support feature diagrams which are an extension for the explicit representation of variation points, and adds elements describing variability in the standard kinds of UML diagrams. The extensions use the standardized extension mechanisms of UML. Therefore the extensions are fully compatible with the standard and enable a broader usability. The distinction between our work and the work of Clauss is that Clauss integrates feature models in UML diagrams like the Use Case Diagram, whereas our approach extends the UML metamodel to integrate variability models as a independent modelling language into UML.

Becker argued in his work [Bec03] that the increasing amount of variability in software systems leads to a situation where the complexity of variability management becomes a primary concern during software development. On the one hand side methodic support to analyze and specify variability on an abstract level is already available. On the other hand side corresponding support on realization level is still lacking. The goal of this paper is to introduce more efficient approaches to manage variability. Becker [Bec03] developed a general metamodel for variability models on an examination of the most common concepts in variability modelling. Furthermore the authors developed an own language to specify variability, the so called Variability Specification Language (VSL). The advantage of VSL is that it is a XML-based language that can be applied in a broad range of documents and thus allows to handle variability in a uniform manner. The work of Becker is specified on a high-level, while our approach goes with the development of an UML-profile and the mapping to a business process modelling language more in detail.

Ziadi et al. [ZHJ03] want to solve the gap that managing variability in software systems is still a problem. that The authors propose a UML profile for software product lines that should allow the description of variability of product models. In this case the UML models should be considered as reference models from which product models can be derived and created. The authors also propose an extension for class diagrams and an extension for sequence diagrams, but they did not well picked the appropriate stereotypes for the profile. For instance the authors chose classifier, package and feature as a metaclass for the ≪optional≫ stereotype. The problem here is that these metaclasses did not have the semantics to be the metaclass for a dependency relationship.

7 Conclusion

Business Process Modelling Languages (BPMLs) differ in the extent of describing different aspects of a business process case. This leads to a couple of problems. This section presents a short overview of these problems, and summarizes solutions that have been worked out in this thesis.

Section 2 discusses the evaluation of six well-known conceptual Business Process Modelling Languages (BPMLs). It shows which aspects of processes (e.g. activities, roles, interactions, data, etc.) are expressed in each BPML and which ones are not supported.

Although BPMLs have been widely used in research and industry, a comprehensive comparison is missing. Also, a general framework for an evaluation of BPMLs is not available. In order to overcome this gap, a generic metamodel was proposed in section 2. It captures a wide range of business process concepts, because metamodels represent the core concepts of BPMLs and are a good foundation for an evaluation. This metamodel is derived from business process theory [Ham96] [Har91] [JEJ94] [Mar95] and well-established industry and research concepts.

Based on this metamodel, six BPMLs were evaluated, namely UML 2 Activity Diagram (UML 2 AD), Eventdriven Process Chain (EPC), Business Process Modelling Notation (BPMN), Integrated Definition Method 3 (IDEF3), Role Activity Diagram (RAD), and Petri Nets. The metamodel was categorised according to four perspectives: organisational, functional, behavioural, and informational. These four perspectives were extended with the business process context perspective in order to address context information, like process goals or their measures. Basically, the functional and the behavioural perspectives are very well represented in all BPMLs, while the organisational and informational perspectives are only partly supported, and business process context is not explicitly supported.

Section 3 discussed the problem that in general the main focus of conceptual business process modelling languages (BPMLs) is organizing the control flow of tasks of a business process. Usually the explicit modelling of process goals and their measures (see section 2.4) is not addressed in BPMLs. . To solve this gap, the BPMLs UML 2 AD, BPMN, and EPC were extended with business process goals and performance measures. For this challenge different mechanisms are used. At UML 2 Activity Diagrams a UML profile was created, and at the EPC a new view was introduced, as well as at BPMN a new category was established.

We have designed a UML 2 profile for integrating business process goals and performance measures into UML 2 Activity Diagrams. The profile provides an explicit illustration of the performance measures time, cost, and quality. Furthermore, it is possible to show the goals a business process must achieve, as well as the organisational structure that is concerned with alerts that belong to a measure. In order to capture these characteristics, we have extended the UML 2 metamodel for Activity Diagrams, and described them by the stereotypes of our UML 2 profile for performance measures and goals.

EPC as well as BPMN belong to the most famous languages, but both are not able to represent performance measures. In contrast to UML 2 AD, BPMN has no specific metamodel, it defines only a mapping to the Business Process Definition Metamodel (BPDM) [OMG07a]. BPDM provides a general process modeling metamodel that supports the BPMN notation. As explained in section 2 the EPC consists of different views. The EPC provides for each view a metamodel, but not an integrated metamodel that interconnects the related views. A metamodel for EPC and a metamodel for BPMN was defined. Both metamodels are based on the Meta-Object Facility (MOF), the OMG's meta-metamodel [OMG05a]. The metamodels with its extension were presented for integrating business process goals into these languages.

Furthermore the performance measures time, cost, and quality are integrated into defined metamodel, because without measuring the process goals it is not possible to assess if a goal is fulfilled or not. These extensions better illustrate the requirements of a certain business process and enhance the expressiveness of a model. Furthermore the organisational structure, a concept that is already available in EPCs, is integrated in BPMN and UML 2 AD.

Section 4 defined a mapping between the UML profile for performance measures and goals and BPEL for establishing a basis to transform a specific business process modelling language and its conceptually described performance measures to an execution language as well as to monitor the process instances continuously.

This described approach consists of two main steps. First, the UML 2 profile for performance measures and goals is annoted with stereotyped actions, and mapped to BPEL which is shown by a graphically described example. The performance measures cost and quality are not mapped to BPEL, because our approach focusses on the instance level of a business process and not on the type level. Furthermore cost and quality cannot be shown in BPEL, because their are not part of BPEL. Second, the mapping of the UML Profile to BPEL is defined as a UML Profile in eclipse. Thus we provide a foundation for transforming the UML profile to BPEL code with eclipse.

Section 5 focused on the problem that linking business processes with software elements is inadequately supported in business process modelling. On the one hand side it is not possible to show software requirements or components which have an influence on a BPML. On the other hand side it is not possible to show the variabilities of a BPML.

In this section the UML 2 profile for EPCs by [KL06] was connected with UML 2 elements representing software for integrating software requirements and components into a BPML. The goal was to support the connection between business processes and software systems, as well as providing business process models to software developers in a well known notation. Stereotyped actions were linked with use cases as well as with components by the means of dependencies. Use cases address the elicitation of software requirements supporting an action. Components represent the software systems an action requires for its further processing.

Variability models are designed for modelling variabilities of a software. A variability model defines the variability of a software product line and is used during the different life cycle stages of software product lines [PBvdL05]. Variability modelling is a domain specific modelling technique, that is becoming more and more integrated into traditional software engineering. Variability models also have an impact on processes, because variabilities may change the process flow. Unfortunately they are not part of a well-known modelling framework for a higher usability, like the Unified Modelling Language. To address this limitation, section 5 provides a UML 2 profile for variability models, and shows the the dependency between variability models and activity diagrams for visualizing the relationship between variability models and process models.

Bibliography

[ABJP05] Birger Andersson, Ilia Bider, Paul Johannesson, and Erik Perjons. Towards a formal definition of goal-oriented business process patterns. *Business Process Management Journal*, 11(6):650–662, 2005.

[ARGP06] Elvira Rolón Aguilar, Francisco Ruiz, Félix García, and Mario Piattini. Evaluation measures for business process models. In Hisham Haddad, editor, *SAC*, pages 1567–1568. ACM, 2006.

[Bau90] Bernd Baumgarten. *Petri-Netze: Grundlagen und Anwendungen.* BI-Wissen- schaftsverlag, Mannheim, 1990.

[Bec03] Martin Becker. Towards a General Model of Variability in Product Families. In *First workshop on Software Variability Management*, 2003.

[Bez05] Jean Bezivin. On the unification power of models. *Software and System Modeling Journal*, 4(2):171–188, 2005.

[BS04] Behzad Bordbar and Athanasios Staikopoulos. On behavioural model transformation in web services. In *ER (Workshops)*, pages 667–678, 2004.

[BSWS02] Sandy Tyndale Biscoe, Oliver Sims, Bryan Wood, and Chris Sluman. Business modelling for component systems with uml. In *Proceedings of the Sixth International Enterprise Distributed Object Computing Conference (EDOC '02)*, pages 120–131. IEEE Press, 2002.

[Cas05] Fabio Casati. Industry trends in business process management getting ready for prime time. In *16th International Workshop on Database and Expert Systems Applications (DEXA 2005), First International Workshop on Business Process Monitoring and Performance Management (BPMPM 2005)*. IEEE Press, August 2005.

[CKO92] Bill Curtis, Marc Kellner, and Jim Over. Process modeling. *Communication of the ACM*, 35(9):75–90, 1992.

[Cla01] Michael Clau. Modeling variability with uml. *GCSE 2001 Young Researchers Workshop*, 2001.

[DKM+05] Lois Delcambre, Christian Kop, Heinrich C. Mayr, John Mylopoulos, and Oscar Pastor, editors. *Conceptual Modeling - ER 2005, 24th International Conference on Conceptual Modeling, Klagenfurt, Austria, October 24-28, 2005, Proceedings*, volume 3716 of *Lecture Notes in Computer Science*. Springer, 2005.

[DRvdA+07] Michael Dreiling, Michael Rosemann, Will van der Aalst, Lutz Heuser, and Karsten Schulz. Model-based software configuration: Patterns and languages. *European Journal of Information Systems*, 15:1–23, 2007.

[GAGI03] T Gardner, J Amsden, C Griffin, and S Iyengar. *Draft UML 1.4 Profile for Automated Business Processes with a mapping to the BPEL 1.0.* IBM alphaWorks, 2003.

[Ham96] Michael Hammer. *Beyond Reengineering How the process-centered organization is changing our work and our lives.* Harper Collins Publishers, 1996.

[Har91] James H. Harrington. *Business Process Improvement - The breakthrough strategy for total quality, productivity, and competitiveness.* McGraw-Hill, 1991.

[HKRK05] Martin Hitz, Gerti Kappel, Werner Retschitzegger, and Elisabeth Kapsammer. *UML @ Work.* dpunkt.verlag GmbH Heidelberg, 2005.

[HRG83] A Holt, R Ramsey, and J Grimes. Coordinating system technology as the basis for a programming environment. *Electrical Communication*, 57(4):308–314, 1983.

[IBM03] IBM. *Business Process Execution Language for Web Services version 1.1.* http://www-

128.ibm.com/developerworks/library/specification/ws-bpel/, 2003.

[Jea05] Jean-Jacques Dubray. *BPEL 1.1 Metamodel.* http://www.ebpml.org/bpel4ws.htm, 2005.

[JEJ94] Ivar Jacobson, Maria Ericson, and Agneta Jacobson. *The Object Advantage Business Process Reengineering with Object Technology*. ACM Press, Addison-Wesley Publishing, 1994.

[Jen92] Kurt Jensen. *Coloured Petri Nets: A High Level Language for Systems Design and Analysis.* Springer-Verlag, Berlin, 1992.

[JK06] Frédéric Jouault and Ivan Kurtev. On the architectural alignment of atl and qvt. In *SAC '06: Proceedings of the 2006 ACM symposium on Applied computing*, pages 1188–1195, New York, NY, USA, 2006. ACM.

[KK97] Peter Kueng and Peter Kawalek. Goal-based business process models - creation and evaluation. *Business Process Management Journal*, 3(1):17–38, April 1997.

[KL06] Birgit Korherr and Beate List. A uml 2 profile for event driven process chains. In *Proceedings of the 1st IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (CONFENIS 2006)*, pages 161–172. Springer Verlag, IFIP, 2006.

[KWB03] Anneke Kleppe, Jos Warmer, and Wim Bast. *MDA Explained. The Model Driven Architecture: Practice and Promise*. Addison-Wesley, 2003.

[LK05] Beate List and Birgit Korherr. A uml 2 profile for business process modelling. In Jacky Akoka, Stephen W. Liddle, Il-Yeol Song, Michela Bertolotto, Isabelle Comyn-Wattiau, Samira Si-Said Cherfi, Willem-Jan van den Heuvel, Bernhard Thalheim, Manuel Kolp, Paolo Bresciani, Juan Trujillo, Christian Kop, and Heinrich C. Mayr, editors, *ER (Workshops)*, volume 3770 of *Lecture Notes in Computer Science*, pages 85–96. Springer, 2005.

[LYP02] Fu-Ren Lin, Meng-Chyn Yang, and Yu-Hua Pai. A generic structure for business process modeling. *Business Process Management Journal*, 8(1):19–41, 2002.

[Mar95] Martyn Ould. *Business Processes - Modelling and Analysis for Re-engineering and Improvement*. John Wiley and Sons, 1995.

[MMP$^+$95] Richard J. Mayer, Christopher P. Menzel, Michael K. Painter, Paula S. deWitte, Thomas Blinn, and Benjamin Perakath. *Information Integration for concurrent Engineering (IICE) IDEF3 Process Description Capture*. http://www.idef.com, 1995.

[MNN04] Jan Mendling, Gustaf Neumann, and Markus Nüttgens. A comparison of xml interchange formats for business process modelling. In Fernand Feltz, Andreas Oberweis, and Benoît Otjacques, editors, *EMISA*, volume 56 of *LNI*, pages 129–140. GI, 2004.

[NC03] Dina Neiger and Leonid Churilov. Goal-oriented decomposition of event-driven process chains with value focused thinking. In *In Proceedings of the 14th Australasian Conference on Information Systems (ACIS 2003)*, pages 1–12, 2003.

[NC04] Dina Neiger and Leonid Churilov. Goal-oriented business process modeling with epcs and value-focused thinking. In Jörg Desel, Barbara Pernici, and Mathias Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 98–115. Springer, 2004.

[NEK$^+$05] Selmin Nurcan, Anne Etien, Rim Kaabi, Iyad Zoukar, and Colette Rolland. A strategy driven business process modelling approach. *Business Process Management Journal*, 11(6):628–649, 2005.

[Oas07] Oasis. *Business Process Execution Language for Web Services version 2.0*. http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, 2007.

[OMG05a] OMG. *MOF 2.0 Specification*. http://www.omg.org, 2005.

[OMG05b] OMG. *OCL 2.0 Specification*. http://www.uml.org, 2005.

[OMG05c] OMG. *UML 2.0 Superstructure*. http://www.uml.org, 2005.

[OMG06a] OMG. *Business Process Modelling Notation*. http://www.bpmn.org, 2006.

[OMG06b] OMG. *MDA Guide V. 1.0.1*. http://www.bpmn.org, 2006.

[OMG07a] OMG. *Business Process Definition Metamodel, Final Submission*. http://www.modeldriven.org/web/bpdm, 2007.

[OMG07b] OMG. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Final Adopted Specification*. http://www.uml.org, 2007.

[PBvdL05] Klaus Pohl, Gnter Bckle, and Frank van der Linden. *Software Product Line Engineering*. Springer-Verlag Berlin Heidelberg, 2005.

[Pet62] Carl A. Petri. *Kommunikation mit Automaten*. Schriften des IIM Nr. 2, Institut fr Instrumentelle Mathematik, 1962.

[RtHEvdA05] Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In Delcambre et al. [DKM+05], pages 353–368.

[RvdA07] Michael Rosemann and Will van der Aalst. A configurable reference modelling language. *Information Systems*, 32:1–23, 2007.

[SAJ+02] Eva Soederstroem, Birger Andersson, Paul Johannesson, Erik Perjons, and Benkt Wangler. Towards a framework for comparing process modelling languages. In Anne Banks Pidduck, John Mylopoulos, Carson C. Woo, and M. Tamer Özsu, editors, *CAiSE*, volume 2348 of *Lecture Notes in Computer Science*, pages 600–611. Springer, 2002.

[Sch99] August-Wilhelm Scheer. *ARIS Business Process Modeling*. Springer Verlag, 1999.

[SVC+01] Pedro Sinogas, André Vasconcelos, Artur Caetano, João Neves, Ricardo Mendes, and José M. Tribolet. Business processes extensions to uml profile for business modeling. In *ICEIS (2)*, pages 673–678, 2001.

[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, 2003.

[Vit04] Valdis Vitolins. Business process measures. In *Proceedings of the International Conference Baltic DB and IS*. Scientific Papers University of Latvia Vol. 673, 2004.

[W3C01] W3C. *Web Services Description Language 1.1*. http://www.w3.org/TR/wsdl, 2001.

[Wor98] Workflow Management Coalition. *Interface 1: Process Definition Interchange Process Model, WfMC TC-1016-P*. http://www.wfmc.org/standards/docs/if19807r10.pdf, 1998.

[WSS$^+$07] Manuel Wimmer, Andrea Schauerhuber, Michael Strommer, Wieland Schwinger, and Gerti Kappel. A Semi-automatic Approach for Bridging DSLs with UML. In *Workshop Proc. of 7th OOPSLA Workshop on Domain-Specific Modeling (DSM07), Montreal, Canada*, 2007.

[WvdAD$^+$05] Petia Wohed, Wil M. P. van der Aalst, Marlon Dumas, Arthur H. M. ter Hofstede, and Nick Russell. Pattern-based analysis of the control-flow perspective of uml activity diagrams. In Delcambre et al. [DKM$^+$05], pages 63–78.

[ZHJ03] Tewfik Ziadi, Loïc Hélouët, and Jean-Marc Jézéquel. Towards a UML Profile for Software Product Lines. In *Software Product-Family Engineering, 5th International Workshop*, pages 129–139. Springer, 2003.