



SPECIFIC TARGETED RESEARCH PROJECT INFORMATION SOCIETY TECHNOLOGIES

FP6-IST-2005-033606

Visualize all model driven programming VIDE

WP 7	Deliverable number 7.1 Metamodel and notation of the VIDE process modelling language, requirements concerning process model
-------------	--

Project name: Visualize all model driven programming

Start date of the project: 01 July 2006

Duration of the project: 30 months

Project coordinator: Polish - Japanese Institute of Information Technology

Leading partner: Institute for Information System (IWi) at the German Research Center for Artificial Intelligence (DFKI)

Due date of deliverable: 30.09.2007

Actual submission date: 9.11.2007

Status: developed / draft / **final**

Document type: report

Document acronym: DEL

Editor(s): Christian Seel, Andreas Martin

Reviewer(s): Sheridan Jeary, Tomasz Czwarno

Accepting: Kazimierz Subieta

Location: <http://www.vide-ist.eu>

Version: 1.18

Dissemination level: PU/PP/RE/CO

Abstract:

After the requirements of the end users have been defined, the software development concept of VIDE plans the creation of business process models. The business process models are the first models of the development process with a defined language.

Therefore work package 7, whose results are described in this deliverable, has the tasks to evaluate existing business process modelling and execution languages (task 7.1), to define a CIM level language for VIDE (task 7.3) and to analyse which business processes are supported by VIDE and how the software development with VIDE is organized (task 7.2).

For this purpose eight business process languages have been described and evaluated. Based on this evaluation a VIDE CIM level language (VCLL) has been defined. This language provides three modelling views and the use of business rules. It is compliant to standards as much as possible. Its specification is done using UML metamodels, which are refined with OCL-statements. Additionally a graphical notation, which is based on the Business Process Model Notation (BPMN) is provided.

Furthermore the business processes that are supported by VIDE are classified in a morphological box and the procedure model for the software development process with VIDE is defined based on the spiral model.

The VIDE consortium:

Polish-Japanese Institute of Information Technology (PJIT)	Coordinator	Poland
Rodan Systems S.A.	Partner	Poland
Institute for Information Systems at the German Research Center for Artificial Intelligence	Partner	Germany
Fraunhofer	Partner	Germany
Bournemouth University	Partner	United Kingdom
SOFTEAM	Partner	France
TNM Software GmbH	Partner	Germany
SAP AG	Partner	Germany
ALTEC	Partner	Greece

History of changes

<i>Date</i>	<i>Vers ion</i>	<i>Author</i>	<i>Change description</i>
10.12.2006	1.01	Christian Seel	document creation
10.03.2007	1.02	Christian Seel	BU contribution added
30.08.2007	1.03	Christian Seel	Metamodel description added
30.09.2007	1.15	Christian Seel	Comments of PJIIT and BU regarded
08.11.2007	1.18	Christian Seel	Language and layout corrections

Executive summary

The Computation Independent Modelling (CIM)-Level is the first level in which user defined modelling languages form part of the software development process. The purpose of the CIM level is to integrate business people that are usually the supposed users of the software system in the development process and to link the business process based description to the more technical models. The difference between the CIM and Platform Independent Modelling (PIM)-Level is the view on the software system that is developed. While the PIM model describes the parts of the software system like classes, method, attributes, etc., CIM models regard the system from outside. They describe the way the system is used and which business processes or parts of processes it supports.

Work package 7 has as its major task to provide a CIM level modelling language that fits in the MDA approach of VIDE. For this purpose the relationship between the CIM and PIM levels are analysed in this deliverable. VIDE provides two relations between CIM and PIM level. On the one hand the CIM level models can be used as a user's description in order to create an application with the VIDE CIM level language. On the other hand the VIDE CIM level language provides the opportunity to orchestrate an application. For this second orchestration branch the VIDE CIM level language describes the control flow between different applications. This description is exported in form of the vendor independent standard XPDL 2.0 and can be used in a workflow management system which implements the orchestration.

In order to design the VIDE CIM level language eight existing business process modelling and execution languages have been evaluated (task 7.1). For this purpose the requirements of deliverable D.1.1 have been refined and an evaluation schema has been developed. The evaluation results in the recommending the combination of the Business Process Modelling Notation (BPMN) and XPDL 2.0 for the orchestration as the best base for the VIDE CIM level language design.

The design of the language extends the expressiveness of BPMN by data modelling, modelling of organisational units and business rules. As BPMN is just a notation the whole language has been defined as a UML metamodel, which is further refined by OCL statements. Additionally a graphical notation for this language is defined in order to be able to implement a graphical modelling tool.

The use of the VIDE CIM level language is also discussed. Business processes that are supported by VIDE have been classified. For this discussion attributes for the classification of business processes have been collected from the literature and partners. Based on these attributes business processes that are supported by VIDE are classified. The results are depicted in a morphological box.

Furthermore the software development process with VIDE is described. The process is based on the spiral model and is designed for the high involvement of business users (customers) in development. It also allows the division of a software system into different VIDE applications that can then be combined by the orchestration functionality that the VCLL provides.

Table of content

Abstract:	- 2 -
History of changes	- 3 -
Executive summary	- 4 -
Table of content	- 5 -
List of figures	- 6 -
List of tables	- 7 -
1. Introduction	- 8 -
2. Analysis of business process modelling and execution languages	- 9 -
2.1 Positioning of the VIDE CIM level language into the project	- 9 -
2.2 Requirements of the VIDE CIM level language	- 11 -
2.3 Evaluation of business process modelling and execution languages	- 14 -
2.3.1 Evaluation Schema	- 14 -
2.3.2 Language evaluation	- 14 -
2.3.2.1 Event driven Process Chain	- 14 -
2.3.2.2 Business Process Modeling Notation	- 19 -
2.3.2.3 Labelled Transition System Analyser	- 22 -
2.3.2.4 UML Activity Diagram	- 24 -
2.3.2.5 Unified Modelling Language Use case	- 29 -
2.3.2.6 Role Activity Diagrams	- 36 -
2.3.2.7 Business Process Execution Language for Web Services	- 41 -
2.3.2.8 XML Process Definition Language	- 47 -
2.4 Summary of the language evaluation	- 49 -
3. Design of the VIDE CIM level language	- 50 -
3.1 Specification of the VIDE CIM level language	- 50 -
3.1.1 Metamodel of the VIDE CIM level language	- 50 -
3.1.1.1 Process view	- 50 -
3.1.1.2 Data view	- 55 -
3.1.1.3 Organisational view	- 56 -
3.1.1.4 Business Rule view	- 57 -
3.1.2 Additional OCL constraints	- 59 -
3.1.2.1 Pool-lane-lane element relation	- 59 -
3.1.2.2 Sub-process-lane element relation	- 59 -
3.1.2.3 Event types	- 60 -
3.1.2.4 Sequence flow	- 60 -
3.1.2.5 Message flow	- 61 -
3.1.3 Graphical Notation of the VIDE CIM level language	- 62 -
3.2 Modelling example	- 66 -
4. Business processes supported by VIDE	- 70 -
4.1 Classification of business processes	- 70 -
4.2 Criteria of Business processes supported by VIDE	- 72 -
5. Procedure Model for the VIDE software development process	- 74 -
5.1 Evaluation of existing software development procedure models	- 74 -
5.1.1 The waterfall model	- 74 -
5.1.2 The V-model	- 75 -
5.1.3 Evolutionary development	- 77 -
5.1.4 Component-based software engineering	- 79 -
5.1.5 Incremental delivery	- 80 -
5.1.6 The software prototyping model	- 82 -
5.1.7 The object-oriented model	- 83 -
5.1.8 Concurrent Engineering	- 85 -
5.1.9 Spiral development	- 86 -
5.2 Vide Procedure model	- 88 -
References	- 92 -

List of figures

Figure 1: CIM perspective on the VIDE architecture	10 -
Figure 2: Meta-Model of the eEPC	16 -
Figure 3: EPC modelling example	18 -
Figure 4: Event types of BPMN	20 -
Figure 5: BPMN Modelling Example	21 -
Figure 6: LTS modelling example	23 -
Figure 7: Constructs of UML AD	25 -
Figure 8: UML AD Metamodel	26 -
Figure 9: UML AD modelling example	28 -
Figure 10: Relationships in use case diagrams	30 -
Figure 11: Notations for <<include>> and an example usage	30 -
Figure 12: <<extend>> example usage	31 -
Figure 13: Use case modelling concepts - found in [OMG07]	32 -
Figure 14: Notation for roles	37 -
Figure 15: Notation for actions	37 -
Figure 16: Interaction modelling example	37 -
Figure 17: Notation for case refinement	37 -
Figure 18: Notation for part refinement	37 -
Figure 19: Various notations for states	38 -
Figure 20: Notation for final state	38 -
Figure 21: Notation for Iteration	38 -
Figure 22: Notation for triggers	38 -
Figure 23: Notation for new roles	38 -
Figure 24: RAD modelling elements - found in [CB05]	39 -
Figure 25: RAD modelling example	40 -
Figure 26: BPEL Metamodel of the Activity Elements	42 -
Figure 27: BPEL Metamodel	44 -
Figure 28: BPEL modelling example	45 -
Figure 29: WfMC process definition meta-model	48 -
Figure 30: Process view metamodel	50 -
Figure 31: Combinations of Event Categories and Subcategories	55 -
Figure 32: Data view metamodel	56 -
Figure 33: Organisational view metamodel	56 -
Figure 34: Example for decision business rules	58 -
Figure 35: Process view, aggregated for orchestration	67 -
Figure 36: Detailed process view	68 -
Figure 37: Data view	68 -
Figure 38: Morphological box for business process classification	71 -
Figure 39: Classification of business processes supported by VIDE	73 -
Figure 40: Waterfall process models [Somm07]	74 -
Figure 41: The V-model [Balz98]	76 -
Figure 42: Product states in the V-model [Balz98]	77 -
Figure 43: Evolutionary development process [Somm07]	78 -
Figure 44: Component-based software engineering process [Somm07]	79 -
Figure 45: Incremental development process [Somm07]	81 -
Figure 46: Software prototyping process [Somm07]	82 -
Figure 47: The object-oriented development process [Balz98]	84 -
Figure 48: Concurrent engineering process [Balz98]	85 -
Figure 49: Spiral development process [Somm07, Balz98]	87 -
Figure 50: VIDE users in relation to MDA levels (c.f. D.1.1)	89 -
Figure 51: VIDE Procedure Model	90 -

List of tables

Table 1: CIM level language requirements of D.1.1	- 11 -
Table 2: Multiplicities of the Predecessor/Successor-Relationship, similar to [Be02]	- 16 -
Table 3: Evaluation of EPC.....	- 19 -
Table 4: Evaluation of BPMN.....	- 22 -
Table 5: Evaluation of LTSA	- 24 -
Table 6: Evaluation of UML AD	- 29 -
Table 7: Evaluation of UML Use Cases.....	- 34 -
Table 8: Evaluation of RAD	- 41 -
Table 9: Evaluation of BPEL	- 47 -
Table 10: Evaluation of XPD L	- 49 -

1. Introduction

The Deliverable D.7.1 “Metamodel and notation of the VIDE process modelling language, requirements concerning process model” is based on the work of work package 7. Work package 7 consists of three subtasks. The first one, task 7.1, analyses existing business process modelling and execution languages. The second task (7.2) is to classify business processes that are supported by VIDE and to define a procedure model for the software development process with VIDE. The third task 7.3 is to design the VIDE CIM level language (VCLL).

The deliverable is structured into six main sections. The first section is the introduction which gives an overview of the structure of the deliverable and relates each section to the three tasks of work package 7. The second section deals with the analysis of existing business processes and execution languages. It summarises the results of task 7.1. The section starts with a short positioning of the CIM level into the VIDE architecture and depicting the two relations between the CIM and PIM level. It then presents requirements of the VIDE CIM level language, (which derive from consortium partners and targeted VIDE CIM level users) and the architecture and positioning of the CIM level language into the project. Afterwards an evaluation schema for the business process modelling languages is presented and then applied to eight languages. The second chapter closes with a summary of the individual evaluation of each language and the design decisions that were derived from it for the VIDE CIM level language.

The third chapter deals with the specification of the VIDE CIM level language. The specification is MOF compliant and is based on standards as much as possible. The VIDE CIM level language is defined by a metamodel and enriched with OCL statements. Both the specification of the VCLL and chapter 3 is divided into four views: the process view, data view, organisational view and business rule view. The section ends with a modelling example using the VCLL, which was based on a real-life case provided by SAP. This chapter contains the results of task 7.3.

The next two chapters are related to task 7.2. Firstly, in chapter 4 business processes are classified according to classification criteria found in the literature. These criteria are summed up in a morphological box. Afterwards the business processes that are supported by VIDE are classified. Chapter 5 deals with the VIDE software development process. Currently used software development procedure models are evaluated. Based on these a state-of-the-art analysis of the VIDE software procedural model is presented.

The sixth chapter summarises the deliverable and depicts further issues related to work package 7, such as the implementation of a software modelling tool for the VCLL (WP 9) and the transformation from CIM to PIM (WP 5).

2. Analysis of business process modelling and execution languages

2.1 Positioning of the VIDE CIM level language into the project

The Model Driven Architecture Approach as it was developed by the Object Management Group (OMG) consists of four levels. The three upper levels are models based while the forth one is executable program code.

- **Computation Independent Model (CIM):**

The CIM level is independent from technical aspects. It rather describes the behaviour and the way the application systems will be used. Therefore on CIM level not the internal view of an application system is depicted as moreover the view of its stakeholders. The aim of CIM modelling is to bridge the GAP between the business and the IT based view [OMG 03] on software development, as the complexity and the frequency of changes in business is rising and the involvement of business experts becomes more important. CIM models can be both structure and behaviour oriented, as CIM models e.g. describe the business process the application system will be involved in and the data structure it operates on as well.

- **Platform Independent Model (PIM):**

But the CIM doesn't contain all necessary information that is required to execute it. Therefore on the PIM level the design and the inner structure of the application is described. Anyway PIM models do not have platform specific structures. According to the OMG 99% of MDS software development projects use UML on the PIM level [OMG07-MDA].

- **Platform Specific Model (PSM):**

The PSM models are created by transformations of PIM models to a specific target platform. By target platform programming languages or frameworks like J2EE, CORBA, .NET are addressed. On this level platform specific information like specific data types are added.

The VIDE CIM level language, which is developed in WP 7, is located on the Computation Independent level. According to the description of work the CIM level language should be able to cover the functionality of a workflow system (p. 59 DoW). The idea of this requirement is to achieve the ability to orchestrate VIDE application by the usage of a workflow management system. On the other hand there is OMG's approach to MDA, which sees CIM as the starting point of the MDA-driven software development process.

Therefore the CIM level language provides a "double-nature" approach that is shown in Figure 1. The VIDE CIM level language covers on the one hand the functionality of a workflow management system and allows an export in XPDL 2.0. XPDL 2.0 is an open standard of the WfMC [XPDL05], which allows describing workflows in a vendor-independent format. Based on this export the information that is missing in order to execute the workflow description of CIM level is added in the WfMS's design component. In VIDE the tool OOWF from RODAN will be used for this purpose. After the CIM description of the workflow has been enriched on PIM level it can be transferred to the workflow engine and be executed. The workflow engine has the capability to start VIDE application. This process is depicted by the left arrow in Figure 1.

On the other hand the VIDE CIM level language can be used as starting point and for the description of requirements for the application development. In this case the whole expressiveness of the language can be used (see chapter 3.1), as the compliance to XPDL 2.0 doesn't need to be regarded. In this branch a transformation wizard, which is develop in WP 5, will be used in order to keep the relation between CIM and PIM models and vice versa. The parts that are covered in WP 7 are marked red in the figure.

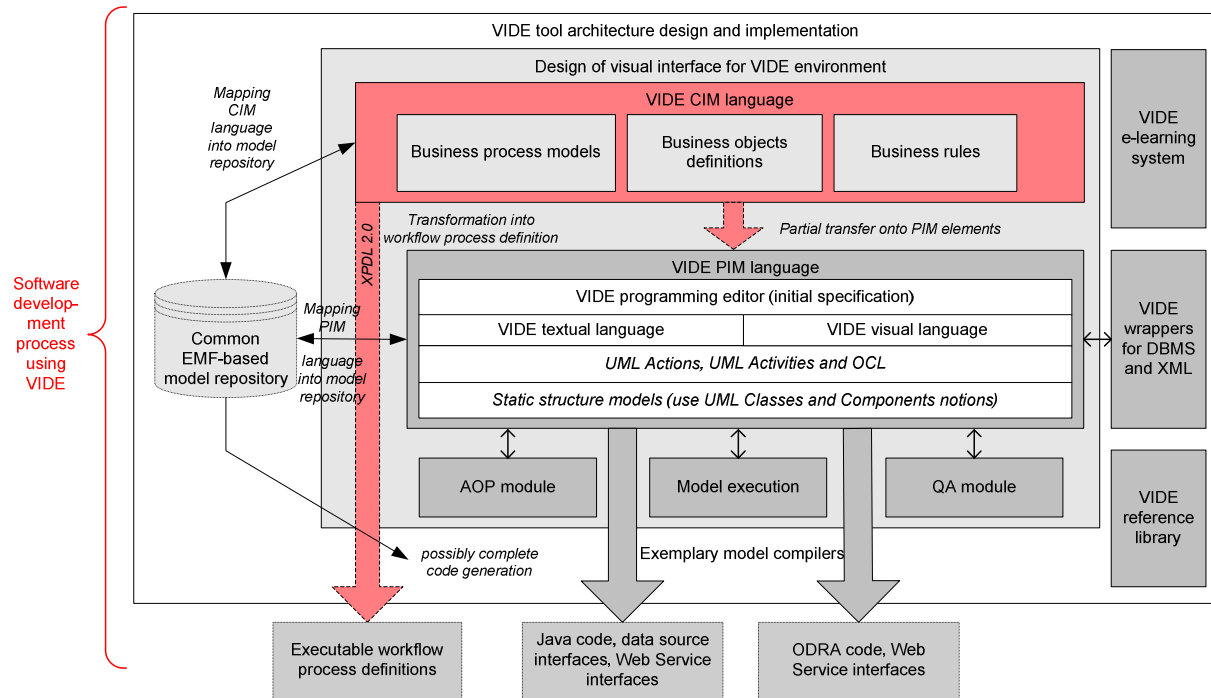


Figure 1: CIM perspective on the VIDE architecture

The development of the VIDE CIM level language is described later in chapter 3.1. The next sections analyse the requirements of the VIDE CIM level language. Afterwards existing business process modelling and execution languages are evaluated based on these requirements.

2.2 Requirements of the VIDE CIM level language

This section describes the requirements of the VIDE CIM level language. Each requirement is formulated into a standardised table shown below:

CIM REQ: ID	REQ stakeholder: CIM User technical VIDE architecture	MAY SHOULD MUST
<i>Requirement description</i>		
Related requirement IDs:		

The left field in the first row assigns a unique number to each requirement in order to identify it unambiguously. The middle field identifies the origin of the requirement. Three values are possible for this field: user, technical and VIDE architecture.

User requirements are requirements that derive from the needs of business people and their way of describing things and dealing with their domain. Technical requirements derive from the use of tools, frameworks and standards in the project, such as MOF, EMF, GMF and GEF. The last variety of requirements derives from the architectural approach presented in the section above.

The right field in the first row describes the priority of the requirement. In the centre field of each requirement is a full description. Additionally, the origin of a requirement is noted here, if the requirements are interdependent or derive from requirements of deliverable D.1.1. Therefore the requirements of deliverable D.1.1 are cited here. Most of the requirements of D.1.1 are not relevant for the CIM level language because they refer to a different MDA-level or to the functionality of tools which do not influence the VIDE CIM level language directly. The relevant requirements of D.1.1 for the evaluation of the CIM level languages are:

Requirement Number	Name	Priority
REQ – NonFunc 1	Accessibility at the CIM level	SHOULD
REQ – NonFunc 2	CIM level collaboration	MAY
REQ – NonFunc/Semantics 4	Clear and unambiguous notation	SHOULD
REQ – NonFunc 6	Appropriate textual/graphical fidelity	SHOULD
REQ – NonFunc 9	Scalability of proposed solution	MUST
REQ – User 1	Flexibility and interoperability of VIDE language and tools	SHOULD
REQ – Lang 4	Compliance with Standards	SHOULD
REQ – Lang 7	Language for CIM, PIM, PSM modelling	SHOULD
REQ – Tool 9	CIM modelling standards	MAY

Table 1: CIM level language requirements of D.1.1

These relevant requirements are extended and further refined into the following 13 requirements. The requirements have been gathered within the VIDE consortium among scientific and industry partners, with respect to the description of work and the expected result of the project.

CIM REQ: 1	REQ stakeholder: CIM User	MAY SHOULD MUST
<p>The language must be able to describe business processes from a business user's point of view. In order to check this requirement it is refined into sub-criteria: The language must:</p> <ul style="list-style-type: none"> • Allow the description of business processes, which means it should at least: <ul style="list-style-type: none"> • be able to describe activities, • the control flow between them, • branches and joins in the control flow • and data objects that are used within the process, as VIDE is especially aiming at data-intense business applications. • allow the modelling of business processes seamlessly, including manual activities • use familiar and standardised notation, if possible • allow the modelling of business process without technical details (as method calls or detailed data types), as the user group of the CIM level language is not familiar with these 		

concepts.
Related requirement IDs: REQ – NonFunc 1

CIM REQ:2	REQ stakeholder: CIM User	MAY SHOULD MUST
The language should allow users to describe their business goals or business rules.		
Related requirement IDs: REQ – NonFunc 1		

CIM REQ: 3	REQ stakeholder: CIM User	MAY SHOULD MUST
The language may use terms or primitives which are familiar to business users, such as “activity” instead of terms like “method”.		
Related requirement IDs: REQ – NonFunc 1		

CIM REQ: 4	REQ stakeholder: CIM User	MAY SHOULD MUST
The language should be intuitive, which means that the notation can be learned within a relatively short time-scale. If the notation is intuitive this would reduce the learning time. Therefore known standards and notations can reduce the learning time.		
Related requirement IDs: CIM REQ 5,		

CIM REQ: 5	REQ stakeholder: CIM User	MAY SHOULD MUST
The CIM level language may be compliant to well-known standards, if at all possible.		
Related requirement IDs: REQ – Lang 4, CIM REQ 4		

CIM REQ: 6	REQ stakeholder: CIM User	MAY SHOULD MUST
Graphical notations are the state-of-the-art and widely spread in CIM modelling as languages like the UML and frameworks like ARIS shows. Therefore the VIDE CIM level language must have a graphical notation.		
Related requirement IDs: REQ – NonFunc 6 , CIM REQ 7,		

CIM REQ: 7	REQ stakeholder: technical	MAY SHOULD MUST
The successful modelling languages such as UML are supported by graphical modelling tools. Without tool support the larger models cannot be handled, modelled in groups nor be reused. Therefore the support of the VIDE CIM level language by a graphical modelling tool is required.		
Related requirement IDs: REQ – NonFunc 2,		

CIM REQ: 8	REQ stakeholder: technical	MAY SHOULD MUST
The integratability into other BPM tools is desirable in order to create a vendor independent solution. The integratability is facilitated by the use of standards.		
Related requirement IDs: REQ – NonFunc 2, REQ – User 1, CIM REQ 5		

CIM REQ: 9	REQ stakeholder: technical	MAY SHOULD MUST
The description of the syntax and semantics should be done formally in order to avoid ambiguities of the VIDE CIM level language.		
Related requirement IDs: REQ – NonFunc/Semantics 4		

CIM REQ: 10	REQ stakeholder: technical	MAY SHOULD MUST
The modelling language should be able to handle complex models for larger business processes and scenarios.		
Related requirement IDs: REQ – NonFunc 9		

CIM REQ: 11	REQ stakeholder: VIDE architecture	MAY SHOULD MUST
As stated in the description of work the VIDE CIM level language must be able to support the description of workflows.		
Related requirement IDs:		

CIM REQ: 12	REQ stakeholder: VIDE architecture	MAY SHOULD MUST
In order to fit in the MDA-approach for VIDE the CIM language must contain information that is transformable into the underlying PIM level.		
Related requirement IDs:		

CIM REQ: 13	REQ stakeholder: user	MAY SHOULD MUST
In order to manage different types of information the CIM language should be structured, which enhances the readability and speeds up the understanding of models by users. Therefore different views like in the ARIS framework can be introduced.		
Related requirement IDs:		

The D.1.1 requirements “REQ – Lang 7” and “REQ – Tool 9” are related to the use of the languages BPML and BPMN. They can not directly been seen as language independent like the other requirements. Therefore both languages are in the set of evaluated languages.

2.3 Evaluation of business process modelling and execution languages

In this section existing business process modelling and execution languages are evaluated according to the requirements that are depicted in the section above. In order to be able to compare different languages the major aspects of each language are described. An evaluation schema is presented in order to facilitate the comparability and the overview of the different languages.

2.3.1 Evaluation Schema

The evaluation schema consists of 12 sections. Not all sections have to be used for all languages, e. g. if there is no defined metamodel the corresponding section don't describe one.

Language Name: This keyword contains the official name of the language.

Acronym: This keyword contains the official acronym of the language.

Specification document(s): This section contains the specification documents of the language.

Specifying organisation: This section contains name of the standardization organisation that specifies the language, e. g. the OMG.

Evaluated version(s): In order to be aware of version specific issues and changes the version of the evaluated language is stated here.

Focus of the language: This section describes for which purpose the language is designed and typically used, e.g. a language can be designed as exchange format for workflow descriptions between several workflow management systems.

Description of the language and its model elements: This section describes the modelling language, including its elements, the way they are used and what their semantics are.

Metamodel: As one of the purposes of WP 7 is to design a language and define a metamodel for it, the metamodels of existing language specifications are described here. This section is not available for every language because some languages do not have metamodels defined.

Modelling Example: This section provides modelling examples in the evaluated language, which contain the most important structures and elements of the language. The purpose of this section is to recognize features and the use of the languages other than on the meta level.

Part of Framework: Some languages are part of a framework and have a special role in this framework, e. g. the Event-Driven-Process-Chain (EPC) is part of the ARIS framework and serves as an integration modelling language for the different views of the framework. Therefore this section describes the role of the evaluated modelling language in the framework, if the language is part of a framework.

Supporting software tools: In order to get an overview of available tools and the features they provide a selection of the major software tools supporting the language are stated here. This list can be helpful for the implementation of the prototype (WP 9), as it facilitates to choose the set features that the prototype will support.

Evaluation: In this section the evaluation of the modelling language is done. The evaluation is based on the requirements stated in section 2.2 and indicates which parts and concepts of the evaluated language will be considered in the development of the VIDE CIM level language.

2.3.2 Language evaluation

2.3.2.1 Event driven Process Chain

Acronym:
EPC/eEPC

Specification document(s):

Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage "ereignis-gesteuerter Prozessketten (EPK)". In: Veröffentlichungen des Instituts für Wirtschaftsinformatik, No. 89, Universität des Saarlandes, Saarbrücken, 1992.

Specifying organisation:

The EPC has been developed in a research project done by the Institute for Information Systems and SAP. There is no organisation specifying the language.

Evaluated Version(s):

There is no versioning available for the EPC.

Focus of the language:

The EPC has been developed for the description of business processes on the conceptual level. Therefore it is not focused only on application development. Furthermore it is a multi-purpose language, which is also used for visualisation, analysis and optimization of business processes from an organisational perspective. The target group of this language are mainly business users, but it is also used for the requirements analysis phase in software development projects.

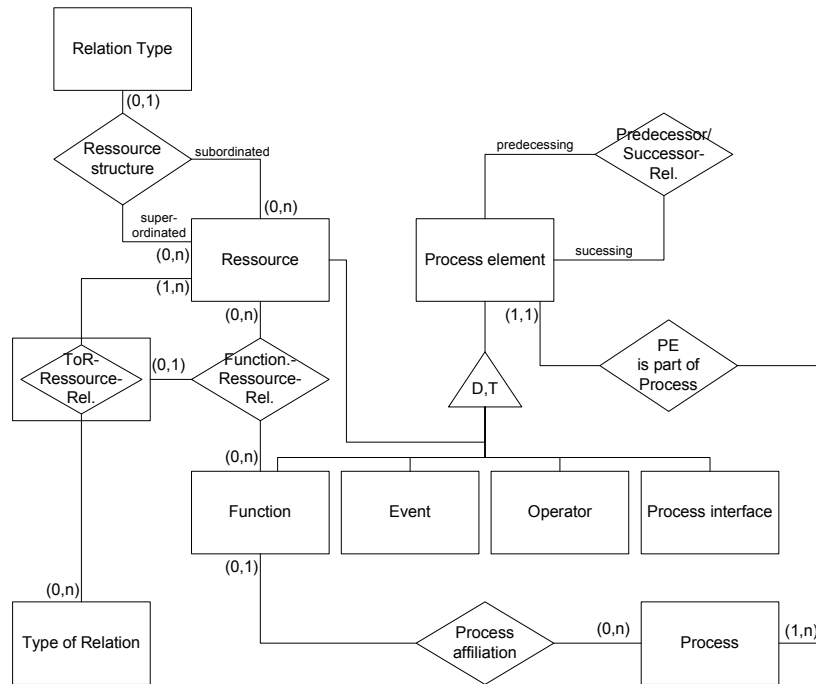
Description of the language and its model elements:

The EPC describes business processes by the use of alternating functions and events, similar to Petri nets with transitions and places. Functions describe business activities and events passive states. The events and functions are linked to each other by edges, which describe the control flow of the business process. The control flow functions and events can only be connected to each other. Every function and event may not have more than one ingoing and one outgoing edge. In order to split and join the control flow operators with the occurrences OR, XOR and AND can be used after functions and events. The OR connector has the meaning of "inclusive or", which means that one or more or even all of the successor elements can be triggered. The XOR connector means "exclusive or", which allows exactly one successor element to be triggered. The AND split triggers all successor elements and parallelizes the control flow. All three types of connectors can be used as split and as join connectors as well. They can be used at any place in EPC, except the OR- and XOR-Operators must not be used after events. The last remaining element, that could be connected via the control flow are process interfaces, which can be applied at the end and the beginning of an EPC to connect two EPCs from different models.

The process interface has the semantic of a reference which points from the end of one process to the beginning of another one. Therefore large process models can be split into different parts which have a different focus. Additionally a function in an EPC can be refined by a more detailed EPC. Therefore a hierarchy of business processes from high level overview processes to very detailed business processes can be created. Furthermore resources which are the input or output of functions can be modelled. Therefore a resource, like an organisational unit, is annotated to the function in which it is involved.

Metamodel:

In the EPC meta-model, which is constructed as an Entity-Relationship-Model (ERM) [Ch76], the four model elements "function", "event", "operator" and "process interface" are generalised as "process element". The connection between these four model elements are represented by the "Predecessor-Successor-Relationship". A problem of the generalisation to the "process element" and its recursive relationship are multiplicities of this relationship which should define which connections of these model elements are permitted. This problem can be solved by determining the multiplicities for each combination of predecessor and successor, e.g. if a function is followed by an event the multiplicities of the predecessor are (0,1) and the multiplicities of the successor are (0,1), too. The multiplicities of all possible cases are shown in Table 1.

**Figure 2: Meta-Model of the eEPC**

The next important meta-model element is the „process“. Every process element is part of exactly one process and each process consists of one or more process elements. The construct “process” can be used to create hierarchies of process models. Therefore a function is detailed by a sub-process. This refining of functions with sub-processes can be done for an unlimited number of levels. This issue is represented in the meta-model by the relation “process affiliation” between a function and a process. This allows that one process can not be refined to many functions. A function can either not be refined or be refined by exactly one process, because if a function was refined by more than one process the execution of the function would be ambiguous. The unlimited hierarchy levels of function affiliations are possible, because of the generalisation of the function to process element which is part of a process. So every sub-process consists of process elements. These elements could also be functions which are refined by another sub process.

Table 2: Multiplicities of the Predecessor/Successor-Relationship, similar to [Be02]

Predecessor	Successor	Multiplicity Predecessor	Multiplicity Successor
Event	Function	(0,1)	(0,1)
Event	Event	(0,0)	(0,0)
Event	AND-Operator	(0,1)	(0,n)
Event	(X)OR-Operator	(0,1)	(0,n)
Function/P.I.	Event	(0,1)	(0,1)
Function/P.I.	Function	(0,0)	(0,0)
Function/P.I.	AND-Operator	(0,1)	(0,n)
Function/P.I.	(X)OR-Operator	(0,1)	(0,n)
AND-Operator	Event	(0,n)	(0,1)
AND-Operator	Function	(0,n)	(0,1)
AND-Operator	AND-Operator	(0,n)	(0,n)
AND-Operator	(X)OR-Operator	(0,n)	(0,n)
(X)OR-Operator	Event	(0,n)	(0,1)
(X)OR-Operator	Function	(0,n)	(0,1)
(X)OR-Operator	AND-Operator	(0,n)	(0,n)
(X)OR-Operator	(X)OR-Operator	(0,n)	(0,n)
Starting Event	Process Interface	(0,1)	(1,1)

Process Interface	End Event	(1,1)	(0,1)
Event	Process Module	(0,1)	(0,1)
Function	Process Module	(0,0)	(0,0)
AND-Operator	Process Module	(0,0)	(0,0)
(X)OR-Operator	Process Module	(0,0)	(0,0)
Process Module	Event	(0,1)	(0,1)
Process Module	Function	(0,0)	(0,0)
Process Module	AND-Operator	(0,0)	(0,0)
Process Module	(X)OR-Operator	(0,0)	(0,0)

The last important characteristic of the modelling language EPC are resources, which can be annotated to functions. Resources¹ like organisational units, application systems or documents have its resource specific type relation to a function. For instance, an organisational unit can have the type of relation “is responsible for”, which is not allowed for a document. So the meta-model contains a relationship between the type of relation and the resource, which determines which type of relation to a function is possible for what resource. As one resource can have more than one type of relation and one type of relation can be suitable for more than one resource, the relationship between a resource and a function is specified by a combination of the resource and its type of relation. Therefore, in the ER-Meta-Model the relationship “ToR-Resource-Rel.” is redefined to an entity type and creates a triple relationship with the entity type “function” and the entity type “resource”. So the exact type of the relation for each connection of a resource to a function is specified. Additional resources can be related to each other, e.g. one organisational unit can have the relationship of the type “reports to” to another organisational unit. This is taken into account by the relationship “Resource Structure”, which also includes the “type of relation”.

Modelling Example:

The example process which is expressed in the EPC example model is a warehouse process. The following paragraph summarises this process: “After the delivery has been arrived, the invoice is verified by the accounting control using the accounting control system. If the invoice entries are not correct, the delivery will be sent back to sender or supplier. The result of this invoice check is stored in the accounting control system by the accounting control. Afterwards for each good the master data, which are stored in the warehouse management system, are checked on correctness by the warehouse management. If necessary, incorrect data into the warehouse management system are modified or new data are created by the warehouse management in this step. Then the goods are moved to the storages and the warehouse management adds the goods to the list of store goods, which is saved in the warehouse management system”. Figure 3 represents this process as EPC:

¹ There is no common understanding, which resources can be annotated to function in the EPC. An impression of the variety of resources gives the ARIS-Toolset of the IDS Scheer AG, which provides in its current release (7.01) 115 resource occurrences.

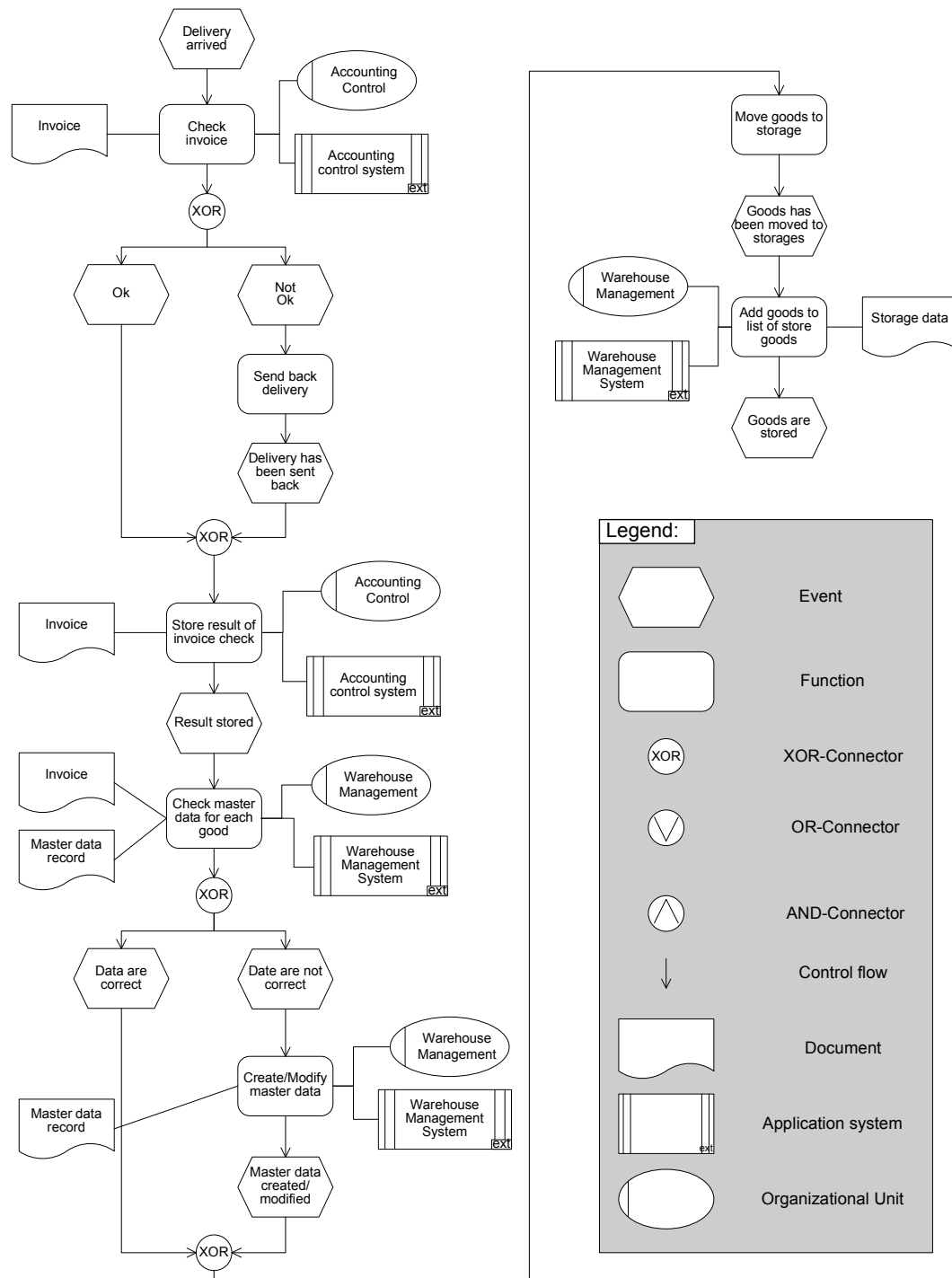


Figure 3: EPC modelling example

Part of Framework:

The EPC is the core modelling language of the ARIS framework (c.f. deliverable D.1.1) at the conceptual level. One of the major concepts of ARIS is the decomposition of enterprise models into five different views. The different views are reunited in the control view. The EPC realises that integration of different views with the annotated resources. These resources are part of other views, e.g. the organisational unit in the EPC is part of the organisational view.

Supporting software tools:

ARIS Business Architect (IDS Scheer AG), VISIO (Microsoft), EPC-Tools (Open Source) and other.

Evaluation:

The EPC is designed in order to model business processes. It is able to describe activities and a complex control flow between them. Therefore it satisfies CIM REQ 1. But the EPC is not able to explicitly represent Business Rules. Business Goals can be included into eEPCs models. The requirements 3 and 4 are fulfilled. EPCs are frequently used for modelling of domain specific models, such as the reference model for industrial business processes by Scheer [Sche94]. Requirement 5 is mostly fulfilled as EPCs can be stated as quasi standard as they are supported by a large variety of tools and spread over many industries. But they are not standardised by a standardization organisation. As the modelling example above shows EPCs have a graphical notation and have a broad tool support. Therefore requirement 6 and 7 are totally fulfilled. Requirement 8 is fulfilled as well, as there are several exchange formats for EPCs like the ARIS Mark-up Language AML [MeNü04] and the EPC Mark-up Language EPML [MeNü06]. The metamodel above shows that EPCs have a partly defined syntax. But as authors like Kindler claim the semantics of EPCs, especially of the OR-connector, is ambiguous [Kind06]. Therefore requirement 9 is only partly fulfilled. Examples like [Sche94] show that EPCs can be used for large and complex models. But the need to use an event after every function decreases the ability to represent complex processes a little, because some events are trivial and do not represent new information, e.g. the event “invoice is checked” which follows the function “check invoice”. Concerning requirement 11 EPCs are not designed to represent executable workflows, therefore this requirement is not fulfilled. But EPC represent most of the information that are needed on CIM level in order to be used in an MDA-approach, which means requirement 12 is mostly fulfilled. A big advantage of EPCs is their use in the ARIS framework. The EPC is the only evaluated language which is able to integrate model elements from different views, such as data objects or organisational units. The evaluation of EPCs is summed up in the following table:

Table 3: Evaluation of EPC²

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	✓	8	✓
2	✗	9	(✓)
3	✓	10	✓
4	✓	11	✗
5	(✓)	12	(✓)
6	✓	13	✓
7	✓		

2.3.2.2 Business Process Modeling Notation**Acronym:**

BPMN

Specification document(s):OMG Final Adopted Specification, <http://www.omg.org/docs/dtc/06-02-01.pdf>**Specifying organisation:**

The standardisation was initially done by Business Process Management Initiative (BPMI). Since June 2005 the Object Management Group (OMG) is responsible for BPMN.

Evaluated version(s):BPMN 1.0, OMG Final Adopted Specification, 6th February 2006**Focus of the language:**

The BPMN has been designed for business process modelling. In contrast to the EPC BPMN is not regarding issues, that are only enriching the business process description, such as organisational structures and resources, data and information models, strategy or business rules [OMG06]. As the language is designed as a notation it focuses especially the visualisation of business processes. A model which is designed in BPMN is called a Business Process Diagram (BPD).

Description of the language and its model elements:

² The symbol ✓ means the requirement is fulfilled, (✓) means the requirement is mostly fulfilled with very little restrictions, (✗) the requirement is not fulfilled however there is a very small functionality available and ✗ means the requirement is not fulfilled.

A BPD can be understood as a graph which uses activities and annotated objects as nodes and the flow between them as edges. The language elements of BPMN can be distinguished into four groups: flow objects, connecting objects, swim lanes and artefacts [Whit04].

The flow objects consist of events, activities and gateways. Activities represent business related actions that are carried out in the regarded business process. They can be subdivided into atomic and compound actions. Atomic activities consist of a single action, whilst compound activities represent a sub-process. A sub-process can either be collapsed or expanded, which indicates if the elements of sub-process are hidden. BPDs are started and terminated with events. An event can either be a start, an intermediate or an end event. These events can have triggers and may cause a result. Triggers can be messages, a timer, an error, the cancellation of a process, compensation, a rule, a link or multiple of the triggers mentioned before. Figure 4 gives an overview on the types of events.

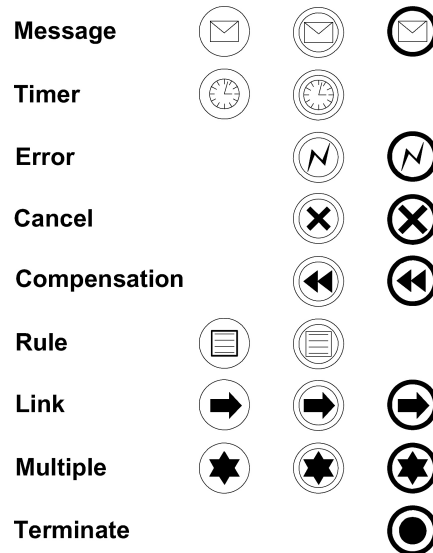


Figure 4: Event types of BPMN

Gateways are used to fork and join the flow in a BPD. They can be based on data or events. The semantics of a gateway both for fork and join can be exclusive decision between alternative paths (XOR), an inclusive decision (OR), a complex condition (e.g. 3 out of 5) or an AND semantic.

These flow objects are linked by connecting objects. They consist of the *sequence flow*, *message flow* and *associations*. The sequence flow determines the execution sequence of the activities from a start event through multiple activities or gateways to an end event. A message flow is used to exchange information between different participants of a process. The concepts for the representation of participants are introduced later. Associations are used to associate artefacts to flow objects, e.g. an association is used to annotate a compensation activity which is out of the normal control flow to the activities it compensates.

Swim lanes are a mean to classify activities into different categories. They can be used to assign activities to organisational units. Therefore two concepts exist: *pools* and *lanes*. A pool can represent a participant, e.g. a company, in a process. It is a container that groups the participant's activities. These pools can be subdivided by lanes. A lane for instance can represent departments in a company which is represented as a pool. Message flows can only exist between different pools.

These basic elements are extended by artefacts. The most common used artefacts are *data objects*. Data objects are linked to activities which can use them as input, create them as output or both. In order to emphasize parts of the business process that are related, *groups* can be used. Groups do not have any influence on the process sequence, they just support analysis purposes. A similar mean are *annotations*. They can be linked to all other language elements and provide comments, which serve for analysis purposes.

An additional model element are *transactions*. A compound activity, representing a sub-process, can be marked as transaction, which means it is either carried out completely or not at all.

Metamodel:

BPMN is specified in textual way. Therefore there is no metamodel available.

Modelling Example:

The example is based on the warehouse scenario described in D.1.1. Two parties, which are represented by a pool, are involved in the process. The supplier delivers goods to the manufacturer. The manufacturer is first checking the invoice. If it contains any errors the delivered goods are returned to the supplier. In any case the

result of the invoice check is stored. This is done by the department “accounting control” as the swim lane indicates. The second department involved is the “warehouse management”. This unit checks the master data of each good for correctness and corrects them if necessary. Then the goods are moved to their storage places and added to the inventory lists.

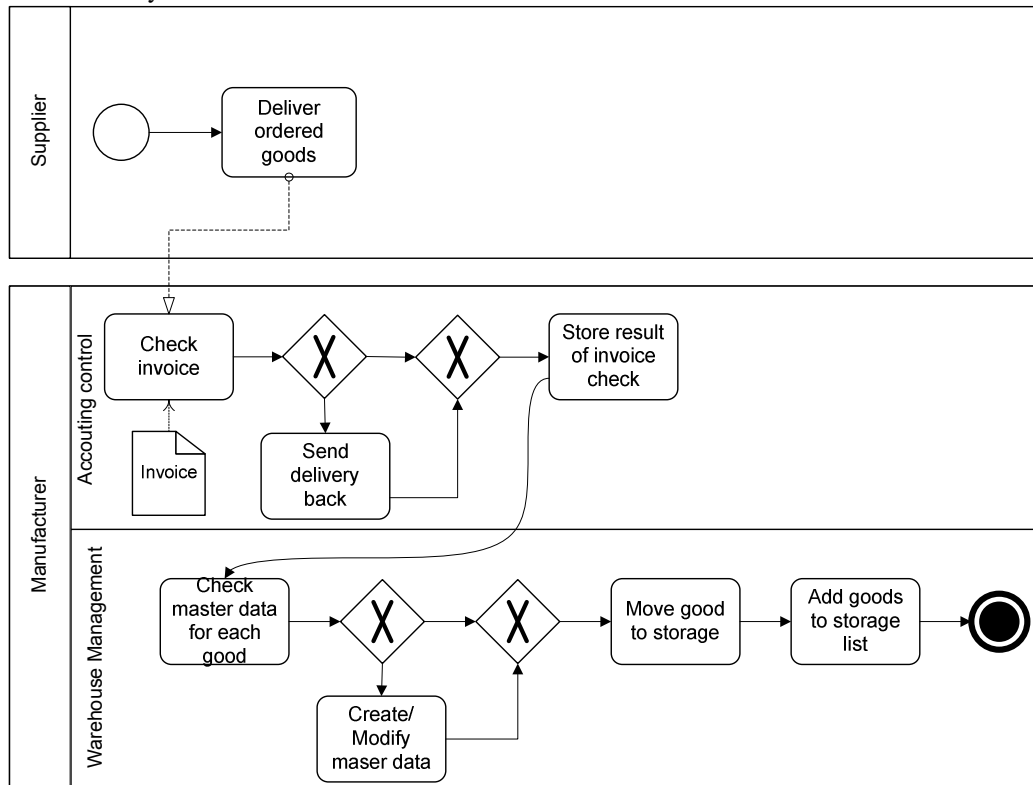


Figure 5: BPMN Modelling Example

Part of Framework:

BPMN is not part of a larger framework.

Supporting software tools:

Together (Borland), ARIS Business Architect (IDS Scheer), WBI Modeler (IBM). Further 42 BPMN Implementors and Quotes are listed under http://www.bpmn.org/BPMN_Supporters.htm.

Evaluation:

BPMN is able to serve as a graphical representation of other language, that don't have an own notation such as BPEL and XPD. BPMN is able to describe activities and a complex control flow between them. Therefore it satisfies CIM REQ 1. But BPMN is not able to represent explicitly Business Rules or Business Goals. The requirements 3 and 4 are fulfilled because BPMN allows users to integrate labels they are familiar with into the models and it doesn't have too many or complex modelling elements, which are required in order to create simple models. BPMN is standardised by the OMG, therefore requirement 5 is fulfilled. BPMN is a graphical notation and it is supported by various modelling tools. As BPMN is only a graphical notation its integratability into other tools (requirement 8) depends on its textual representation. Therefore this requirement is only partly fulfilled. The definition of BPMN is done in natural language and with modelling examples, but there is no sound metamodel. BPMN can be used for large models, as several examples like [KoLi07], [BFS00] or [OMG06] show. BPMN 1.1 is able to represent XPD. 2.0, which is explicitly a language for workflow description. Therefore BPMN fulfils requirement 11. BPMN can be used in a MDA-approach but information like data structures are not regarded in BPMN. BPMN is not designed to handle different view it is only focussing on processes. The evaluation of BPMN is summed up in the following table:

Table 4: Evaluation of BPMN

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	✓	8	(✓)
2	✗	9	(✗)
3	✓	10	✓
4	✓	11	✓
5	✓	12	(✓)
6	✓	13	✗
7	✓		

2.3.2.3 Labelled Transition System Analyser

Acronym:

LTSA

Specification document(s):

Information about using LTSA can be found in the book of Jeff Magee & Jeff Kramer: “Concurrency: State Models and Java Programs”.

Specifying organisation:

The LTSA has been developed by Jeff Magee, Jeff Kramer, Robert Chatley and Sebastian Uchitel at The Department of Computing at Imperial College London. There is no organisation specifying the language.

Evaluated Version(s):

The latest stable version is from November 2, 2005. LTSA-WS Eclipse Plug-in is available, and its current version is 09/10/2006 v0.5.1.

Focus of the language:

LTSA is a verification tool for design-level software analysis of concurrent systems. It mechanically checks that the specification of a concurrent system satisfies the properties required of its behaviour. In addition, LTSA supports specification animation to facilitate interactive exploration of system behaviour.

Description of the language its model elements:

A system in LTSA is modelled as a set of interacting finite state machines. The properties required by the system are also modelled as state machines. LTSA supports Compositional Reachability Analysis (CRA) of a software system based on its hierarchical structure. CRA incrementally computes and abstracts the behaviour of composite components using the architecture of the system as a guide to perform the composition [GKC99]. More formally, each component of a specification is described as a Labelled Transition System (LTS), which contains all the states a component may reach and all the transitions it may perform. However, explicit description of an LTS in terms of its states, set of action labels and transition relation is cumbersome for other than small systems. Consequently, LTSA supports a process algebra notation as the input language – the Finite State Processes (FSP) which uses LTS semantics – for concise description of component behaviour. The tool allows the LTS corresponding to a FSP specification to be viewed graphically aiding both design and verification of system models.

Metamodel:

Systems are structured as sets of simple activities that are represented as sequential processes. Process models are described graphically using state machines (LTS) and textually as finite state processes (FSP), they are then displayed and analysed by the LTSA analysis tool. These finite state machines transit from state to state by executing a sequence of atomic actions. Using LTS this is modelled in the following way:

- States are represented by circles usually with a number or letter inside (in the LTSA notation convention a red circle with the number 0 is the start state, the consecutive states are blue circles).
- Actions are represented by labels above arrows linking states where a transition takes place (the last action, if there exists one, is highlighted in red).

There can be many actions starting from one state because actions can be concurrent. Recursive actions that point from a state to itself, as well as through a number of intermediate states, are possible. The conditions for

choosing the next transition to a state (if there are multiple to choose from) can be specified in FSP as a guarded action.

The intention of LTSA is to implement the modelled processes as threads in Java.

Modelling Example:

The following example is a model described using a state machine of a simple ATM for cash withdrawals. The graphical form of the state machine follows the Labelled Transition Systems notation. The analyzed process is described in more detail here:

“Using an ATM, bank customers can access their bank accounts in order to make cash withdrawals. In this simplified example only that transaction is available. The ATM is switched on and awaiting customer interaction. When a customer wants to use the machine, a credit card is required. It must be inserted into the machine and the correct PIN must be entered. If the entered PIN is incorrect the transaction cannot continue and the PIN must be entered again. After an unspecified number of failed PIN verifications access is denied and the card is retained by the machine (the number of allowed PIN errors can be specified in the textual FSP form). The ATM then goes back to its default state. A successful PIN verification permits cash withdrawal. The transaction ends and the card is returned to the customer. The ATM is ready to perform another transaction.”

The example process which is expressed in the EPC example model is a warehouse process. The following figure summarises this process:

ATM example

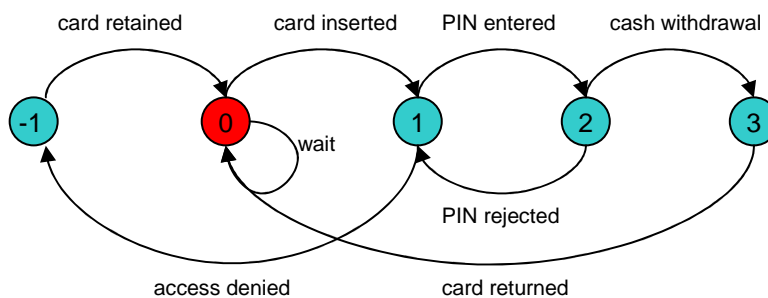


Figure 6: LTS modelling example

Part of Framework:

The LTSA is not part of any larger framework. The tool itself is based on Java SceneBeans Framework which is an animation framework for Java [SBF].

Supporting software tools:

LTSA-Message Sequence Charts (MSC) Analyser [MSC], LTSA-Web Service (WS) [LWS].

Evaluation:

LTSA describes states and transitions between them. They can mostly model business processes and the control flow between them as transitions. LTSAs do not support Business Rules or Business goals as model elements. The use of the language is not intuitive for business people and they are more difficult to learn than less formal languages. LTSAs are defined but not a standard. They have a graphical notation as shown in the example above, but only some modelling tools support them. There is no standardised exchange format for LTSA between different tools. The specification of LTSA is not done formally by metamodel. With regard to complex models LTSA can be rather sophisticated especially if a large number of branches have to be considered. LTSA is not designed to describe workflows. It can be used on CIM level but is not really intuitive for business users and does not contain all relevant information for the creation of PIMs because all aspects about data or participants are not depicted by LTSAs. They don't support the integration of different modelling views. The evaluation of LTSAs is summed up in the following table:

Table 5: Evaluation of LTSA

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	(✓)	8	✗
2	✗	9	(✗)
3	(✗)	10	(✗)
4	(✗)	11	✗
5	(✗)	12	(✓)
6	✓	13	✗
7	(✓)		

2.3.2.4 UML Activity Diagram**Acronym:**

UML AD

Specification document(s):

OMG (Ed.): Unified Modeling Language: Superstructure, Version 2.0, August 2005

Specifying organisation:

Object Management Group (OMG)

Evaluated version(s):

UML 2.0

Focus of the language:

Activity diagrams enable the description of various kinds of processes. They are used for different granularity levels in modelling, e. g. documentation of business, algorithmic or technical based processes. They are often applied on the conceptual level of software development. Their focus is the sequence and conditions for coordinating lower-level behaviours, rather than which classifiers own those behaviours.

Description of the language and its model elements:

The Activity Diagram (AD) is a behavioural diagram, whose semantics are based on Petri net principles. It represents actions, which are linked by control and object flows in a net structure. Every diagram starts with one or more initial or control nodes. There are three kinds of nodes: action nodes, object nodes and control nodes as well as two kinds of flows: control flows and object flows. A token contains an object, datum, or locus of control, and is presented in the activity diagram at a particular node. Each token is distinct from any other, even if it contains the same value.

The nodes are connected to each other by edges, which represents the control flow of the process. Action nodes describe an action, which is the elementary component of the AD. An action may have sets of incoming and outgoing activity edges that specify control flow and data flow from and to other nodes. Object nodes provide and accept objects and data as they flow in and out of invoked behaviours, and may act as buffers, collecting tokens as they wait to move downstream.

Object flows are used for sequencing data produced by one node that is used by other nodes and control flows are applied for sequencing the execution of nodes.

The different control nodes describe the logical structure of the business process:

1. The initial node may be used as starting point of the activity diagram
2. A decision node is a control node that chooses between outgoing flows. It must have a single activity edge entering it, and one or more edges leaving it. The edges coming in and out of a decision node must be either all object flows or all control flows.
3. The merge nodes bring together multiple alternate flows and are not used to synchronize concurrent flows but to accept one among several alternate flows. It must have two or more edges entering it and a single activity edge leaving it.
The functionality of merge nodes and decision nodes can be combined by using the same diamond-shaped node symbol. The edges coming into and out of a fork node must be either all object flows or all control flows.
4. Join nodes are introduced to support parallelism in activity. It must have one or more activity edges entering it, and only one edge leaving it.

5. Fork nodes split a flow into multiple concurrent flows. It has one incoming edge and multiple outgoing edges. The edges coming into and out of a fork node must be either all object flows or all control flows. The functionality of join nodes and fork nodes can be combined by using the same node symbol
6. The activity “final node” is a final node that stops all flows in an activity. Whereas a flow final destroys all tokens that arrive at it and has no effect on other flows in the activity.

The Activity diagram also provides the modelling of various kinds of actions, e.g. the “Accept event action”, “Send Signal Action” or the “Raise Exception Action”. Accept event actions, handle event occurrences detected by the object owning the behaviour. Send Signal Action is an action that creates a signal instance from its inputs, and transmits it to the target object, where it may cause the firing of a state machine transition or the execution of an activity. Actions can be initiated at a specific date. This wait time action is notated with an hour glass. Raise Exception Action is an action that causes an exception to occur (with a “lightning bolt” symbol). To constrain and show a view of the contained nodes, nodes and edges are divided into partitions, which often correspond to organisational units into a business model (see example below). They are represented by swim lane notations.

Modelling an activity diagram means that the use of action and activity notation is optional. A textual notation may be employed instead.

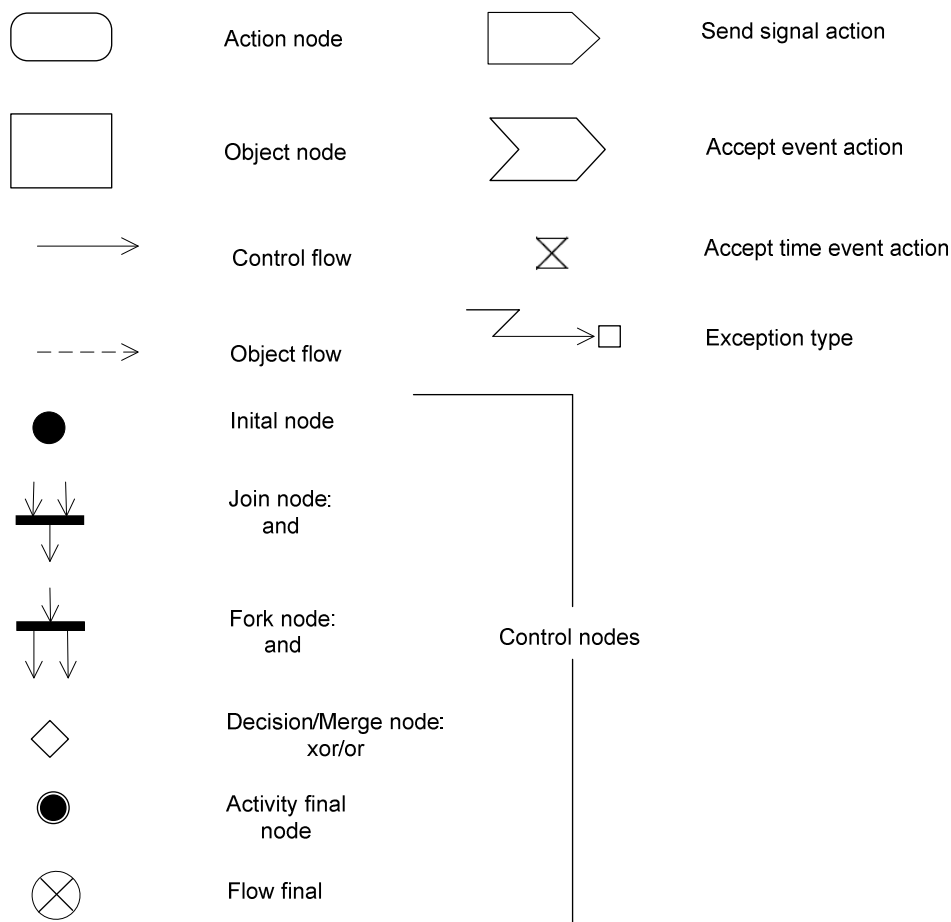


Figure 7: Constructs of UML AD

Metamodel:

The figure below shows a part of the metamodel for Activity diagrams. It has been created by abstracting and combining miscellaneous activities and modelling elements defined in the UML 2.0 Superstructure Specification. It represents the most important metamodel elements, which means activities, flows, objects and control nodes.

Activity groups are a generic grouping construct for nodes and edges. Nodes and edges can belong to more than one group. Those groups have no inherent semantics and can be used for various purposes. Subclasses of Activity Groups may add semantics.

An activity node is an abstract class for the steps of an activity. It covers executable nodes, control nodes and object nodes. (BasicActivities) Nodes can be replaced in generalisation and (CompleteActivities) be contained in interruptible regions.

Activity Edges are an abstract class for the connections along which tokens flow between activity nodes. They cover control and data flow edges. Activity edges can control token flow.

Activity partitions are introduced to support the assignment of domain-specific information to nodes and edges.

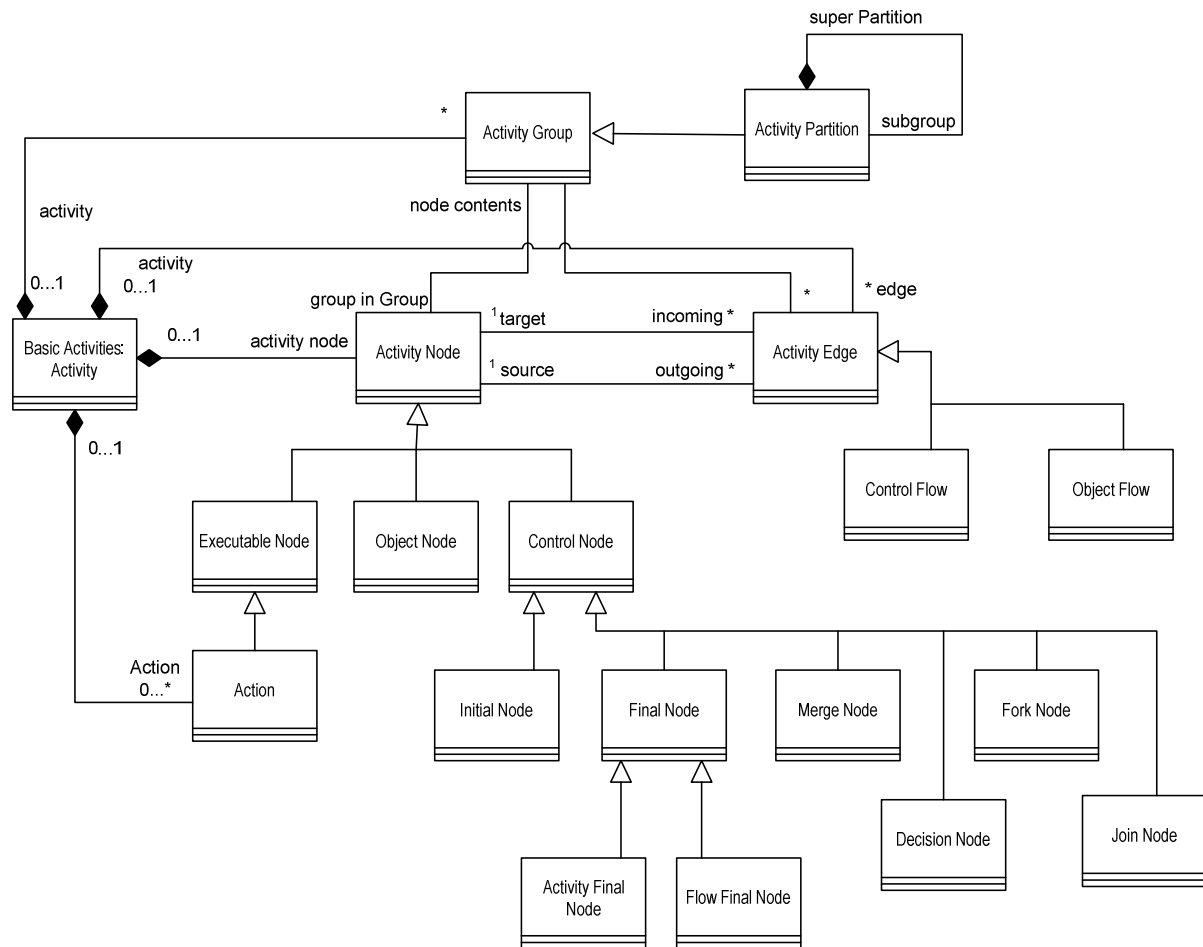


Figure 8: UML AD Metamodel

According to: Bordbar, Behzad and Staikopoulos, Athanasios: On Behavioural Model Transformation in Web Services. In: Wang, Shan (Ed.): Conceptual Modeling for Advanced Application Domains. Lecture Notes in Computer Science no. 3289, Springer, Berlin, p. 667 – 678 (670).

For more introduction information and semantic framework see the “Action” (p. 215 – 227) and “Activity” (p. 288 – 299) metaclasses provided in the specification of the OMG. There you will find metaclasses for following packages:

Package Basic Actions:

- Basic actions
- Basic pins
- Basic invocation actions

Package Intermediate Actions:

- Intermediate invocation actions
- Object actions
- Structural Feature Actions
- Link identification
- Read link actions
- Write link actions
- Miscellaneous actions

Package Complete Actions:

- Accept event actions
- Object actions (Complete actions)

Package Fundamental Activities:

- Fundamental nodes
- Fundamental groups

Package Basic Activities:

- Nodes (Basic Activities)
- Flows
- Elements

Package Intermediate Activities

- Object nodes (Intermediate Activities)
- Control nodes (Intermediate Activities)
- Partitions
- Flows (Intermediate Activities)

Package Complete Activities:

- Elements (Complete Activities)
- Constraints (Complete Activities)

<ul style="list-style-type: none"> • Link identifications (Complete actions) • Read link actions (Complete actions) • Write link actions (Complete actions) 	<ul style="list-style-type: none"> • Flows (Complete Activities) • Object nodes (Complete Activities) • Control pins • Data stores • Parameter sets • Control nodes (Complete Activities) • Interruptible regions
Package Structured Actions:	Package Structured Activities:
<ul style="list-style-type: none"> • Variable actions • Raise exception actions • Action input in 	<ul style="list-style-type: none"> • Structured nodes
	Package Complete Structured Activities:
	<ul style="list-style-type: none"> • Structured nodes (Complete Structured Activities)
	Package Extra Structured Activities:
	<ul style="list-style-type: none"> • Exceptions • Expansion regions

Modelling Example:

For comparison reasons, the same warehouse process as in chapter 2.3.2.1 is presented in following paragraphs including some additional modifications:

- When trying to store the result into the accounting control system a bug occurs and the data cannot be stored
- If the invoice entries are not correct, not only the delivery is sent back, but a complaint letter is sent to the supplier as well.

AD Warehouse Management

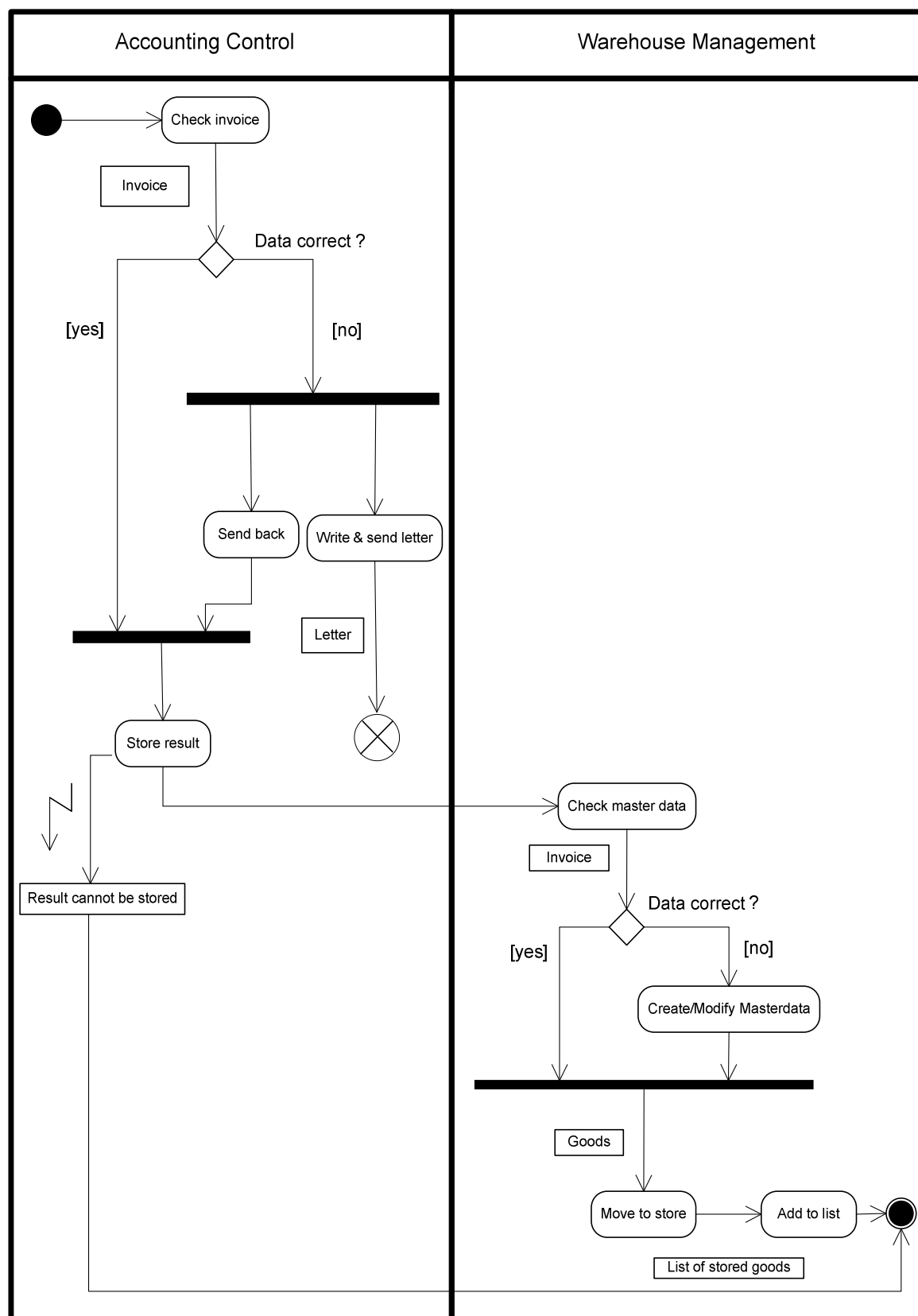


Figure 9: UML AD modelling example

Part of Framework:

The activity diagrams are part of the UML Framework. The UML is an internationally standardised specification language for object modelling. It was invented by the Object Management Group (OMG), which enhances the language continuously. The UML includes a graphical notation and is not only used for software modelling, but also for business process modelling, systems engineering modelling and representing organisational structures.

Supporting software tools:

Rational Rose (IBM), VISO (Microsoft), Visual Paradigm for UML (Visual Paradigm), Altova UModel® 2007 (Altova)

Evaluation:

The activity diagram can describe business activities and the control flow between them. But it does not support Business Rules. As the activities can be labelled by users the terms of the UML AD are familiar to business users. It is rather easy to understand and to learn. As the activity diagram is part of the UML it is a defined standard. As the example above shows it has a graphical notation and appropriate tool support by a large variety of tools. Depending on the storage format, an activity diagram can be interchanged between different tools. The syntax of the UML AD is soundly defined by the UML specification. It is able to handle complex models and scenarios. But it does not support the description of workflows explicitly. Used in an MDA-approach the UML-AD contains relevant information but other diagrams for example for data modelling are needed in order to represent all relevant information of the CIM level. UML-AD does not support the integration of different modelling views. The evaluation of activity diagrams is summed up in the following table:

Table 6: Evaluation of UML AD

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	✓	8	(✓)
2	✗	9	✓
3	✓	10	✓
4	✓	11	✗
5	✓	12	(✓)
6	✓	13	✗
7	✓		

2.3.2.5 Unified Modelling Language Use case

Acronym:

UML Use case

Specification document(s):

There are two main specification documents where UML notations are defined, both of which can be accessed from: <http://www.uml.org/>

- The UML infrastructure provides an architectural (or a foundational) basis for the UML notations.
- The UML Superstructure (currently version 2.1.1) defines the constructs for the UML notations with Use Cases being described in Chapter 16.

Specifying organisation:

Object Management Group (OMG)

Evaluated version(s):

Version 2.1.1

Focus of the language:

Use cases are an example of a scenario based approach to requirements, in that they decompose the overall functionality into smaller scenarios or 'stories', (called use cases) where the user (and their interactions with a system) is the focus of concern. That is, each 'use case' describes a particular interaction between the user (termed the actor) and the system. The use case is intended primarily for describing these circumstances of system usage (as interactions between the system and external users of the system), without describing internal (system issues).

Although often used for both requirements and specification purposes, use cases actually match Jackson's definition of specification, and are better suited to this purpose. This is because use cases describe the interactions (across a system boundary) between an actor (or actors), and a system. This interface, between problem domain, and system (or machine to be built) is Jackson's definition of specification [MJ95]. Indeed, the UML further suggests that use case descriptions should proceed such that each action of an actor elicits a

meaningful response, and ‘well written’ descriptions often exhibit ‘adjacency pair relations’, which, for example, relate the actions of actors with corresponding system responses. Hence, an ideal usage is that requirements are described in text or with supporting models and that use case descriptions then form the bulk of the specification.

The simplicity (and accessibility) of the use case notations means that use cases are often used within requirements elicitation, since system users and customers can pinpoint described usages that match (or do not match) their reality. Indeed, the UML tends to blur the distinction between requirements and specification, and thus views use cases as a vehicle for the entire requirements phase. Furthermore, use cases are also used for describing business processes where use case models depict business stakeholders and the tasks that they perform within the business.

It is, perhaps, worth mentioning that whilst the use case diagram is well understood (see below), and agreed, the ideal form of the supporting use case description has been an area of much debate, with many suggestions for improvements and extensions having come from both academics and practitioners. Given that the diagram merely shows a partitioning of functionality, whilst the description actually forms the specification, it seems sensible, here, to review some of the literature on such improvements and extensions.

Description of the language and its model elements:

A use case diagram comprises actors (external system users) and the use cases (or system functions) that the actors take part in. Communication between an actor and the use case is shown as a directed arrow from the actor to the use case, or undirected arrow to show bi-directional communication. Direct associations between actors are not allowed in a use case model (though, of course, would be vital to describe requirements or processes within the problem domain). However, it is possible to define a generalisation relationship between actors where one actor is a special case of another actor.

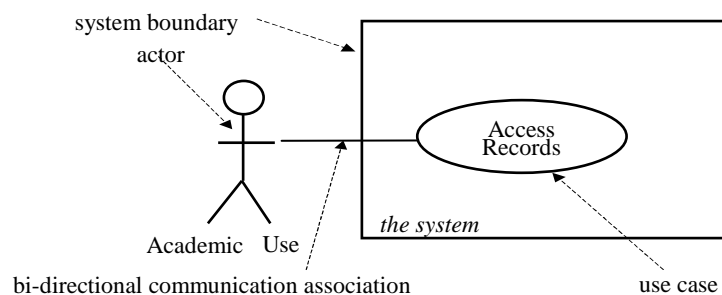


Figure 10: Relationships in use case diagrams

There are two special types of associations within use cases; the <<include>> relationship and the <<extend>> relationship.

In brief, an <<include>> relationship is used to avoid rewriting the same behaviour several times, i.e. common behaviour is placed in <<include>> use cases. An <<include>> relationship means that the base use case explicitly uses the behaviour of another use case at a specified location in the base. Hence, the <<include>> relationship provides a means for factoring out commonly needed behaviour that may be reused within another use case. The included use case never stands alone, but is only instantiated as part of some larger base that includes it. For example, consider an online shopping site. This might have a number of use cases, such as ordering or tracking orders, which all need to validate the user. Hence, they would ‘include’, and thus, reuse a validate user ‘use case’.

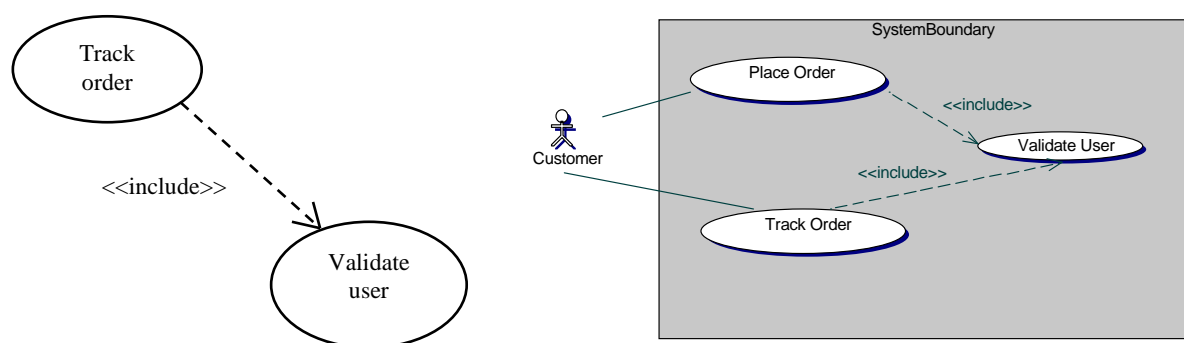


Figure 11: Notations for <<include>> and an example usage

In contrast, the <<extend>> association represents an exception or alternative course of action in a given use case, which will only occur under certain conditions (or states) of the base use case. Hence, <<extend>> use cases show that some rare or (specialized) behaviour is occurring because the main use case has reached an unusual condition. The extension use case must be completed before control is returned to the base use case.

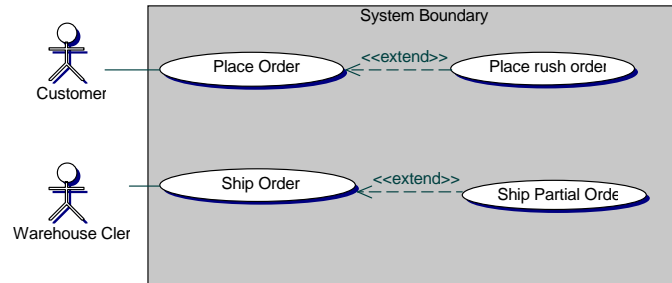


Figure 12: <<extend>> example usage

Although many texts often focus on diagrams, in practice it is the use case description that is the crucial content of the specification. Each use case model (the diagram) should be accompanied by a description, which specifies the detail of the use cases. The description will also include other important elements of a use case, e.g. the pre and post state for that entire use case, some context and so on (see below). The pre and post states of a use case respectively define the state prior to occurrence of the use case and state after the occurrence of the use case. Modellers will write use case descriptions to outline the detail of actions within a use case diagram. The description is often considered more suited to system specification than the diagram [CA01]. The reason for this is that a modeller is able to provide details regarding interactions among the system and its external users in much more detail with the use case description as compared to the diagram. Hence, modellers are often encouraged to provide associated descriptions alongside their use case diagram models.

The UML gives some guidance for the structure and layout of use case descriptions and many other authors have suggested either guidelines or templates of varying degrees of complexity. The standard guidance is that the description consists of:

- **Use Case Title:** Name of the Use Case
- **Actors:** Those involved within the particular use case
- **Context:** A sentence or so, setting the scene.
- **Pre-condition:** For the entire use case to take place
- **Main Flow of Events:** Each use case step should be shown on a numbered line.
- **Alternative (or Exceptional) Flow of Events:** Where each alternative is numbered according to the step from which it deviates.

However, further guidelines, on style, structure and content, are typically used in addition to the structure suggested above. Empirical studies on such guidelines all appear to suggest that the adoption of such guidance does lead to improvements in the ‘quality’ of the resulting description; typically in terms of increasing comprehension, reducing ambiguity, and so on. (The only main differences of opinion, being, not surprisingly about which sets of guidelines perform best). Hence, in considering exceptions (later), one such set of guidelines (the CP rules) will be described briefly.

Metamodel:

The main metamodeling package provided by the OMG is *Core*. The Core package is meant to be reused by other OMG packages such as the UML and CWM packages. The UML package itself is comprised of various packages that organise its modelling ideas into logical categories. For instance, both the **Actors** and UseCases in the metamodel shown below inherit from the BasicBehaviors package. Additionally, UseCases are shown to be contained in a Classifier, which in turn inherits from the Kernel package. Use case modelling concepts such as the extend and include relationships are also shown.

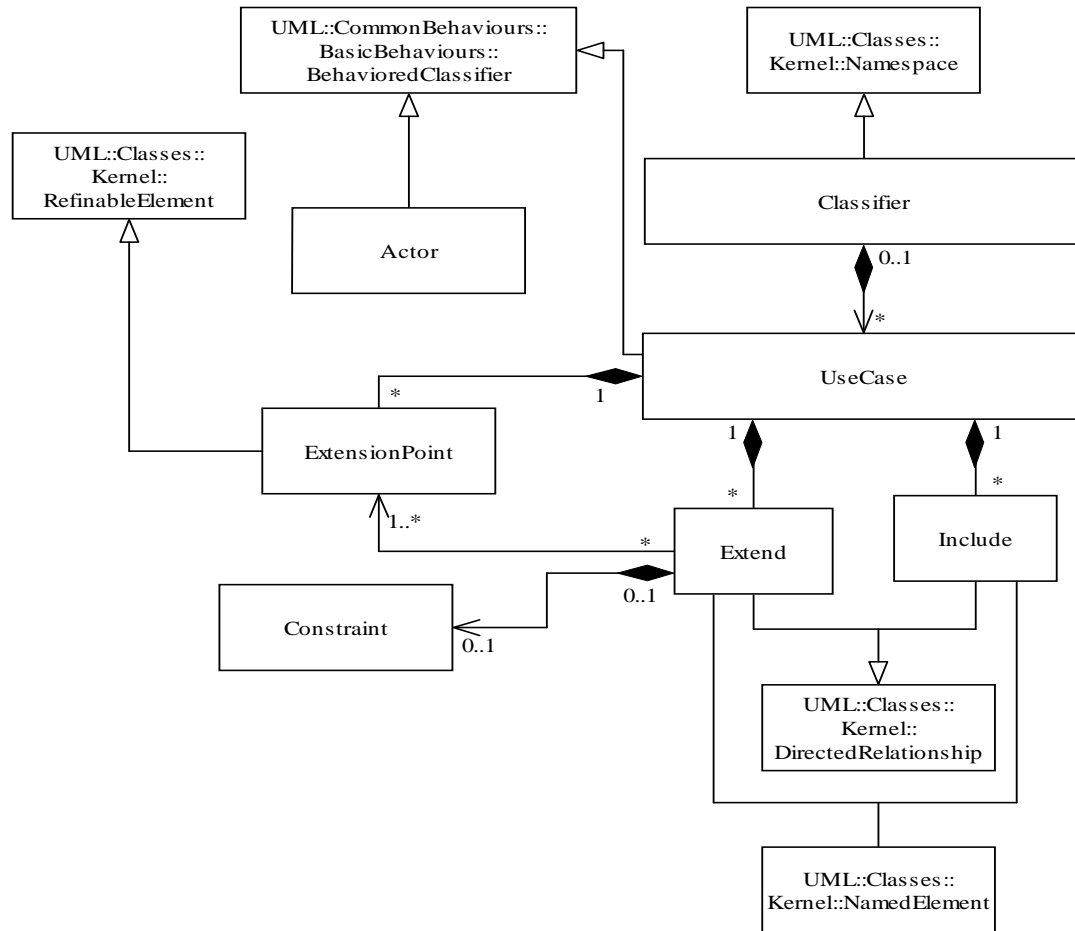


Figure 13: Use case modelling concepts - found in [OMG07]

Modelling Example:

Consider the commonly cited ATM use case (also found in the D1 delivery document). The use case describes interactions between an ATM user (the customer) and the ATM system. Again, the description is written using guidelines (the CP rules):

ATM use case description

Preconditions:

The ATM has cash available

There is no card in ATM's card reader.

1. Customer inserts card.
2. ATM requests Customer's PIN.
3. Customer enters PIN.
4. System validates PIN.
5. ATM displays options: Withdraw Cash; Withdraw Cash with Receipt; Check Balance; Order Statement; Make Deposit; Change PIN.
6. Customer selects "Withdraw Cash with Receipt".
7. ATM displays dispensing amounts
8. Customer selects desired amount.
9. System checks balance.
10. ATM ejects card.
11. Customer takes card.
12. ATM dispenses currency.
13. Customer takes currency.
14. ATM displays "wait for receipt message".
15. ATM prints receipt.
16. Customer takes receipt.
17. ATM displays welcome message.

Postconditions

The ATM cashbox has less cash.
ATM notifies customer bank

Alternatives

- 7.1 Customer selects “other amount”.
- 7.2 ATM displays instruction message. (For manual entry)
- 7.3 Customer enters amount. (ATM goes to 8).

Exceptions

- 1. The card is invalid; ATM ejects card and displays message.
- 3. The PIN is wrong; ATM goes to step 2 twice more (to allow Customer to retry); on the fourth try it retains the card and displays message.
- 9. Account balance invalid. ATM ejects card and displays message
- 10. ATM sounds alert (reminds customer to take card).

The use case description for the ATM has a main path (also termed sunny day scenario) which consists of steps 1 to 17. An alternative path is shown, which is associated with step 7 of the main path. The alternative path simply describes a different path in the use case that may be followed by the external user (actor) during the performance of the use case.

Exceptions may also occur during system use, and one of the exceptions in the ATM use case relates to the ATM card being invalid (this to be dealt with at step 1), the PIN being incorrect (this to be dealt with at step 3), etc.

Part of Framework:

Use case diagrams are part of the UML notation set. Other notations within the UML include class and object diagrams, activity diagrams, statecharts, sequence diagrams, collaboration diagrams, component diagrams and deployment diagrams. A widely recognised process model for the UML notations is the Rational Unified Process (RUP). The RUP situates use cases at the centre of object oriented software development, thereby arguing (correctly) that requirements analysis is a key stage in software development, and that use cases provide an accessible means for requirements and specification. Hence, many observe that derivation of system structure (e.g., class and object diagram) is informed by the use case model.

Supporting software tools:

Many tools support the construction of UML notations, including use cases. For example, Rational Rose is widely used in industry and within learning institutions (e.g. universities) for UML modelling. Other tools include Together, Objectteering, MS Visio.

Evaluation:Strengths and Weaknesses of Use Cases

A cynical view of the dominance of use cases as a requirements phase approach might be that their inclusion in the UML gives them a credibility that means their adoption is assured. However, it is clear that use cases do offer some genuine advantages and that they are well liked by many requirements phase users. Their main advantages are:

- They offer a simple and accessible approach which can be understood by many stakeholders
- In placing the user (actor) at the centre of the scenario, they allow users to understand their processes from their perspective.

In brief, our view might be characterised as one of pragmatism, in that whilst we recognise their weaknesses, we also realise that people will use them for genuine reasons of accessibility and so on, Hence, a sensible approach, is that rather than reject them for their flaws, we attempt to offer suggestions for improvements in their usage. This can best be summarised by an extract from a recent conference paper (SQM 2006 and Software Quality Journal).

“Use cases [1] are now a well established and popular method of specification [2, 3]. Indeed, the intuitiveness of the notation is typically cited as a major reason for their large scale adoption [4]. The freedom to write descriptions that are accessible to a breadth of audiences allows greater contribution from a variety of stakeholders, which can significantly improve the effectiveness of validation [5, 6].

Despite such widespread usage, there are still many suggestions that the application of use cases [7], and particularly use case descriptions, is problematic [8]. Reservations about their utility tend to fall into one of two categories. There are those who consider that the notation itself does not provide enough power or expressiveness to describe the nuances of specification [9]. Typically, these authors suggest that the notation

should either be augmented (such as with pre and post conditions) [10, 11] or supplemented (with other diagram types) [9]. Similarly, there are those who suggest that lack of prescription in the application of the use case is the problem [12, 13]. That the very freedom and expressiveness allowed by what is, in essence, a structured form of natural language leads to problems in structure and comprehension [14, 15, 16]. The authors feel that both categories have valid arguments. Augmentations to the original use case description may help to solve specific issues, for example, addition of pre and post conditions can highlight dependencies amongst events [17, 18]. However, a focus on improving the quality of ‘standard’ use case descriptions may be more in keeping with the ethos of providing an accessible notation for the requirements phases [12, 19]”.

The evaluation of Use Cases is summed up in the following table:

Table 7: Evaluation of UML Use Cases

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	(X)	8	✓
2	X	9	✓
3	✓	10	✓
4	✓	11	X
5	✓	12	(X)
6	✓	13	X
7	✓		

Extensions

Guidelines and Rules

Rather than reiterate the many arguments for use case guidelines (or refer to supporting empirical evidence) the reader is referred to two recent and related papers on the approach, both in the Software Quality Journal. The first introduces a subset of writing guidelines and then relates empirical work to assess the impact of such guidelines, while the second (earlier paper) describes the theoretical background to the production of assessment criteria for use case descriptions.

1. Phalp, K.T., Vincent, J.V and Cox, K. (2007), Improving the Quality of Use Case Descriptions, Special Issue of the Software Quality Journal, to appear.
2. Phalp, K.T., Vincent, J.V and Cox, K. (2007), Assessing the Quality of Use Case Descriptions, Accepted for the Software Quality Journal, February 2006

However, in order to appreciate the general approach, without recourse to such a detailed treatment, it may be useful to outline a set of use case writing guidelines (our own CP rules), such that the reader is familiar with the general approach taken. Hence, the following is taken from a set of lecture slides which are used at Bournemouth to introduce the CP rules (students then use such rules in describing use cases both for individual and project work).

CP Use Case Writing Rules

CP Structure guidelines

Structure 1: Subject verb object. For example,

The operator presses the button.

Structure 2: Subject verb object prepositional phrase. For example,

The system reminds the operator to save all the open files.

Structure 3: Underline other use case names. For example,

The user makes a new equipment request.

CP style guidelines

These are applicable to all sentences in the use case description.

Style 1: Each sentence in the description should be on a new, numbered line. Alternatives and exceptions should be described in a section below the main description and the sentence numbers should agree. For example:

Main Flow

1. The patient record appears on the screen.
2. The doctor enters the patient's new address.

Alternative Flow

2. The doctor deletes the patient's record.

The alternatives go below the main flow and the sentence numbers agree (2 and 2).

Style 2: All sentences are in present tense format. The use case should describe events and actions in the here and now, not the past or the future. Some examples:

The operator presses the button.

The checkout operator enters the amount.

Style 3: Avoid using adverbs and adjectives, these add unnecessary clutter to the description and give values that are difficult to quantify. Only use negatives in alternative and exceptional flows of events. Avoid using pronouns (e.g., he, she, it, we, their, etc.). Examples:

The doctor writes the prescription slowly (adverb).

The patient swallows the big pill (adjective).

The patient stands next to the doctor.

He puts the prescription in his pocket (pronouns).

Who is "he"? Whose pocket is "his"? Write proper nouns / names instead.

Style 4: Give explanations if necessary. Explanations should be enclosed in brackets:

The librarian enters the borrower's details (details are: name, address, phone number, library card number).

Style 5: There should be logical coherence throughout the description. The sentence you are writing now should refer to something in the last sentence or a previous sentence, if possible. We understand the use case better this way.

1. The cat sits on the mat.

2. The mat belongs to Fred.

The mat in (2) coheres to mat in sentence (1).

Style 6: When an action occurs there should be a meaningful response to that action. For example, when there is an input there should be a response to that input somewhere in the use case, usually immediately. This makes sure we do not forget to respond to any action in the use case description.

The doctor enters the patient's record identification number.

The system displays the patient's record.

Style 7: Underline other use case names that are included or extend the main the description.

The user makes a new equipment request.

The EDUCATOR approach and tool-set (addition of pre and post states)

We reiterate again here that one of the main benefits associated with the use case description is that it is accessible to non-technical stakeholders of a system. An additional and related benefit is that the description provides scenarios of system usage from the point of view of the system users. Whereas such a view is important during validation, the use case description does not offer a means for considering dependencies among use case steps. That is, it is not possible to explicitly describe neither dependencies among use case actions, nor dependencies across use cases. For instance, in the ATM use case outlined above, it is not clear to the modeller whether step 4 is dependent on step 3, step 2, step 1, all of them (3, 2 and 1) or none. The Educator approach addresses this shortcoming of use case descriptions by proposing the augmentation of each constituent step with pre and post states as a way to facilitate reasoning about dependencies among actions. The EducatorTool (support tool for the Educator approach) provides support for authoring descriptions that adhere to the approach,

but more importantly, the tool provides enactable capability to allow developers and other stakeholders to step through the logic of the description as a means to validating implied behaviour.

The key argument of the Educator approach is that considering dependencies among use case actions (intra-use case dependencies) and dependencies among actions of distinct use cases (inter-use case dependencies) is crucial to validation of the description against stakeholder expectations. This argument is similar to that motivating previous work on writing guidelines. For guidelines, the argument is that adopting common writing rules does enhance common understanding of the description, which (common understanding) is crucial to validation. For the Educator approach, the argument is again, that, facilitating reasoning about dependency issues allows stakeholders and developers to validate the description by considering the ramifications of the implied behaviour of the description.

2.3.2.6 Role Activity Diagrams

Acronym:

RADs

Specification document(s):

The main RAD notation is outlined in [MO95]. Other articles where RADs are discussed include [KP98] and [GA98].

Specifying organisation: There is no standardization organisation that specifies RADs.

Evaluated version(s):

Current Riva process version of notation can be found under
<http://www.the-old-school.demon.co.uk/vc/veniceresources.htm>

Focus of the language:

Role Activity Diagrams (RADs) are widely used for modelling business processes. The central notion is that the business process and its activities (either actions or interaction) are grouped into a set of interacting roles, 'which describe the desired behaviour of individual groups, or systems'. [MO92]. Role Activity Diagrams were designed specifically to include constructs important to the business process modeller, whilst still retaining a small and intuitive set of primitives. Hence, the notation includes only a few main concepts: roles, action, interactions, states, case refinement (choice) and part-refinement (parallel). In addition, later versions include notations for triggers, starting new instances of roles, and to denote the driving role in an interaction.

The notation has relatively strong semantics, being developed originally from Petri-nets, and whilst they appear somewhat flowchart like in nature, Role Activity Diagrams are actually a simple state machine, with all of the main constructs being related to the state of the role.

For example, actions can be defined formally in terms of the pre and post states of a role. With a RAD model of a business process, a role has state, and once the role undertakes an activity, it moves to the next state. Similarly interactions can act as points of synchronisation where the interaction is controlled by the pre states of the interacting roles. Other authors have also used this formal basis of Role Activity Diagrams to suggest more formal versions, for example, the RolEnact notation maps to the main RAD constructs, whilst providing an enactable capability to enable prototyping of process behaviours.

RADs afford the modeller the power to show performance of concurrent business processes and choice among processes.

Description of the language and its model elements:

The main elements of the RAD notation are summarised as follows:

Roles – a role defines a set of activities that help to achieve a goal when performed together. Roles may map to actual organisational roles (e.g. clerk) and describe types or classes of behaviour. For example, a cashier role might actually be taken on by a number of different people at different times. Similarly, an individual (actor or resource) might take on different roles at different times. A role is considered to be independent of other roles, acting in parallel, but communicates through interactions (where interactions are the only formal point of synchronisation). 'A role involves a set of activities which, taken together, carry out a particular responsibility or set of responsibilities' [MO 95].

RoleName**Figure 14: Notation for roles**

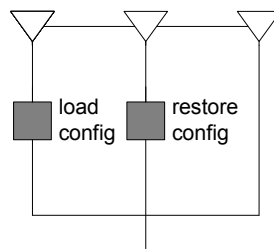
Actions- An action is an activity which a role carries out in isolation. Actions are shown within roles as a solid square with the name of the action against the square. Formally, an action moves the role from its current (or before) state to its next (after) state.

**Figure 15: Notation for actions**

Interactions - Interactions among roles depict the way in which business stakeholders interact in the business setting where the processes are undertaken. Interactions are shown using a horizontal line between the activities at which the interaction is occurring across the involved roles. Formally, the consequence of an interaction is that all of the roles involved move from their current state to their next state, and states can be added to highlight the synchronization or control of interactions. (The example shows the post states for an interaction). Interactions are synchronous, even though they may occur over time. One may also denote the driving, or initiating role typically by showing its action square with diagonal shading, whilst those passive ends of the interactions are left with an un-shaded square. Note that in reality such distinctions can be somewhat artificial, for example, whilst an interaction such as sending something clearly might have an initiating role, where interactions are conversations or perhaps even meetings, the initiating role is less clear.


**Figure 16: Interaction modelling example****Case refinement (Alternative Paths)**

Within RADs, choice between activities (case refinement) is shown using linked inverted triangles. There may any number of such threads, one of the threads, or alternative cases, being chosen.

**Figure 17: Notation for case refinement****Part refinement (Concurrent Paths)**

Often, a modeller may want to show activities that may occur in parallel as sub-threads of the main activity. These are shown using point-up triangles shape. Each thread, in effect, is independent of the other, and operates in parallel, only becoming synchronized when the threads rejoin. Indeed, within RolEnact (an enactable version of RADs), part refinement is achieved by considering each thread as if it were a separate role, threads thus rejoining via an additional interaction.

**Figure 18: Notation for part refinement**

Replications of part-refinements are denoted with the  symbol.

States

The original RAD notation simply used the vertical lines between actions or interactions to denote state. Other variants then added a simple circle to denote state, and this addition is now accepted as part of the notation. However, many modellers tend to use this addition sparingly, perhaps only adding states to the model where to clarify synchronization or dependency issues. Martin Ould, in the latest version of the Riva notation, uses these state circles only to denote states that relate to specific ‘goals’, for example completion of a project or important business process element would be denoted with an explicit state.

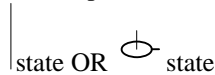


Figure 19: Various notations for states

Another further option is to use special states to denote the end of threads. Historically, many modellers have used the explicit state symbol, with some named state such as end, or final to show this final state. Indeed, the first commercial RAD tool, used to use a state with UK stop road-sign image. Similarly, the latest Riva version of the notation has a specific end of thread or ‘stop’ symbol



Figure 20: Notation for final state

Iteration

Iteration is shown either by literally looping back to a previous part of the role, or sometimes simply by using the same state name as an earlier state, that is, using the RAD as a state machine.



Figure 21: Notation for Iteration

Triggers

Triggers are used to denote events, usually external which will act as triggers for actions, that is, will change the state of the role. For example, a typical trigger might be where a project is started.



Figure 22: Notation for triggers

New Role

In many processes, one role may initiate or start another, e.g., a director may appoint a project manager. Starting a new role is shown with a simple crossed square. Again one variant of this is to show an interaction line to the new role, where the crossed square replaces the driving square for a normal interaction.



Figure 23: Notation for new roles

Metamodel:

We show the RAD metamodel as a set of classes of RAD model elements with relevant associations among them. Within the metamodel, the RAD Model class is a set of role model elements. A RAD Model strongly contains roles. A role is a container for a Node model element, where a node is a superclass for State, CaseRefinement, Interaction and Action. Action nodes are either activities, part refinements, external events or a point of synchronisation. Actions are undertaken by the containing role, while Interactions are undertaken by the initiating role and participating role. In the metamodel, ActionType is defined separately as a type with which to qualify an action. For instance, an Action that is a part refinement provides the modeller a means to show parallel execution threads, whereas case refinement actions provide the means to show choice between activities. And of course, activities are the basic unit of work for a role.

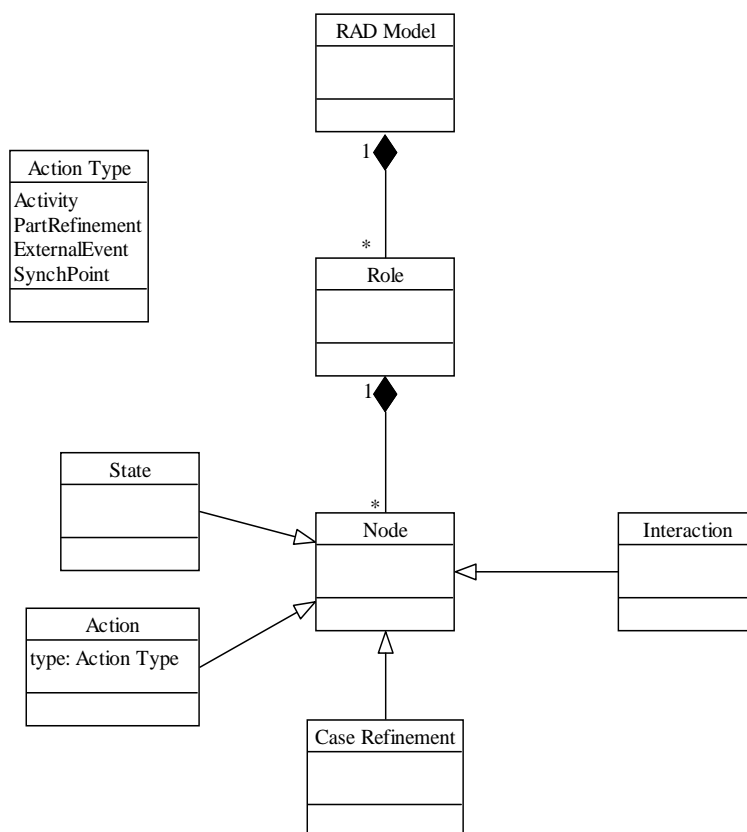


Figure 24: RAD modelling elements - found in [CB05]

Modelling Example:

RADs are a visual notation for representing business process models. The example RAD model shown below constitutes three roles: the Divisional Director, the Project Manager and Designer roles. Roles group together activities associated with the role (or the activities that a role is responsible for are grouped in a role). For instance, the Divisional Director approves a new project, then agrees the terms of reference (TOR) with the Project Manager. In turn, the Project Manager writes TOR for the Designer and delegates the project to the Designer. Upon agreeing TOR with project manager, the Designer can take part in two parallel threads. These parallel threads involve choosing a method for producing the design and an estimate (e.g. cost) for it. Essentially, the activities undertaken by the roles are interdependent, and it is seen from the RAD model that the Project Manager's action of producing debrief report cannot happen until the Designer has passed actual effort figures to the project manager.

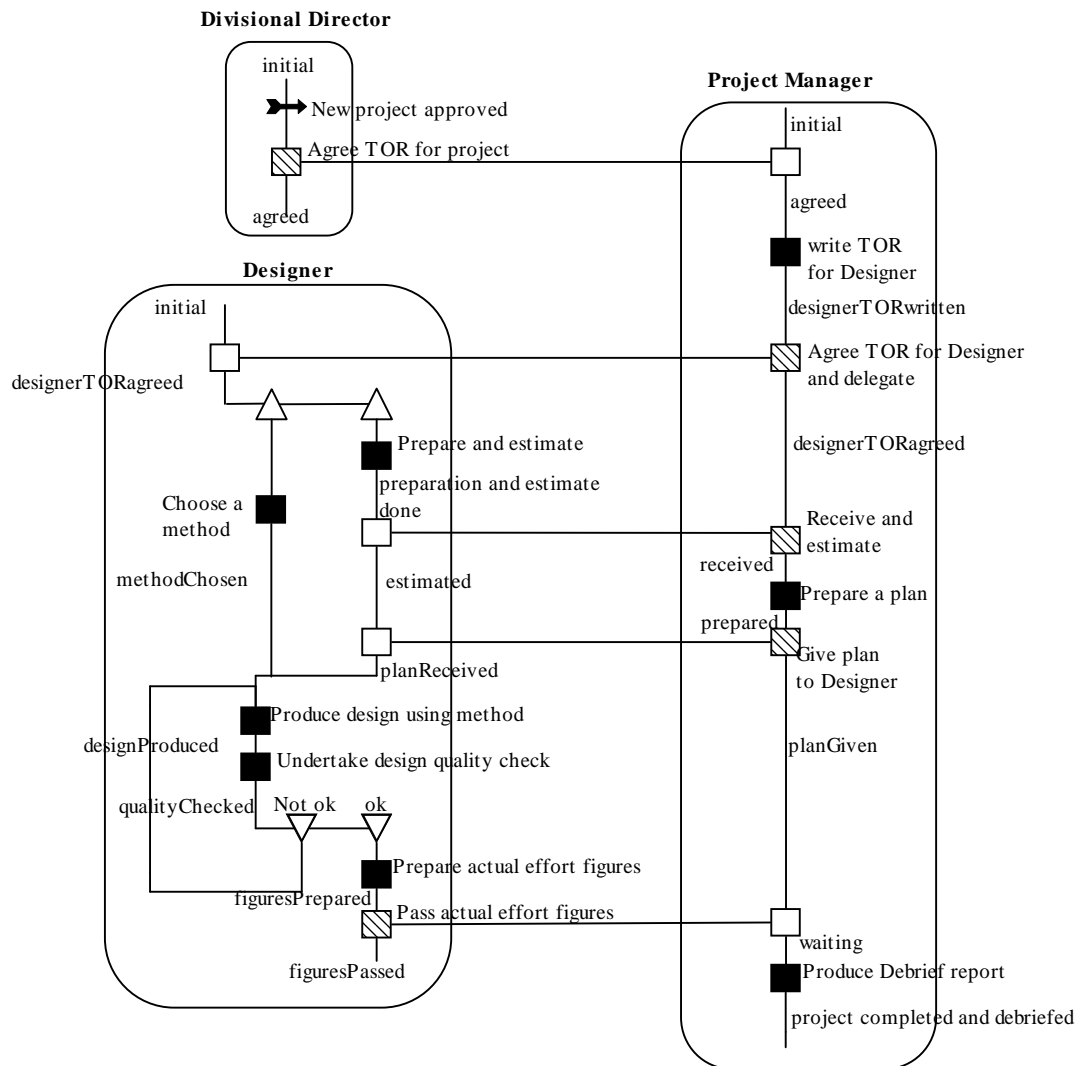


Figure 25: RAD modelling example

Part of Framework:

RADs are not part of a framework.

Supporting software tools:

Visio, SmartDraw, Riva tools (Instream) and RADRunner tools

Evaluation: RADs are an intuitive notation for modelling business processes. It is argued in [GA98] and [KP98] RADs provide a business view of roles, their responsibilities and interactions among roles without biasing the model toward future consideration for any automation of (all or) part of the process. Two reasons for proposing RAD based CIM modelling are the expressiveness of the RAD notation on the one hand and its ease of understanding (by business people) on the other.

One way to reuse RAD modelling concepts for CIM is to consider the representation of the RAD metamodel using EMF's Ecore. The aim would be to create the RAD metamodel (e.g., as annotated, Java interfaces) to capture model information that is not expressible directly in standard Java. VIDE would then seek to provide Ecore representations of RAD models, and possibly, an (Eclipse) editor plug-in for RAD-based CIM modelling can be provided. Hence, key usages of RADs would be to allow business people to develop rich business process models leveraging the expressiveness of RADs (e.g. ability to depict interactions, and actions, including parallel threads and choice), but also provide the VIDE developer a means to transform aspects of the RAD model into a PIM model.

Rule-based mappings between RAD model elements and PIM model elements could also be considered, especially given existing MDA languages such as ATLAS (and associated AMW toolset) that already have this

type of model transformation view (producing a weaving model for deriving a target model given a source model). For example, in some cases, it may be that roles (from RAD) would be further developed as class diagrams within a PIM model. However, such mappings would have to be flexible (that is, allow the modeller to use the RAD model to determine which parts of it is useful to form parts of PIM).

The evaluation of RAD is summed up in the following table:

Table 8: Evaluation of RAD

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	✓	8	(✓)
2	✗	9	(✓)
3	✓	10	(✗)
4	(✓)	11	✗
5	(✗)	12	(✓)
6	✓	13	✗
7	✓		

2.3.2.7 Business Process Execution Language for Web Services

Acronym:

BPEL4WS – For discussions not being concerned with a specific version, the term “BPEL” is sufficient.

Specification document(s):

Business Process Execution Language for Web Services Specification, version 1.1 dated May 5, 2003

Specifying organisation:

Organisation for the Advancement of Structured Information Standards (OASIS), since April 2003

Evaluated version(s):

Version 1.1

Focus of the language:

BPEL is a business process modelling language for “programming in the large”. It can be described as a programming language and can be executed directly, but is more likely to be automatically generated from workflow diagrams.

BPEL4WS allows the definition of both business processes that make use of Web services and business processes that externalize their functionality as Web services. Thus BPEL's messaging facilities depend on the use of the Web Services Description Language (WSDL) 1.1 to describe outgoing and incoming messages. BPEL is an orchestration language, not a choreography language and is serialized in XML.

Description of the language and its model elements:

Business processes specified via BPEL describe the exchange of messages between Web services. To be instantiated, each business process must include at least one "start activity". This must be an initial activity in the sense that there exists no basic activity that logically precedes it in the behaviour of the process. Activities are the actions that are performed within a business process. A flow is a directed graph with the activities as nodes and so-called links as edges connecting the activities. The flow specifies the order of the activities performed within the process.

The token „Activity“ can be either a basic activity, a structured activity or a scope.

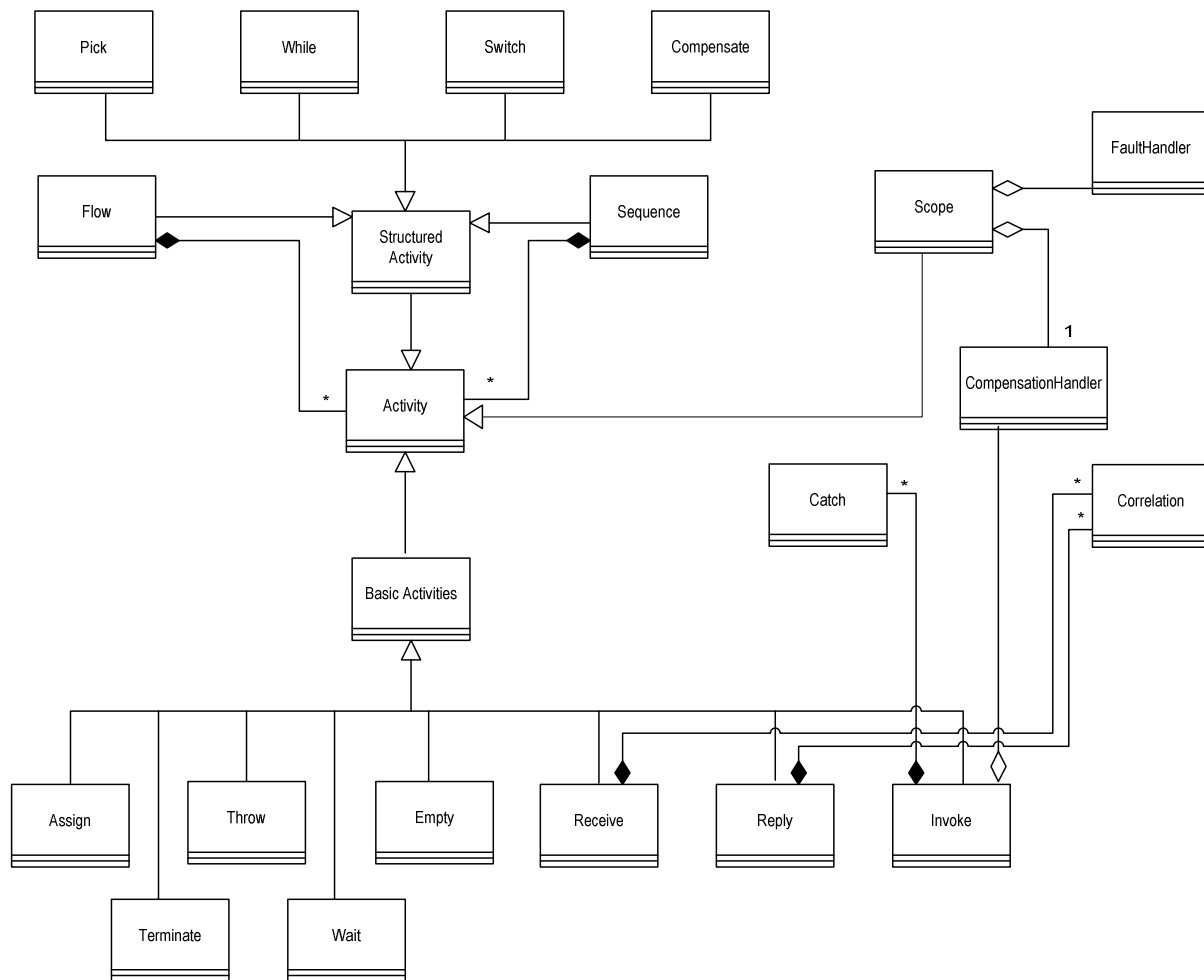


Figure 26: BPEL Metamodel of the Activity Elements

Basic Activities are elementary activities, which are not composed by other activities. There exist six Basic Activities within BPEL4WS:

- **assign** – modifies the content of variables: it can be used to update the values of variables with new data. An <assign> construct can contain any number of elementary assignments.
- **invoke** – synchronous (request/response) or asynchronous call of a Web Service: The <invoke> construct allows the business process to invoke a one-way or request-response operation. A fault response to an invoke activity is one source of faults, with obvious name and data aspects based on the definition of the fault in the WSDL operation. The optional fault handlers attached to a scope provide a way to define a set of custom fault-handling activities, syntactically defined as catch activities. Each catch activity is defined to intercept a specific kind of fault.
- **receive/reply** – offers a synchronous or asynchronous Web Service interface: The <receive> construct allows the business process to do a blocking wait for a matching message to arrive. The <reply> construct allows the business process to send a message in reply to a message that was received through a <receive> construct. The combination of a <receive> and a <reply> forms a request-response operation on the WSDL portType for the process.
- **throw** – generates a fault from inside the business process, which can be solved by debugging. If the debugging process does not occur, the bug receives the global scope and terminates therefore the process.
- **wait** – allows to wait for a given time period or until a certain time has passed.
- **empty** – inserts a "no-operation" instruction into a business process, for example during a debugging process. This is useful for synchronization of concurrent activities, for instance.
- **terminate** – Although <terminate> is permitted as an interpretation of the token activity, it is only available in executable processes. The terminate activity can be used to immediately terminate the behaviour of a business process instance within which the terminate activity is performed. All currently running activities MUST be terminated as soon as possible without any fault handling or compensation behaviour.

Structured Activities – These activities contain other activities and allow recursive compositions of complex processes. They prescribe the order in which a collection of activities takes place. They depict how a business process is created by composing the basic activities. They perform into structures that stand for the control patterns, data flow, handling of faults and external events and coordination of message exchanges between process instances involved in a business protocol.

BPEL4WS consists of following structured Activities:

- **sequence** – A sequence activity contains one or more activities that are performed sequentially, in the order in which they are listed within the <sequence> element, that is, in lexical order. The sequence activity completes when the final activity in the sequence has completed.
- **while** – The while activity supports repeated performance of a specified iterative activity. The iterative activity is performed until the given Boolean while condition no longer holds true.
- **switch** – conditional execution of activities: it allows to select exactly one branch of activity from a set of choices.
- **flow** – one or more activities can be performed concurrently. Links can be used within concurrent activities to define arbitrary control structures. The simplest use of flow is equivalent to a nested concurrency construct. Every link declared within a flow activity **MUST** have exactly one activity within the flow as its source and exactly one activity within the flow as its target. The source and target of a link **MAY** be nested arbitrarily deeply within the (structured) activities that are directly nested within the flow, except for the boundary-crossing restrictions.
- **pick** – Pick activities block and wait for a suitable message to arrive or for a time-out alarm to go off. When one of these triggers occurs, the associated activity is performed and the pick completes.
- **compensate** – The <compensate> construct is used to invoke compensation on an inner scope that has already completed normally. This construct can be invoked only from within a fault handler or another compensation handler.

Ordinary sequential control between activities is provided by sequence, switch and while.

The token “activity” can also be a scope: The behaviour context for each activity is provided by a scope. The scope construct defines a nested activity and integrate activities to a transactional unit with its own associated variables, fault handlers, event handler, compensation handler and correlation sets.

All scope elements are syntactically optional and some have default semantics when left out. Each scope has a primary activity that defines its normal behaviour. The primary activity might be a complex structured activity, with many nested activities within it to arbitrary depth. The scope is shared by all the nested activities.

Without considering links, the semantics of business processes, scopes and structured activities state when a given activity is ready to start.

There is no standard graphical notation for WS-BPEL, as the OASIS technical committee decided this was out of scope. The most popular notation for a direct visual representation of BPEL is the Business Process Modeling Notation (BPMN).

As an illustration of the feasibility of this approach, the BPMN specification includes an informal and partial mapping from BPMN to BPEL 1.1. However, it has exposed fundamental differences between BPMN and BPEL, which make it very difficult, and in some cases impossible, to generate human-readable BPEL code from BPMN models.

People often participate in the execution of business processes introducing new aspects, such as human interaction patterns. To support a broad range of scenarios that involve people within business processes, a BPEL extension is required, so called BPEL4People. It is defined in a way that it is layered on top of the BPEL language so that its features can be composed with the BPEL core features whenever needed. The BPEL process definition has the people activity as a new activity type that incorporates user interactions. People activities are implemented by tasks performed by users. Particular users who perform a task may be specified at design time, at deployment time, or at runtime.

The design of BPEL envisages extensibility so that systems builders can use other languages as well. BPELJ may enable Java to function as a 'programming in the small' language within BPEL. BPELJ allows these two programming languages to be used together to build complete business process applications.

Metamodel:

A process definition in BPEL consists of one activity, a series of partners and variables with specific correlation sets, the definition of fault handlers and compensation handlers.

In short, a BPEL4WS business process definition can be thought of as a template for generating business process instances. The creation of a process instance in BPEL4WS is always implicit; activities that receive messages

(that is, receive activities and pick activities) can be annotated to indicate that the occurrence of that activity results in a new instance of the business process to be created.

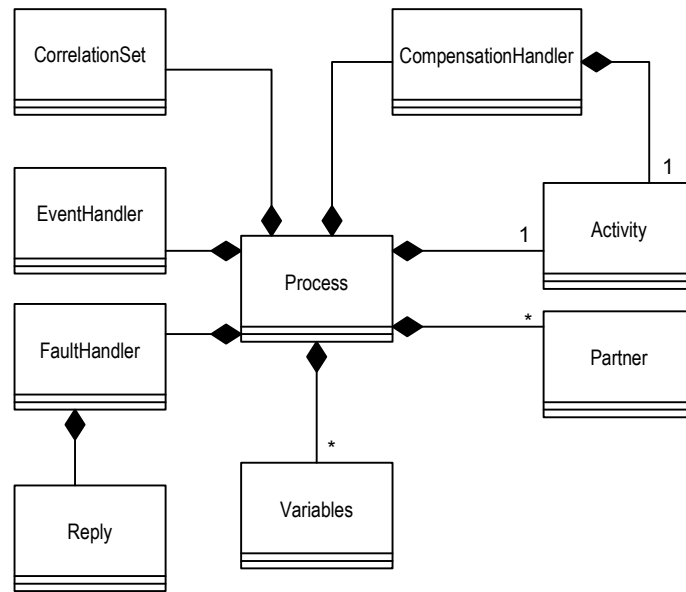


Figure 27: BPEL Metamodel

The Compensation Handler: Scopes can outline a part of the behaviour that is meant to be reversible in an application defined way by a compensation handler. A compensation handler is simply a wrapper for a compensation activity. It is recognized that in many scenarios the compensation handler has to receive data about the current state of the world and return data regarding the results of the compensation. The compensation handler allows long-lasting transactions.

An important requirement for realistic modelling cross-enterprise business interactions, in which the business processes of each enterprise interact through Web Service interfaces with the processes of other enterprises, is the ability to model the required relationship with a partner process. Partner links characterize the shape of a relationship with a partner by defining the message and port types used in the interactions in both directions.

A partner link type describes the conversational relationship between two services by defining the "roles" played by each of the services in the conversation and specifying the portType provided by each service to receive messages within the context of the conversation. Each role determines exactly one WSDL portType.

Variables provide the means for holding messages that describe the state of a business process. The messages held are often those that have been received from partners or are to be sent to partners. Variables can also hold data that are necessary for holding state related to the process and never exchanged with partners. Variables permit processes to maintain state data and process history based on messages exchanged.

So each variable is declared within a scope and is said to belong to that scope. Variables that are attached to the global process scope are called global variables. Variables may also belong to other, non-global scopes, and such variables are named local variables. Each variable is visible only in the scope in which it is determined and in all scopes nested within the scope it belongs to. Therefore, global variables are visible throughout the process.

Variables associated with message types can be described as input or output variables for invoke, receive and reply activities. When an invoke operation returns a fault message, this generates a fault in the current scope. The fault variable in the corresponding fault handler is initialized with the fault message received.

The variables specify the data variables used by the process, providing their definitions in terms of WSDL message types, XML Schema simple types, or XML Schema elements.

BPEL4WS addresses correlation scenarios by providing a declarative mechanism to define correlated groups of operations within a service instance. A set of correlation tokens is specified as a set of properties shared by all messages in the correlated group. Also correlation sets are declared within scopes and associated with them in a manner that is analogous to variable declarations.

If more than one start activity is enabled concurrently, then all such activities must apply at least one correlation set and must apply the same correlation sets. If exactly one start activity is anticipated to instantiate the process, the use of correlation sets is unconstrained.

The Fault Handler: Fault handling in a business process may be thought of as a mode switch from the normal processing in a scope. Fault handling in BPEL4WS is always considered as "reverse work" in that its sole aim is to undo the partial and unsuccessful work of a scope in which a fault has arisen. The attainment of the activity of

a fault handler, even when it does not rethrow the fault handled, is never considered successful completion of the attached scope and compensation is never enabled for a scope that has had an associated fault handler invoked.

Because of the flexibility allowed in expressing the faults that a catch activity may handle, it is possible for a fault to match more than one fault handler. The following rules are applied to select the catch activity that will process a fault:

1. If the fault has no associated fault data, a catch activity that specifies a matching faultName value will be selected if present. Otherwise, the default catchAll handler is selected if present.
2. If the fault has associated fault data, a catch activity specifying a matching faultName value and a faultVariable whose type (WSDL message type) matches the type of the fault's data will be selected if present. Otherwise, a catch activity with no specified faultName and with a faultVariable whose type matches the type of the fault data will be selected if present. Otherwise, the default catchAll handler is selected if present.

Because operations invoked can return a fault, a fault handler is provided. When a fault occurs, control is transferred to the fault handler, where a <reply> element is used to return a fault response of type "unableToHandleRequest" to the correspondent requester.

The Event handler: The whole process as well as each scope can be associated with a set of event handlers that are invoked concurrently if the corresponding event occurs. The actions taken within an event handler can be any type of activity, such as sequence or flow, but invocation of compensation handlers using the <compensate/> activity is not permitted. As stated earlier, the <compensate/> activity can only be used in fault and compensation handlers. There are two types of events. First, events can be incoming messages that correspond to a request/response or one-way operation in WSDL. For instance, a status query is likely to be a request/response operation, whereas a cancellation may be a one-way operation. Second, events can be alarms that go off after user-set times.

It is important to stress out that event handlers are considered as part of the normal behaviour of the scope, unlike fault and compensation handlers.

Modelling Example:

A simple process scenario of an application for a travel agency shall show how BPEL works. The scenario consists of following steps: to accept an itinerary from a customer, purchase the tickets from the airline and hand deliver them to the customer.

Figure 28 represents a flow diagram of the activities in this business process. A travel agent particularizes a business process called ticketOrder (line 1). The objective of this simplistic business process is to allow the agent to receive from a customer an itinerary (lines 20 to 23), to pass on this itinerary to an airline requesting the corresponding tickets (lines 26 to 29), and to receive these tickets from the airline (lines 33 to 36) in the end. To keep the example simple, it is assumed that the tickets will be picked up by the customer in person.

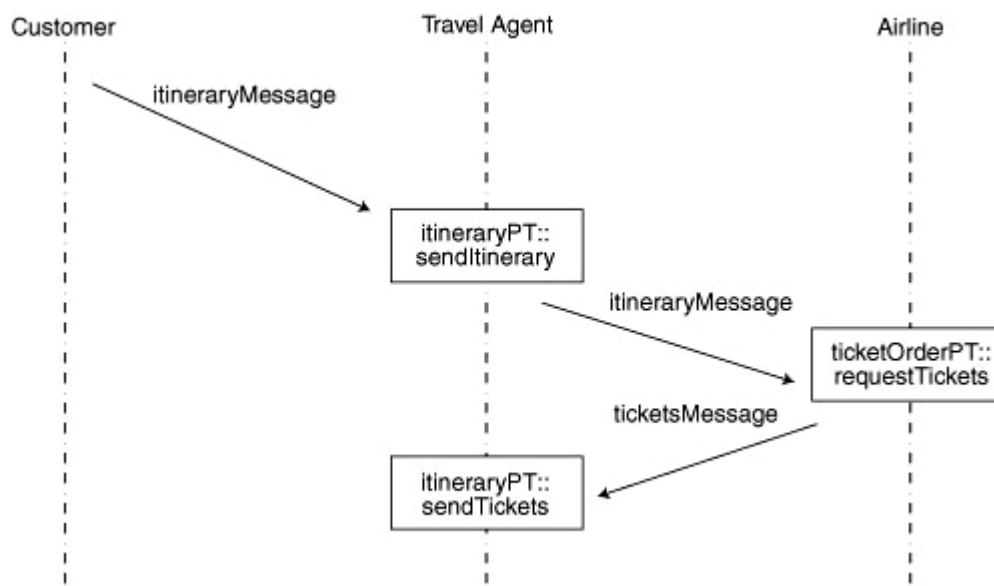


Figure 28: BPEL modelling example

The set of partners the agent's process interacts with are described in lines 2 to 10: lines 3 to 5 establish the partner customer, and lines 6 to 9 introduce the partner airline. A partner definition involves specifying the Web

services mutually applied by the partner or process, respectively (see the next section, "Partners", for more details).

The messages that are persisted by the process are called variables (lines 11 to 14). Variables are WSDL messages that are received from or sent to partners. For example, the process stores an itineraryMessage as an itinerary variable. The itineraryMessage is received from the customer (line 20) when the customer utilizes the sendItinerary operation of the agent's itinerary port (lines 21 and 22). This message is stored into the itinerary variable (line 23) once received. The process then passes on the itinerary message to the airline (line 26) by using the requestTicket operation of the ticketOrder port (lines 27 and 28); whereas this message is a copy of the itinerary variable (line 29).

An activity is the usage of an operation in a business process. To specify the order in which the activities have to be performed, the ticketOrder process structures its activities as a flow (line 15). The links required to define the flow between the ticketOrder process's activities are described in lines 16 to 19. And an activity defines the links that it is a source or target of. For example, the receive activity of line 20 is the source of the order-to-airline link (line 24). And this link has the invoke activity of line 26 as target (line 30).

```

1 <process name="ticketOrder">
2   <partners>
3     <partner name="customer"
4       serviceLinkType="agentLink"
5       myRole="agentService"/>
6     <partner name="airline"
7       serviceLinkType="buyerLink"
8       myRole="ticketRequester"
9       partnerRole="ticketService"/>
10  </partners>
11  <variables>
12    <variable name="itinerary" messageType="itineraryMessage"/>
13    <variable name="tickets" messageType="ticketsMessage"/>
14  </variables>
15  <flow>
16    <links>
17      <link name="order-to-airline"/>
18      <link name="airline-to-agent"/>
19    </links>
20    <receive partner="customer"
21      portType="itineraryPT"
22      operation="sendItinerary"
23      variable="itinerary"
24      <source linkName="order-to-airline"/>
25    </receive>
26    <invoke partner="airline"
27      portType="ticketOrderPT"
28      operation="requestTickets"
29      inputVariable="itinerary"
30      <target linkName="order-to-airline"/>
31      <source linkName="airline-to-agent"/>
32    </invoke>
33    <receive partner="airline"
34      portType="itineraryPT"
35      operation="sendTickets"
36      variable="tickets"
37      <target linkName="airline-to-agent"/>
38    </receive>
39  </flow>
40 </process>

```

Source: <http://www-128.ibm.com/developerworks/library/ws-bpelwp/>

Part of Framework:

BPEL is not part of a framework

Supporting software tools:

WebSphere® Studio Application Developer Integration Edition V5.1, Oracle BPEL Designer (Stylus Studio® 2007 XML)

Evaluation:

BPEL is designed for the execution of business processes. It is an XML language and has no own graphical representation, therefore requirement 6 is not fulfilled. Similar requirement 7, which claims a tool support is only

mostly not fulfilled, as other languages have to be used in order to create BPEL models in a graphical way. BPEL does not allow the description of Business rules or Business Goals (requirement 2). It doesn't use pure business terms and is not designed directly for the use of business people who are not familiar with block-oriented languages like XML. BPEL is detailed enough for the CIM level therefore it fulfils requirement 5. In addition BPEL can be exported and imported by several tools, therefore it is tool interoperable. BPEL has a sound definition, which fulfils requirement 10. It can be used to describe workflows (requirement 11). However, it should only be partly used in a MDA-approach on the CIM, because it is very technical for typical CIM users. BPEL does not integrate different modelling views. It only focuses on the process view. The evaluation of BPEL is summed up in the following table:

Table 9: Evaluation of BPEL

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	✓	8	✓
2	✗	9	✓
3	✗	10	✓
4	✗	11	✓
5	✓	12	(✓)
6	✗	13	✗
7	(✗)		

2.3.2.8 XML Process Definition Language

Acronym:

XPDL

Specification document(s):

Process Definition Interface – XML Process Definition Language, WPMC-TC-1025 (TC-1025_xpdl_2_2005-10-03.pdf)

Specifying organisation:

Workflow Management Coalition

Evaluated Version(s):

Version 2.00, October 3, 2005

Description of the language:

XPDL is a language to define business processes. Starting from version 2.0 XPDL also supports Business Process Modelling Notation and is able to represent the model of a business process.

In XPDL it is possible to define graph-structured processes. The core entities of the language are process definition, activities, workflow participant assignment and transition. In contrast to WS-BPEL, this language is able to represent human workers and assign them to activities. In addition, work assignment rules may be more advanced and include such aspects as role, organisational structure and groups.

The language may be considered as an XML representation of the process definition meta-model included in [XPDL05] and is presented in Figure 29. The structure of the language follows the rules expressed in the package definition meta-model included in [XPDL05].

Metamodel:

XPDL uses the process definition meta-model presented in [XPDL05].

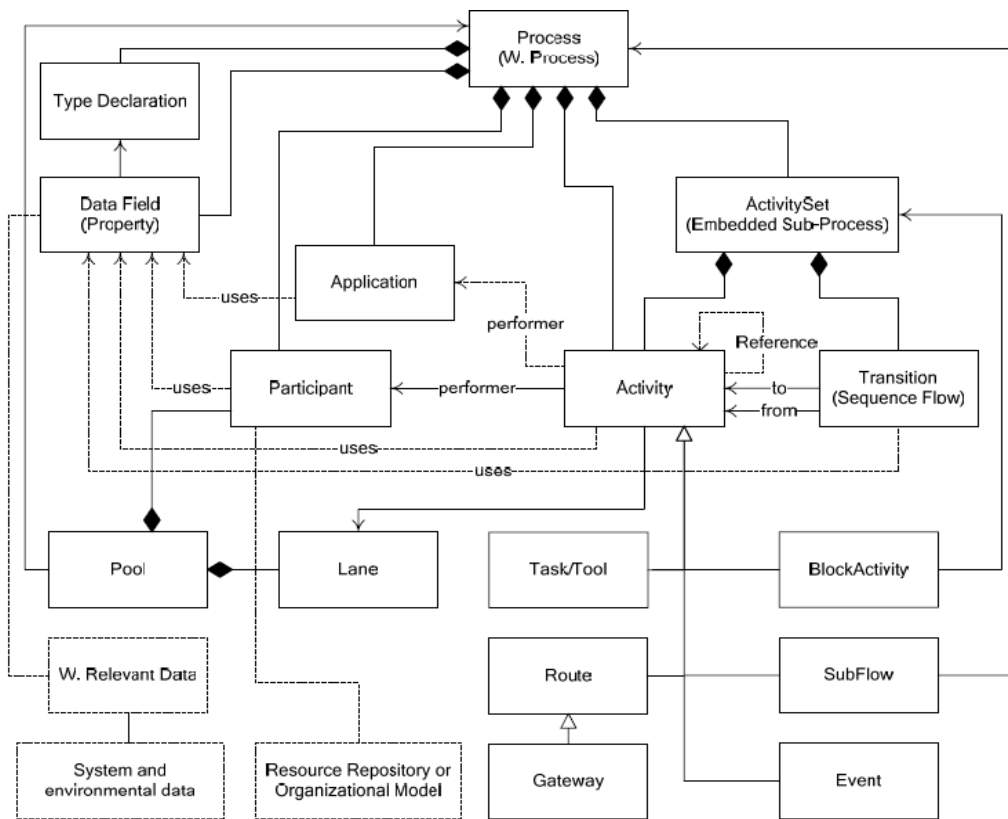


Figure 29: WfMC process definition meta-model

A process definition consists of activities that are organised in a graph-structure via transitions. An activity may be executed by a performer. There is a set of activity types. An activity may be a route activity, a tool (or application), an event or a composed activity that is represented as another process. Every process has also a data container that includes attributes (or data fields) required to execute this process. The activities performed by one person, role, or organisation are organised in lanes (first level) and pools (second, higher level).

Modelling Example:

A detailed example of a XPDL application is included in [XPDL05], section 8.

Part of Framework:

XPDL is a core part of WfMC Standards Framework that includes about 15 standards on various aspects of business process management.

Supporting software tools:

There are at least 40 tools that support XPDL. These tools include both open source and commercial products. The detailed list with the references to concrete implementations is available at <http://www.wfmc.org/standards/xpdl.htm>.

Evaluation:

XPDL may be used to define dynamics of the modelled (and designed) system in terms of business processes (or workflow processes). Since XPDL is a process definition language, the defined processes may be easily executed in an XPDL compliant workflow engine without any additional effort. XPDL may also be useful if the modelled business processes include people as performers of activities. Examples of such processes are administrative processes.

XPDL has also many implementations for both design tools and execution engines. The tools are both open source and commercial ones. Therefore XPDL is able to describe complex business processes (requirement 1). But XPDL does not consider Business Rules (requirement 2). It is an XML-based language and has no graphical representation. Hence it is not intuitive and uses business terms (requirements 3 and 4). But XPDL is a defined standard which fulfils requirement 5. As it has no graphical notation it can only be modelled indirectly using other modelling languages like BPMN. For this reason requirement 6 is not and requirement 7 mostly not fulfilled. XPDL can serve as an interchange format between workflow modelling tools and engine, there it is

highly interoperable (requirement 8) and able to describe workflows (requirement 11). It is soundly defined and can be used for complex scenarios. But XPDL does not support different views. The evaluation of XPDL is summed up in the following table:

Table 10: Evaluation of XPDL

CIM REQ	Fulfilled?	CIM REQ	Fulfilled?
1	✓	8	✓
2	✗	9	✓
3	✗	10	✓
4	✗	11	✓
5	✓	12	(✓)
6	✗	13	✗
7	(✗)		

2.4 Summary of the language evaluation

The EPC supports the description of business processes very well. Additionally it is rather easy to understand and possesses a graphical notation. But it does not cover workflows and has no official standard. Therefore the EPC, won't be the base for the VIDE CIM Level Language. But the concept of the integration of different views will be used in the VCLL.

BPMN has mostly the same advantages as the EPC and additionally it is standardised and covers workflow. Therefore BPMN will serve as the base for the VCLL. With regard to the requirement of tool interoperability and the option to export workflow description, XPDL is a known standard. XPDL does not have a graphical notation. But XPDL 2.0 can be displayed with BPMN 1.1. Therefore the workflow branch of the CIM level architecture consists of BPMN as the notation and XPDL as the exchange format.

But this does not cover all requirements for the CIM to PIM transformation. Additional information is needed. None of the evaluated languages contain detailed information about data structures and organisational structures. Business Rules are not regarded in any language either. Therefore the designed language will be based on the workflow part (with BPMN as the graphical notation and with XPDL). For the transformation from CIMs to PIMs other information such as data structures, organisational structures and business rules are added. This allows the use of the core of the language covered by BPMN for the orchestration of workflows and enriching this model with other views in order to create VIDE PIM models. While XPDL is a soundly defined standard BPMN is only specified by a textual description and modelling examples. Therefore, in order to use frameworks and standards such as MOF, Ecore, GMF and GEF the BPMN part of the graphical VIDE CIM level language has to be defined in a UML metamodel and be refined by OCL statements. This will allow an XPDL export for a part of the model. The whole model, containing all views, will also be stored in XML. This XML format can be generated automatically using the framework GMF. The specification of the VCLL is done in the next chapter.

3. Design of the VIDE CIM level language

3.1 Specification of the VIDE CIM level language

3.1.1 Metamodel of the VIDE CIM level language

The aim of this section is to describe the VIDE CIM metamodel. The purpose of the metamodel is to define the necessary constructs for the graphical VIDE CIM level language. Additionally it provides the data classes that are needed to store VIDE CIM models.

There are three views of VIDE CIM model introduced in this document: process view, data view and organisational view. The most extensively represented one is the process view, which integrates the other views. It consists of eight parts, that describe particular elements of the metamodel. *Structuring objects* are the top-level classes that the model contains. The *control flow* section introduces lane elements, which are connected to each other by flow objects. The *connections* section has two further types of connections between objects in a model. The *annotated elements* part of the metamodel shows model objects that are used to enrich activity objects with relevant information. The *activity* section tells which elements are representing actions in a model. The *events* section describes different types of events which describe what kind of triggers could be used in a model. *Gateways* explain how decisions of different types could be integrated into a VIDE CIM model. The *enumerations* section is the last section and it gives an overview of the complex types used in three different classes. The data view and the organisational view introduce interfaces to two further business process analysis scopes and are described shortly in their own sections.

3.1.1.1 Process view

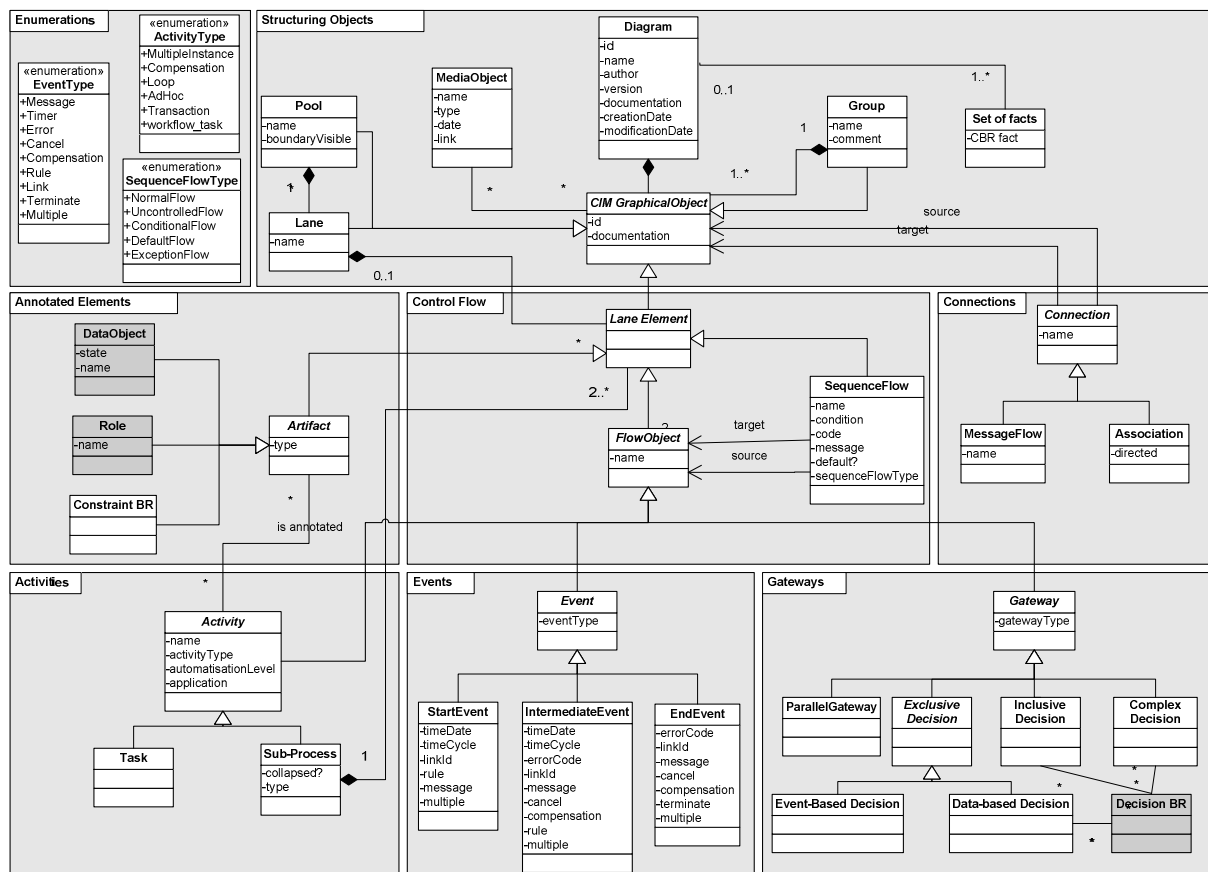


Figure 30: Process view metamodel

3.1.1.1.1 Structuring objects

This section describes the main container elements of a model.

Diagram – every VIDE CIM model is represented through a diagram object. A diagram is a main top-level object container in a VIDE CIM model. It can contain graphical objects, which in turn do not exist without a relation to a model. Therefore there cannot be a VIDE CIM level model without this object.

CIM Graphical Object – is an abstract representation of each figure in a diagram. The visible elements in the diagram are all inherited from this abstract class. Each graphical object belongs to a diagram.

MediaObject – is an unstructured data that could be attached to every element in the diagram. MediaObjects could be hand-written texts, recorded audio data of interviews or videos enriching the background knowledge of the model element they are annotated to.

Group – with this object one can build completely different groups of elements in a diagram. It could be used to stress a collection of objects relevant to a user, for example actions to do or events to react at. It is not considered a subclass of *Artifact* as it is done in OMG BPMN Specification, because it does not necessarily connect flow and non-flow objects with each other but rather grouping subclass instances of *CIM GraphicalObject*.³

Pool – is a container of processes. Each and every simple process represented through the object sequence may be described in one pool. However when the process logic grows a process may be seen stretching its borders over two or more pools. Nevertheless a sequence is always contained within one pool and may not cross its limits. The communication between complex processes having two or more pools takes place through message connections. An example for a pool would be a logistics enterprise in which the business process takes place.

Lane – is a compartment inside a pool. It represents different organisational units taking actions or reacting to events. If there is one lane in a pool then it is considered the same as the pool and isn't shown graphically. Even simple processes can cross lane borders while staying inside the same pool. In a real world a lane could represent an accounting department inside an enterprise, which is responsible for charging customers for the enterprise services.

Set of facts – A set of facts consists of one or more facts that are the base for constraint business rules. If no constraint business rules are used this model element can be omitted. Facts are valid for the whole diagram and all constraint business rules that are used in this diagram. Constraint business rules and facts are further explained in the business rule view.

3.1.1.1.2 Control Flow

This section describes the main elements that guide the sequence of actions in a business process.

Lane Element – this class is an abstract representation of every element that is situated within a lane or a pool. Each subclass instance of *Lane Element* can reside in one or no lane. If there is only one lane in a pool, then all the lane elements belong to a dummy lane. This kind of a lane isn't shown but belongs to the pool. Some lane elements could contain other lane elements. To avoid the unneeded empty nesting of this kind of elements the cardinality with respect to Sub-Process is set to at least 2. It means if lane elements are nested in another lane element, there should be at least two of them (e.g. start and end events).

FlowObject – this abstract class stays for elements that reflect the order in which activities in a process are being carried out. All flow objects inside one lane or a pool are connected with *SequenceFlow* instances. This explains the order in which the activities are to be taken. There are three subclasses of the *FlowObject* addressed more closely in later sections: *Activity*, *Event* and *Gateway*.

SequenceFlow – this class is one of the main factors that control the flow. It connects two *FlowObject* subclass instances within one pool, one source instance and one target instance, into logical sequences of actions that have to be executed in some particular order. Without it there would be just a set of actions and nobody would be able to say which is to do first. *SequenceFlow* instances are lane elements, too. Therefore they don't cross the borders of a pool, at most they could cross the border of a lane in case there is more than one lane in the pool.⁴ An example for a sequence flow would be a certain order prescription: deposit the goods at the warehouse first and then check an invoice.

³ Compare *OMG: Business Process Modelling Notation Specification*, pp. 95-97.

⁴ Compare *OMG: Business Process Modelling Notation Specification*, p. 27.

3.1.1.1.3 Connections

This section gives an explanation of three further types of connections in the metamodel.

Connection – an abstract class that introduces the relation between two subclass instances of *CIM GraphicalObject*. It refers to one source instance and one target instance. Therefore a direction of the connection can be stored.

MessageFlow – is a class that serves the stressing of a relation between two subclass instances of *CIM GraphicalObject* that don't belong to the same participant lane boundary.⁵ It also belongs to elements that control the sequence of activity execution. Two processes situated in different pools can “communicate” with each other through MessageFlow, thus letting other participants know of important activity executions happening within a process. After receiving an order on goods, for example, there is a message to the warehouse issued saying the collection process should begin.

Association – is a class that represents a relation between two subclass instances of *CIM GraphicalObject*. It is intended to describe a connection between flow objects and non-flow objects. In particular, flow objects could be associated with text and other non-flow objects.⁶ An inventory control department is responsible for inventory taking and thus is associated with it.

3.1.1.1.4 Annotated Elements

This section describes the purpose of activity annotation in the metamodel.

Artifact – is an abstract class representing the non-flow graphical elements in a VIDE CIM model that could be related to flow components. An instance of a subclass of *Artifact* could be related to more than one *Activity* and, other way round, one *Activity* could be related to more than one *Artifact*.

DataObject – is an interface to a data view. This view is specified in an appropriate section after process view.

Role – is an interface to an organisational view. The substantial components of this view are further described in a section following the data view section.

ConstraintBR – this class represents a functional category⁷ of Business Rules that restricts the structure or properties of actions. This category could be seen as an assertion or invariant which provides additional information about the desired behaviour of the application, which should be developed. Instances of this class on the CIM level can be considered as system requirements on the PIM level. Constraint business rules are defined in the business rule view.

3.1.1.1.5 Activities

This section describes the actions within a process in a VIDE CIM model.

Activity – this abstract class stays for work carried out in a business process. It could be simple action or a compound one, as it can be seen in its subclasses Task and Sub-Process. It can also be further annotated with artifacts to show relevant information that doesn't directly belong to the business process itself.

Task – is a representation of an atomic indivisible activity. A Task should be executed completely or not at all. It could be seen as work in the process that is not analyzed further into finer details, like taking an order or issuing an invoice.⁸ A task cannot be subdivided into other activities.

Sub-Process – a class describing a compound activity that could be, depending on the goal of the examination, seen as a whole or as a sequence of activities. Instances of Sub-Process could contain other subclass instances of *LaneElement*. There is no use to put only one element inside the Sub-Process, it would then mean this element is the same as the Sub-Process. In order to avoid this, the cardinality at *LaneElement* is set to at least 2. It preserves Sub-Process instances from containing only one element. Take the good delivery for example – there are many

⁵ Compare *OMG: Business Process Modelling Notation Specification*, p.28.

⁶ Compare *OMG: Business Process Modelling Notation Specification*, p.105.

⁷ Compare C. Seel, B. Simon, D. Werth: Business Rule-enabled Process Modelling in: Proceedings of the e-Challenges 2006 conference, Barcelona, Spain, October 25-27, 2006

⁸ Compare *OMG: Business Process Modelling Notation Specification*, p.62.

tasks for a warehouse to accomplish. First receive the goods after they arrived, then accept an invoice, deposit goods in the stock, issue a confirmation receipt and so on into the fine details.

Workflow Task – a workflow task can be used in the control flow and represents a VIDE application which is orchestrated. Therefore an application can be assigned to the workflow task.

3.1.1.1.6 Events

This section introduces events managing execution of activities in a VIDE CIM model.

Event – is an abstract class representing a further business process control feature. Events define what occurs during process execution, though only those incidents are accounted for events that have to do with the sequence or timing of activities in a business process.⁹ There are three main types of *Event* that are shown in the subclasses: start, intermediate and end events. An overview of further event subcategories can be seen in Table 1.

StartEvent – is a type of *Event* that triggers the execution of a process. There could be more than one start event in a process, although the use of this modelling construct is not required in every case for a given business process level.¹⁰ For a description of the feasible categories of the *StartEvent* see Table 2. A certain time in a week could be a starting event for an inventory taking.

IntermediateEvent – is a type of event that occurs during the process execution after a *StartEvent* and before an *EndEvent*. It should not start any process or directly terminate a running one.¹¹ For a description of the feasible categories of the *IntermediateEvent* see Table 3. A possible value for an intermediate event is an error, for example a customer couldn't be debited for delivery because account volume was insufficient at the time.

EndEvent – is a type of event that ends the business process at the certain level and eventually produces a result from the business process execution. This result can be transferred for further treatment onto subsequent processes. There could be more than one end event in a process, although the use of this event is not required in every case for a given process level.¹² For a description of the feasible categories of the *EndEvent* see Table 4. The termination of the business process is an end event causing all of the activities to stop, like the deadline that has been reached.

3.1.1.1.7 Gateways

In this section branches and joins that are represented through gateways are described.

Gateway – is an abstract class representing a further business process control feature. It is used after an action within a process if decisions have to be made what further actions have to be executed. In addition to different kinds of gateways introduced below there are Decision Business Rules (Decision BR) that describe decision procedures to be carried out in order to execute the business process further.

ParallelGateway – this type of gateway has an “AND” semantic. After an “AND” split all of following business process paths next to it will be unconditionally executed. For example after getting informed about a delivery an invoice should be sent and a customer should be debited. The order of these actions is not relevant, they could also happen at the same time, in parallel.

ExclusiveDecision – is an abstract gateway type that describes the “XOR” logic. It means the decision behind this gateway takes one and only one of the paths available next to it and executes this one path. In this kind two types of decision bases are possible: data-based decision and event-based decision.

Data-based Decision – this type of decision uses process data to determine which business process path is to take. The type of expression evaluated depends on the modeller's needs. If no path corresponds to the result of the evaluated expression, then the modeller should concern the use of a default gateway, otherwise a model is

⁹ Compare *OMG: Business Process Modelling Notation Specification*, p.34.

¹⁰ Compare *OMG: Business Process Modelling Notation Specification*, pp.35-39.

¹¹ Compare *OMG: Business Process Modelling Notation Specification*, pp.43-49.

¹² Compare *OMG: Business Process Modelling Notation Specification*, pp.40-43.

incorrect.¹³ An example for this kind of decision is a decision based on delivery volume, this means, after certain volume a discount should be granted.

Event-based Decision – this kind of decision depends on the event occurring at the time of decision making. It doesn't use process data to choose the path but takes actions according to an event that took place.¹⁴ One example is the time event: if the delivery didn't happen up to a certain time, then the price wouldn't be as high as supposed before.

Inclusive Decision – is a type of gateway that supposes the “OR” logical decision. It means after this gateway there is one of the process paths being executed, all of them or every possible combination of them. However, at least one of the paths has to be taken.¹⁵ For example, if a customer states a preference concerning the type of delivery, it could be delivered all at once or as soon as possible item by item.

Complex Decision – is a decision that is used for cases that cannot be modelled by other gateway types. It also can be used to represent a structure of more simple decisions in a compact way.¹⁶ So this type of gateway is used, if there is more than one expression or event that should be considered. For example if both the volume of the delivery and its time are relevant in order to decide what business process paths will be executed.

Decision BR – is a kind of a Business Rule that helps making decisions during process execution through differentiating between possible situations. In contrast to the Constraint BR the Decision BR supports execution of business processes according to business logic expressed by a modeller.

3.1.1.1.8 Enumerations

This section describes three enumeration types in the metamodel.

ActivityType is an auxiliary subcategory of an activity that helps describing its nature. There are the following values of this type available:

- MultipleInstance – this subcategory is assigned in case it is possible that more than one instance of the activity is being executed at a time. The starting time of all activity instances could be the one and the same.¹⁷
- Compensation – this subcategory is assigned to the activities following those that produce complex effects or specific outputs. If the outcome is determined to be undesirable by some specified criteria (such as an order being cancelled), then it will be necessary to “undo” the activities. An activity that might require compensation could be, for example, one that charges a buyer for some service and debits a credit card to do so.¹⁸
- Loop – this subcategory is assigned if the same activity is charged to repeat more than one time in a process. In this case each activity instance starts when the previous is completed.¹⁹
- AdHoc – an Ad Hoc process is a group of activities that have no pre-definable sequence relationships. A set of activities can be defined for the process, but the sequence and number of performances for the activities is completely determined by the performers of the activities and cannot be defined before.²⁰

EventType is a list of possible event subcategories. However, not every event category is compatible with each event subcategory. An overview on feasible combinations of event types is given in a figure 31.

¹³ Compare *OMG: Business Process Modelling Notation Specification*, p.72.

¹⁴ Compare *OMG: Business Process Modelling Notation Specification*, pp.75-78.

¹⁵ Compare *OMG: Business Process Modelling Notation Specification*, pp.78-82.

¹⁶ Compare *OMG: Business Process Modelling Notation Specification*, pp.82-85.

¹⁷ Compare *OMG: Business Process Modelling Notation Specification*, p.121.

¹⁸ *OMG: Business Process Modelling Notation Specification*, p.133.

¹⁹ Compare *OMG: Business Process Modelling Notation Specification*, p.121.

²⁰ *OMG: Business Process Modelling Notation Specification*, p.132.

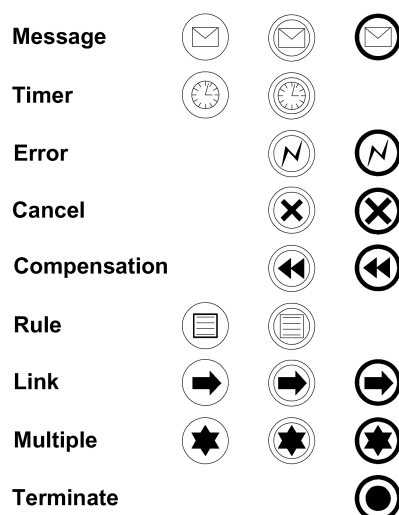


Figure 31 Combinations of Event Categories and Subcategories

SequenceFlowType is a subcategory of this type of connection. There exists the following types of sequence flow:

- NormalFlow – refers to the flow that originates from a StartEvent and continues through activities via alternative and parallel paths until it ends at an EndEvent,²¹
- UncontrolledFlow – refers to the flow that is not affected by any conditions or does not pass through a Gateway. The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple SequenceFlow that converge on or diverge from an activity,²²
- ConditionalFlow – sequence flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be used,²³
- DefaultFlow – for Data-Based Exclusive Decisions or Inclusive Decisions, one type of flow is the Default condition flow. This flow will be used only if all the other outgoing conditional flows are not true at runtime,²⁴
- ExceptionFlow – occurs outside the normal flow of the process and is based upon an IntermediateEvent that occurs during the performance of the process.²⁵

3.1.1.2 Data view

Data view is a common model element of both the data and the process view. The expressiveness of the Data view models is similar to ERM.

²¹ *OMG: Business Process Modelling Notation Specification, p.20.*

²² *OMG: Business Process Modelling Notation Specification, p.20.*

²³ *OMG: Business Process Modelling Notation Specification, p.21.*

²⁴ *OMG: Business Process Modelling Notation Specification, p.21.*

²⁵ *OMG: Business Process Modelling Notation Specification, p.21.*

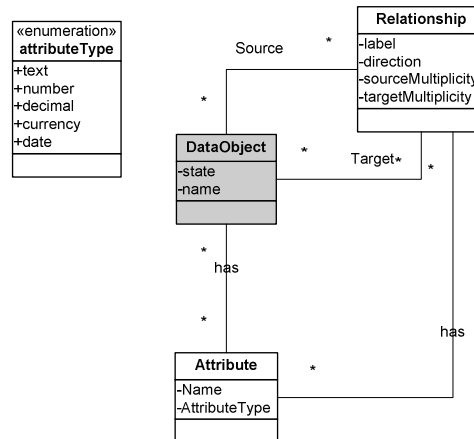


Figure 32: Data view metamodel

DataObject – according to the metamodel of the data view instances of DataObject can be associated with each other through the Relationship class instance and have additional attributes represented through Attribute instances. For example a list of workers in the warehouse with their data is used to choose the appropriate one to deliver goods to customers.

Relationship - leads the connection from one DataObject to another, thus logically structuring the data representation of a VIDE CIM model. Each Relationship instance can also have additional attributes represented through Attribute instances describing its individual properties. This is represented through the connection between these two classes. Following the example with the list of workers a Relationship class could represent a relation between workers and ranking classes describing workers' skills.

Attribute – essentially represents a property of a DataObject instance or of the Relationship. Whether a specific attribute is an instance containing information about a DataObject or a Relationship, could be seen on the connection to the according classes. The simplest example of an Attribute class instance is the name of each worker written in the DataObject containing the list of the available workers.

AttributeType – The type of an attribute can be text, number, decimal, currency and date. These types are used in order to use terms which are familiar to business people and programming language independent.

3.1.1.3 Organisational view

Organisational view describes the organisational structure that is related to an underlying activity. The core element of the organisational view is the role.

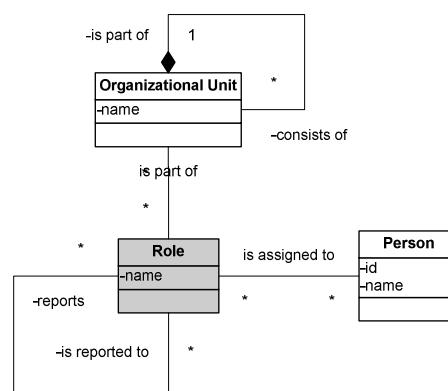


Figure 33: Organisational view metamodel

Role - is a model element that derives from the organisational view and creates a link between the organisational and the process view. Each role is assigned to one or more people, whilst one person can carry out more than one

role. At the same time each role can be part of an organisational unit. Roles can be hierarchically composed. It means for roles that there is a reporting structure between different roles. An accounting clerk debiting the customer after delivery of goods could appear as role.

Organisational unit – organisational units can have a hierarchical structure. For organisational units, the hierarchy represents the department structure of an enterprise, which allows to subdivide/aggregate business units. An accounting department could be an example for an organisational unit consisting of the role of accounting clerk.

Person – is a class representing human or other kind of agents responsible for carrying out the work associated with the organisational unit he or she has been assigned. A clerk in an accounting department could be a simple example of the class Person which is in this case assigned to a role accounting clerk.

3.1.1.4 Business Rule view

3.1.1.4.1 Decision Business Rules

There are two types of business rules that can be used in the VIDE CIM level languages: decision and constraint business rules. Decision business rules are used to describe complex branches of the control flow, e. g. if the decision which activity is the next to execute depends on the combination of several subdecisions.

Therefore decision business rules consist of one or more statements. Each statement consists of business variable values and one activity. If the business variables have the values which are described in the statement the activity, which is referred to in the statement, is executed. One decision business rules can consist of one or more statements. The statements have an OR relation between each other. Optional a last row can be added, which contains the keyword “else” as condition. The action which is assigned to this row is executed if no other condition is true.

Decisions business rules are defined as follows:

P be a business process.

Activity \in P.

Notation for decision business rules in EBNF:

$G = \{S, T, N, P\}$

$S := \{S\}$, $T := \{\text{or, and, a..z, A..Z, 0..9, else}\}$, $N := \{S, OP, CONDITION, ACTIVITY, VALUE, BUSINESSVARIABLE\}$

$P := \{$

$S \rightarrow S \text{ or } S$

$S \rightarrow \text{CONDITION then ACTIVITY}$

$\text{CONDITION} \rightarrow \text{BUSINESSVARIABLE} = \text{VALUE}$

$\text{CONDITION} \rightarrow \text{BUSINESSVARIABLE} = \text{VALUE and } S$

$S \rightarrow \text{else then ACTIVITY}$

$\text{VALUE} \rightarrow \{a..z | A..Z | 0..9\}^+$

$\text{BUSINESSVARIABLE}^{26} \rightarrow \{a..z | A..Z | 0..9\}^+$

$\text{ACTIVITY} \rightarrow \text{"An activity, that is part of the same model the business rule is used in."}$

$\}$

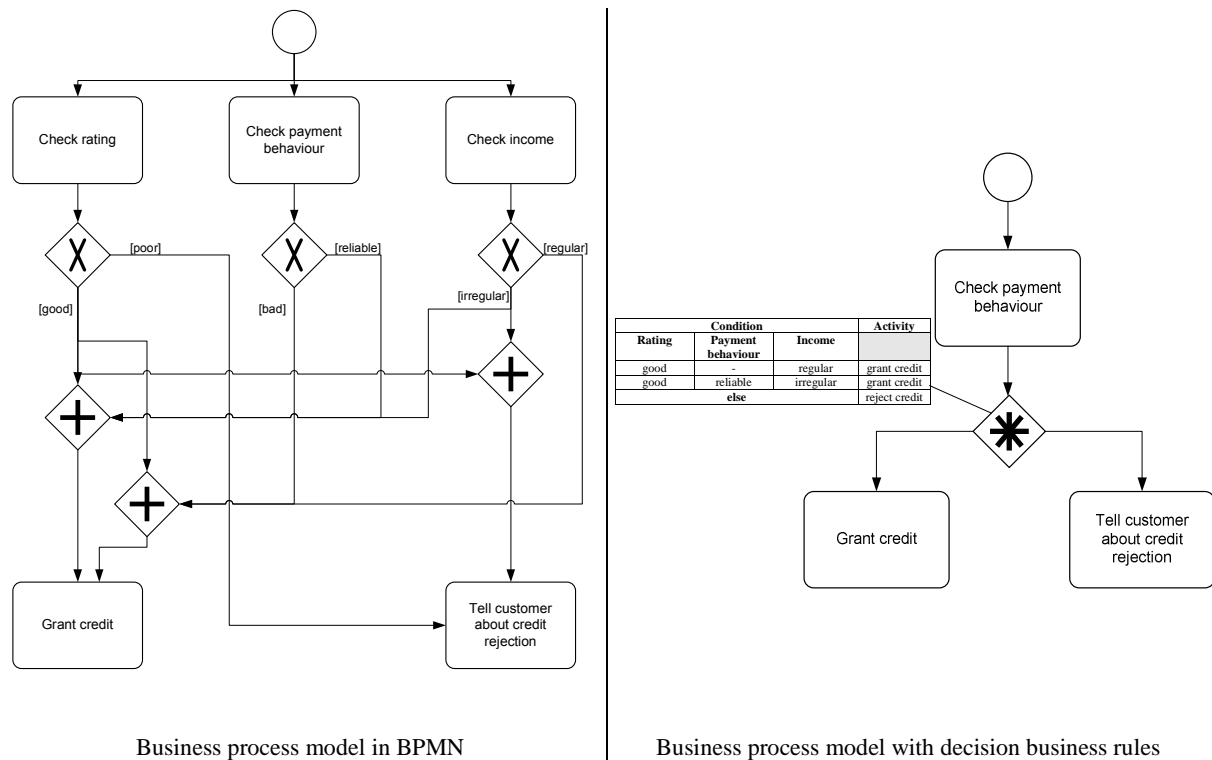
The usage of decision business rules is depicted in the following example:

The business process models describe both the credit appraisal of customer. The credit is approved if

- the customer's credit rating is good and his income is regular or
- if his rating is good and his payment behaviour is reliable and his income is irregular.

The example shows the decision business rules can reduce the complexity in this case.

²⁶ The business variable represents a fact of economic relevance, e.g. cost, cycle time, customer category, etc.

**Figure 34: Example for decision business rules**

3.1.1.4.2 Constraint Business Rules

The second type of business rules that are supported by the VCLL are constraint business rules. They can be annotated to any model element on the CIM level and state constraints from a business point of view. For example defining that an order process can be started by a phone call is not possible for new customers. Constraint business rules are constructs which are similar to natural language in order to make them easily accessible for business users. However, to avoid the ambiguity of natural language the parts of constraint business rules are further defined. For this purpose, the use of natural language has to be restricted to the use of standardised statements [Endl04]. In addition to established approaches like RDF [W3C04a] or OWL [W3C04b], which both focus on semantic-web-technologies, the approach “Semantics of Business Vocabulary and Business Rules Specification” (SBVR), which is defined by the OMG, proves to be a well developed concept for describing business rules in an enterprise-context.

The people addressed by the SBVR-specification are mainly users from the business domain, who should be enabled to formulate rules in a structured but also easy comprehensible manner. There is also a focus on the necessary transformation of the formulated rules into IT-systems. The SBVR defines specifications for the used vocabulary as well as syntactical rules, to allow a structured documentation of business vocabularies, business facts and business rules. Furthermore, the specification describes a XMI-scheme to share business vocabularies and business rules between organisations and IT-systems. The SBVR is designed to be interpretable in predicate logic with a small extension in modal logic. It also defines demands towards the behaviour of IT-systems regarding their ability to share vocabularies and rules that complies with the specification [OMG05].

The SBVR-approach uses three perspectives on business rules. The first perspective is derived from the business rules mantra [BRG06] and supports a simplified approximation towards a business rule. This perspective should support the communication with people who are not familiar with the approach, e.g. decision makers. The second perspective is the representation. It contains the specifications of SBVR which should be used to formulate vocabularies and rules. The third perspective is the meaning. It contains the underlying semantics of the used vocabularies and rules.

The SBVR expresses definitions and rules mainly in the terms of a restricted and structured natural English language. Therefore it is easy to understand them and they can be checked by domain users for relevance and correctness.

For the structuring and formalisation of the language, the SBVR specifies parts of sentences and its presentation. As basic elements for the expression of business rules the elements “term“, “name“, “verb“ and “keyword“ are defined. For example a “term” is defined as a noun which is part of the used business vocabulary. In contrary, the element “name“ is an individual noun, often proper nouns and could be a specification of „terms“. Each element is formatted uniquely (term, Name, verb, keyword).

A specific characteristic are the keywords, which are declared explicitly by the specification and mostly express logical facts.

The SBVR specification differentiates between *quantifications* (e.g. „each“, „some“, „at least one“, „at least n“), *logical operations* (e.g. “and“, “or“, “or...but not both“, “if...then“), *modal operations* (e.g. “it is obligatory that“, “it is impossible that“, “...must...“, “...must not...“, “...never...“, “...may...“) and other *keywords* (e.g. “the“, “a, an“, “another“) [OMG05].

Examples for the use of the specification are:

It is necessary that each rental has exactly one requested car group.

It is obligatory that the duration of each rental is at most 90 days.

or:

If the drop-off location of a rental is not the EU-Rent site of the return branch of the rental then it is obligatory that the rental incurs a location penalty charge.

Corresponding to the earlier described perspectives, these rules are settled in the perspective of representation. Furthermore, for being able to use these rules, it is necessary to formulate explicitly assumptions, which derive from the rule. This results in a list of supporting facts. Supporting facts define the relation between different terms and names by the use of verbs. These facts are introduced in order to avoid misunderstandings or the use of synonyms and homonym.

The facts for the third business rule of the example above could be:

rental has drop-off location

rental has return branch

branch is located at EU-Rent site

rental incurs location penalty charge

thing₁ is thing₂

The facts are grouped to a set of facts, which is related to a VCLL model. So all business rules in one model are based on the same set of facts. This insures that there are no contradictions of facts in one model.

3.1.2 Additional OCL constraints

In order to refine the metamodel the object constraint language (OCL) is used to define the VCLL more soundly.

3.1.2.1 Pool-lane-lane element relation

- 1 A Pool MUST contain 1 to n lanes.
(that is a Lane Element always belongs to one lane)

OCL constraint rule 1:

context Pool **inv**: lanes->size() >= 1

3.1.2.2 Sub-process-lane element relation

- 2 A Sub-Process MUST contain at least 2 lane elements
(to avoid meaningless, potentially endless empty hierarchy)

OCL constraint rule 2:

context Sub-Process **inv**: lane_elements->size() >= 2

3.1.2.3 Event types

- 3 StartEvent MAY BE of types Message, Timer, Rule, Link or Multiple.

OCL constraint rule 3:

context StartEvent **inv:** eventType = ‘Message’ or
 eventType = ‘Timer’ or
 eventType = ‘Rule’ or
 eventType = ‘Link’ or
 eventType = ‘Multiple’

- 4 IntermediateEvent MAY BE of types Message, Timer, Error, Cancel, Compensation, Rule, Link or Multiple.

OCL constraint rule 4:

context IntermediateEvent **inv:** event_type = 'Message' or
event_type = 'Timer' or
event_type = 'Error' or
event_type = 'Cancel' or
event_type = 'Compensate' or
event_type = 'Rule' or
event_type = 'Link' or
event_type = 'Multiple'

- 5 EndEvent MAY BE of types Message, Error, Cancel, Compensation, Link, Multiple or Terminate.

OCL constraint rule 5:

context EndEvent **inv:** eventType = 'Message' or
 eventType = 'Error' or
 eventType = 'Cancel' or
 eventType = 'Compensate' or
 eventType = 'Link' or
 eventType = 'Multiple' or
 eventType = 'Terminate'

3.1.2.4 Sequence flow

- 6 SequenceFlow MUST NOT connect two FlowObjects from different Pools.

OCL constraint rule 6:

context SequenceFlow

inv:

```
let    pA : Pool = self.target.lane.pool
```

```
let pB : Pool = self.source.lane.pool
```

in

$$p_A = p_B$$

The both `self.*.lane.pool` evaluate to `Pool` instances because of the cardinality 1.²⁷

- 7 StartEvent MUST NOT be a target object of a SequenceFlow.

- 8 EndEvent MUST NOT be a source object of a SequenceFlow.

- 9 Artifact MUST NOT be a source of a SequenceFlow.

- 10 Artifact MUST NOT be a target of a SequenceFlow.

- 11 Pool MUST NOT be a source of a SequenceFlow.

- 12 Pool MUST NOT be a target of a SequenceFlow.

- 13 Lane MUST NOT be a source of a SequenceFlow.

²⁷ Compare OCL 2.0 Specification. ptc/2005-06-06, p.31-33.

- 14 Lane MUST NOT be a target of a SequenceFlow.
- 15 DataObject MUST NOT be a source of a SequenceFlow.
- 16 DataObject MUST NOT be a target of a SequenceFlow.
- 17 Role MUST NOT be a source of a SequenceFlow.
- 18 Role MUST NOT be a target of a SequenceFlow.
- 19 Constraint BR MUST NOT be a source of a SequenceFlow.
- 20 Constraint BR MUST NOT be a target of a SequenceFlow.
- 21 Decision BR MUST NOT be a source of a SequenceFlow.
- 22 Decision BR MUST NOT be a target of a SequenceFlow.

OCL constraint rules 7, 10, 12, 14, 16, 18, 20, 22:

context SequenceFlow

inv: target -> forAll(t | not
 (t.ocIsOfType(StartEvent) or
 t.ocIsOfKind(Artifact) or
 t.ocIsOfType(Pool) or
 t.ocIsOfType(Lane) or
 t.ocIsOfType(DecisionBR)
)

OCL constraint rules 8, 9, 11, 13, 15, 17, 19, 21:

context SequenceFlow

inv: source -> forAll(s | not
 (s.ocIsOfType(EndEvent) or
 s.ocIsOfKind(Artifact) or
 s.ocIsOfType(Pool) or
 s.ocIsOfType(Lane) or
 s.ocIsOfType(DecisionBR)
)

3.1.2.5 Message flow

- 23 MessageFlow MUST connect two objects from different Lanes

OCL constraint rule 23.

context MessageFlow

inv:

let 1A : Lane = self.target.ocIsAsType(Lane Element).lane
 let 1B : Lane = self.source.ocIsAsType(Lane Element).lane
 in
 1A <> 1B

The both self.*.lane evaluate to Lane instances because of the cardinality 1²⁸ and re-typing.²⁹

- 24 StartEvent must not be a source object for a MessageFlow.
- 25 IntermediateEvent must not be a source object for a MessageFlow.
- 26 EndEvent must not be a target object for a MessageFlow
- 27 Artifact MUST NOT be a source of a SequenceFlow.
- 28 Artifact MUST NOT be a target of a SequenceFlow.
- 29 Lane MUST NOT be a source of a MessageFlow.
- 30 Lane MUST NOT be a target of a MessageFlow.

²⁸ Compare OCL 2.0 Specification. ptc/2005-06-06, pp.31-33.

²⁹ Compare OCL 2.0 Specification. ptc/2005-06-06, p.28.

- 31 Gateway MUST NOT be a source of a MessageFlow.
 32 Gateway MUST NOT be a target of a MessageFlow.
- 33 DataObject MUST NOT be a source of a MessageFlow.
 34 DataObject MUST NOT be a target of a MessageFlow.
- 35 Role MUST NOT be a source of a MessageFlow.
 36 Role MUST NOT be a target of a MessageFlow.
- 37 Constraint BR MUST NOT be a source of a MessageFlow.
 38 Constraint BR MUST NOT be a target of a MessageFlow.
- 39 Decision BR MUST NOT be a source of a MessageFlow.
 40 Decision BR MUST NOT be a target of a MessageFlow.

OCL constraint rules 26, 28, 30, 32, 34, 36, 40:

context MessageFlow


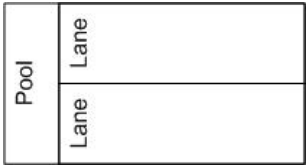
inv: target -> forAll(t | not
 (t.oclIsOfType(EndEvent) or
 t.oclIsOfKind(Artifact) or
 t.oclIsOfType(Lane) or
 t.oclIsOfKind(Gateway) or
 t.oclIsOfType(DecisionBR)
)

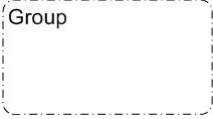





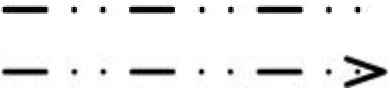
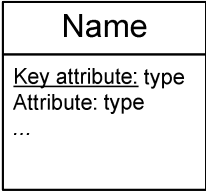
OCL constraint rules 24, 25, 27, 29, 31, 33, 35, 37, 39:

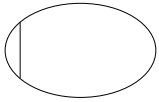


context MessageFlow



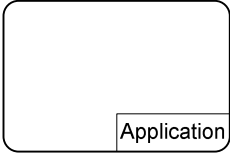



inv: source -> forAll(s | not
 (s.oclIsOfType(StartEvent) or
 s.oclIsOfType(IntermediateEvent) or
 s.oclIsOfKind(Artifact) or
 s.oclIsOfType(Lane) or
 s.oclIsOfKind(Gateway) or
 s.oclIsOfType(DecisionBR)
)



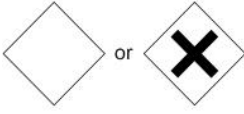


3.1.3 Graphical Notation of the VIDE CIM level language

Model element	Description	Graphical notation
Pool	A Pool represents a Participant in a Process. It is also acts as a “swim lane” and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.	
Lane	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities.	

Group	A grouping of activities that does not affect the Sequence Flow. The grouping can be used for documentation or analysis purposes. Groups can also be used to identify the activities of a distributed transaction that is shown across Pools.		
Media Object	Is an unstructured data that could be attached to every element in the diagram. MediaObjects could be hand-written texts, recorded audio data of interviews or videos enriching the background knowledge of the model element they are annotated to.	Video	
		Audio	
		Text	
Sequence Flow	A Sequence Flow is used to show the order that activities will be performed in a Process.		
Message Flow	A Message Flow is used to show the flow of messages between two entities that are prepared to send and receive them. In BPMN, two separate Pools in the Diagram will represent the two entities.		
Association	An Association is used to associate information with Flow Objects. Text and graphical non-Flow Objects can be associated with the Flow Objects.		
Data Object	Is an interface to a data view, for example a list of workers in the warehouse with their data used to choose the appropriate one to deliver goods to customers.		

Attribute	Each DataObject or Relationship instance may have attributes that describe object-specific properties.	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p style="text-align: center; margin: 0;">Name</p> <p><u>Key attribute:</u> type</p> <p>Attribute: type</p> <p>...</p> </div>
Relationship	Is used to associate DataObject instances with each other. This type of association may also have additional attributes that contain relation-specific information.	<div style="text-align: center;"> <p>(0..*) (1..1)</p> <hr style="width: 50%; margin: 0 auto;"/> </div>
Role	Is a model element that derives from the organisational view and creates and link between the organisational and the process view.	
Person	Each person may be assigned a role in the organisation. A person may also be assigned no or many roles depending on qualifications these roles require.	
Organisational Unit	Each organisational unit is a subpart of the enterprise hierarchy. It is related to roles as roles may be parts of organisational units.	
Constraint BR	This class [SSW06] represents a functional category of Business Rules that restricts the structure or properties of actions.	<div style="border: 1px solid black; padding: 5px; width: fit-content;"> <p style="color: orange;">It is necessary that each</p> <p style="color: blue;"><u>rental</u> has exactly one</p> <p style="color: blue;"><u>requested car group</u>.</p> </div>

Task	A Task is an atomic activity that is included within a Process. A Task is used when the work in the Process is not broken down to a finer level of Process Model detail.	
Sub-Process	A Sub-Process is a compound activity that is included within a Process. It is compound in that it can be broken down into a finer level of detail (a Process) through a set of sub-activities.	
Workflow task	A workflow task is one step in a workflow. It represents the actions that are executed by the activity which is called in this step. Additionally to its description it contains the application which is evoked by the WfMS in this particular step.	
StartEvent	As the name implies, the Start Event indicates where a particular process will start.	
Intermediate Event	Intermediate Events occur between a Start Event and an End Event. It will affect the flow of the process, but will not start or (directly) terminate the process.	
EndEvent	As the name implies, the End Event indicates where a process will end.	

Parallel Gateway	It is a place in the Process where activities can be performed concurrently, rather than sequentially.																
Event-based Decision	This Decision represents a branching point where alternatives are based on an Event that occurs at that point in the Process. The specific Event, usually the receipt of a Message, determines which of the paths will be taken. Other types of Events can be used, such as Timer. Only one of the alternatives will be chosen.																
Data-based Decision	This decision represents a branching point where alternatives are based on conditional expressions contained within the outgoing Sequence Flow. Only one of the alternatives will be chosen.																
Inclusive Decision	This Decision represents a branching point where alternatives are based on conditional expressions contained within the outgoing Sequence Flow. In some sense it is a grouping of related independent binary (Yes/No) Decisions. Since each path is independent, all combinations of the paths may be taken, from zero to all. However, it should be designed so that at least one path is taken. A Default Condition could be used to ensure that at least one path is taken.																
Complex Decision	Complex Gateways are present to handle situations that are not easily handled through the other types of Gateways. Complex Gateways can also be used to combine a set of linked simple Gateways into a single, more compact situation.																
Decision BR	Is a kind of a Business Rule that helps making decisions during process execution through differentiating between possible situations.	<table border="1"> <thead> <tr> <th colspan="2">Condition</th><th>Activity</th></tr> <tr> <th>Expected turnover</th><th>Current working load</th><th></th></tr> </thead> <tbody> <tr> <td>low</td><td>-</td><td>terminate</td></tr> <tr> <td>high</td><td>low</td><td>terminate</td></tr> <tr> <td>high</td><td>high</td><td>qualification</td></tr> </tbody> </table>	Condition		Activity	Expected turnover	Current working load		low	-	terminate	high	low	terminate	high	high	qualification
Condition		Activity															
Expected turnover	Current working load																
low	-	terminate															
high	low	terminate															
high	high	qualification															

3.2 Modelling example

This section provides an example which depicts one scenario in all views of the VCLL. It is based on an example provided by SAP and deals with the generation of business opportunities. The example will demonstrate how

VIDE applications are modelled on the CIM level and integrated into existing software environments. The opportunities identification should extend an already existing Customer-Relationship-Management (CRM)-System. Therefore the orchestration ability of VIDE is used. The new part is designed as two new VIDE applications that are linked by the WfMS. In the orchestration model below each activity represents the call of one application. The called application is stated at the lower right corner of each workflow activity. After the first activity there is a branch, where the process can be terminated.

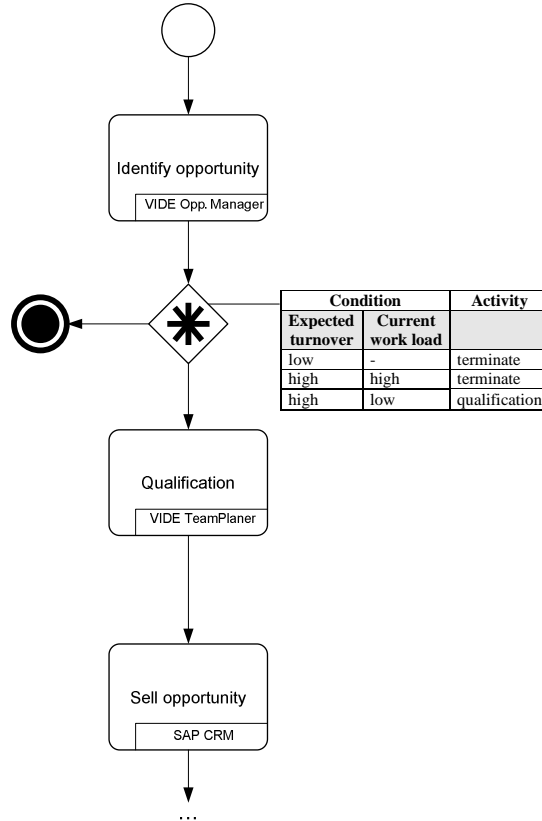


Figure 35: Process view, aggregated for orchestration

As the last activity “sell opportunity” is just invoking an existing system only the first and second activity are refined for their implementation. This is done in the next model. It describes the business process that should be supported by the software. Additionally the data that is used, like “customer data”, roles, like “office based sales employee” and business rules are used. There is one constraint business rule attached to the whole activity “identify opportunity” which describes further constraints on the business level. Furthermore decision business rules are used for the branching of the control flow. Moreover two media objects are used. One is video and the other one a document which creates a link to the information that has been gathered during the previous requirement analysis.

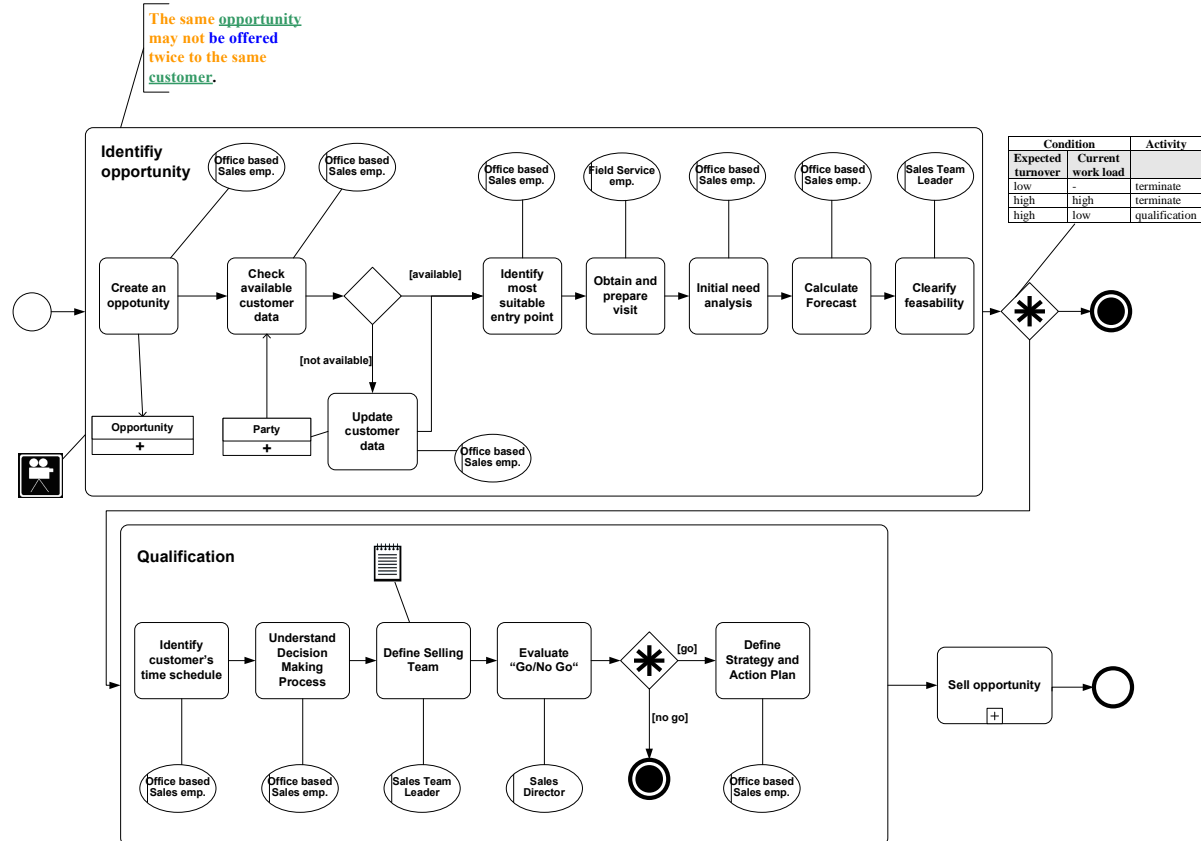


Figure 36: Detailed process view

The data objects and the roles are further defined in different diagrams. The data view shows three data objects, their attributes and their attribute types. On the CIM level only business relevant attributes are specified. The organisational view shows the hierarchy of the different roles, e.g. the field service employee is reporting to the sales team manager. Each role is assigned to the according department and the employees are assigned to roles. The assignment of employees to roles can be used by the WfMS by the instantiation of roles.

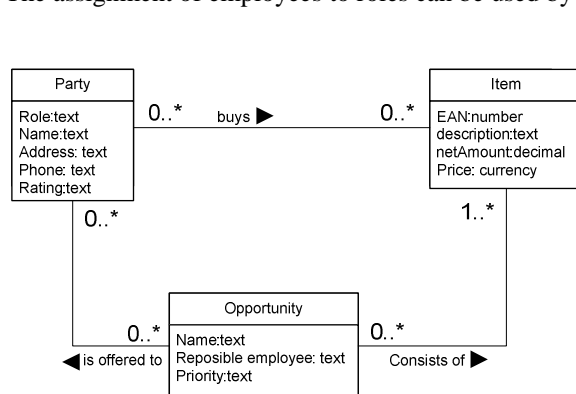


Figure 37: Data view

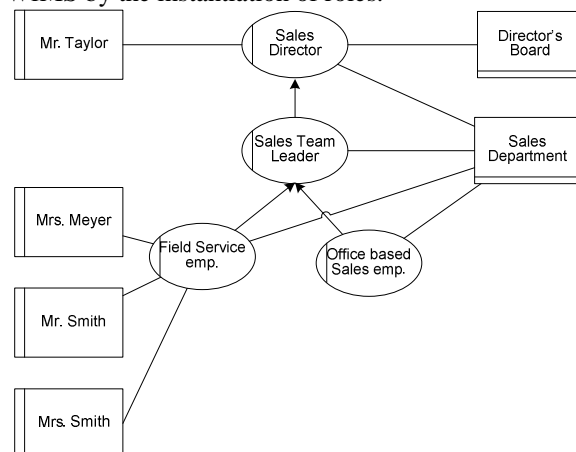


Figure 38: Organisational view

The last part of this example shows the facts that support the constraint business rule used in the process view. The facts are defined according to SBVR. As follows:

Item is product

Service is product

Customer buys product

Product can be an opportunity

thing₁ *is* thing₂

The facts can be used in this example for instance to illustrate, that the business rules that applies for opportunities includes services as well, because a product can be an opportunity and a service can be a product.

4. Business processes supported by VIDE

This chapter is dedicated to task 7.1, which aims at finding criteria for business processes that are supported by VIDE. Therefore in section 4.1 a classification schema for business processes based on the literature is developed. In section 4.2 the business processes supported by VIDE are characterised and not supported types of business processes are excluded based on this classification schema.

4.1 Classification of business processes

Business processes can be classified by different criteria. An important criterion for the use and especially the economic benefit of a software support for business processes is their *repetition rate*. The repetition rate as criterion for the classification of business processes is proposed by several authors [LeRo00], [Schm02], [Maur96], [Ders99], [Rath94], [Reij03], [Giag01] and [PiRe85]. Despite this criterion being very common the values that describe the repetition rate differ between different authors. In order to categorise different possible values we distinguish between three categories of repetition rate: singular, sometimes, frequently. Singular business process only execute once. These are business processes which are individual for each customer or are research and development processes which are not standardised. Business processes that are executed sometimes are not processes that occur in the daily business but occur more than once. An example would be the creation of a balance sheet once a year. The last category contains business processes that occur frequently. These are processes from daily business, which can include variants, but are standardised and documented.

A second criterion for the classification of business processes is their *degree of structure*. The degree of structure as classification criterion is used by several authors, e. g. [She+97], [DHLS96], [Ders99], [Maur96], [PiRe85], [Schm02] and [Aals99]. The distinction between different degrees of structure is stated differently. [BZG02] differentiate between ad-hoc processes and structures, pre-defined activities but ad-hoc processes, and pre-defined processes. By ad-hoc processes they mean business processes that are not structured, planned and documented. Their run-time behaviour is defined not until their execution. The second category consists of planned activities, which can be aggregated to a business processes at runtime. The third category consists of planned, managed and standardised business processes, where each activity as well as the whole process is known at its build-time. The other references mentioned above describe the first category as hastily formed or unstructured. The second category is not mentioned in all references. The third one is described as structured or formally defined. In order to get an intuitive formulation the categories are described as unstructured, semi-structured and (fully) structured. Unstructured refers to business processes where the activities and the control flow of the business process is not defined at their build-time. The opposite are structured business processes. Business processes are classified as semi-structured if their activities or their control flow are partly defined at build-time.

The next criterion is the *alignment of business processes* to strategic levels [Heil94], [ZhCh03], [Gui+06], [Korh07]. Traditionally in economics three different levels are defined: the strategic, the tactical and the operational level. Strategic business processes serve the purpose of long-time planning and definition of goals. These business processes usually require creativity and are not standardised. In order to realise strategic goals the strategic business processes are refined into tactical business processes. They usually have a mid-time range. These tactical business processes are refined again into operational business processes. The operational business processes are executed in every-day work. The creation of business value is done by this type of processes.

The next criterion is the stability or *frequency of changes* of the business processes [AaHe02], [Maur96] [Ders00]. Some business processes are have to be adapted frequently, e.g. for each project. Other are changed rarely and other are very stable. The last category e.g. describes business processes that are predefined by laws, which won't be changed for a long time.

Another attribute of business processes is their *granularity* [She+97] [BeZu99]. The can be modelled in a very detailed manner, so that the activities of the processes can be further refined in a reasonable manner. Compounded business processes have parts that are detailed but other parts that can be refined by a detailed process. The highest granularity is aggregated business processes. They are often depicted as value chains. They show the relation and order of groups of process steps, e. g. that the marketing activities are done before the sales activities.

Additionally several authors classify business processes by the *value* they create [LeRo00]. [LeRo00] distinguish between business processes of low and of high business value. Business processes that create a low value are

mostly administrative and support processes. They are necessary in order to create goods or services but do not create saleable products. Business processes with a high value creation are customer-oriented core processes.

Furthermore business processes can be classified by their scope as *intra- or inter-organisational* [DaSh90] [BZG02] [Haus96]. Intra-organisational business processes take place within one enterprise. All organisational units, hardware and software systems that take part in the processes belong to the same enterprise. Inter-organisational business processes take place between two or more different enterprises. This type of business processes requires interfaces between the application systems that are used in different enterprises. Judicial aspects have to be considered and security aspects have to be taken into account.

In addition, the use of *persistent data* of a business process can be different [PiRe96] [Kale98]. Business processes can just check information or transform a defined input into an output. This type of business process is very rare. They don't use any persistent data. The second type of business processes uses persistent data but does not create or change it. The largest group of business processes use, create and change persistent data.

A very important criterion for the classification of business processes is the *level of automation* [She+97] [Deru96], [Jung05], [KiMa05]. Three levels are distinguished manual, semi-manual and automated. Manual business processes are executed by employees without using application systems, e. g. service or consulting processes. Semi-automated business processes are executed by humans but supported by application systems, e. g. an employee enters the personal data of a customer in an application and the system checks the consistency of data. Automated processes run without human interaction. They are performed completely by application systems, e. g. bookings on bank accounts from one bank to another, which run as batch job every night.

Two other attributes that classify business processes are the *number of process participants* [She+97], [UMB99] and the *number of parallel instances* [Ment99]. The number of processes participants is classified into two categories: high and low. The number of parallel instances can be one, which means there is no parallelism. It can be also be some or many. Some means a small number of instances below ten.

Another attribute of a business processes is data-driven, referring to the data that is involved in the business process. It is the necessity of using *transactions*, which can either be required or not. If it is required the business process has to ensure that it will be completed successfully or comes back to the starting state again, e. g. the transfer of money from one bank account to another, has to be done completely and shouldn't stop after withdrawing the money from the first account and before in-payment to the second account.

The attributes for the classification of business processes and their possible values are summarised in as morphological box in Figure 38.

Attribute	Value		
repetition rate	singular	sometimes	frequently
degree of structure	unstructured	semi-structured	structured
alignment	strategic	tactical	operational
frequency of changes	never	sometimes	often
granularity	detailed	compounded	aggregated
value creation	low		high
process scope	intra-organisational		inter-organisational
usage of persistent information	none	low	high
level of automation	manual	semi-automated	automated
# of process participants	low		high
# parallel instances	one	some	many
transaction necessity	required		not required

Figure 38: Morphological box for business process classification

4.2 Criteria of Business processes supported by VIDE

After criteria for the classification of business processes have been presented, this section classifies the business processes that are supported by VIDE. For this purpose, for each category defined above, the supported business processes are classified.

For the repetition rate, VIDE is able to support all types of business processes. But in addition to other software development methodologies the software development is too expensive for a process which is only executed once in the same way. Therefore a software development project is only reasonable if there is a trade-off between the resources spent in software development and the benefit the developed software creates. As VIDE is aimed particularly at rapid software development, the software development is going to become less expensive and therefore more reasonable for business processes that are only executed sometimes.

Concerning the second criterion, only defined parts of a business process can be implemented. Therefore structured business processes are supported by VIDE. Semi-structured business processes can be treated with the VIDE methodology in two ways. If the control flow of the business process is completely available then it could be used for the orchestration of VIDE applications based on a workflow management system. Otherwise the structured and detailed activities can be implemented using the CIM-to-PIM transformation wizard that VIDE offers. Unstructured or ad-hoc business processes are not supported by VIDE as the logic of the business processes is too vague to create an executable description.

Regarding the strategic alignment of business processes VIDE could support all three levels. But an implementation for the support of creative decisions which have to be taken in strategic or tactical business processes are difficult to describe as business processes and to implement in software. Therefore VIDE supports especially the implementation of everyday business processes with relation to internal or external customers, which are located at the operational level.

Similar to the repetition rate the frequency of changes has an economic impact on the software development process. Business processes which are unchanged or rarely changed just need to be implemented once and can stay unchanged. Unfortunately changing business models, shorter product lifecycles and new competitors in markets increase the need to change business processes and shorten the time in which they remain unchanged. Therefore the software that supports business processes has to be changed more often as well. As VIDE starts its MDA approach at the CIM level and keeps the relation between CIM and PIM objects (see WP 5) the implementation of changes is relatively fast, because changes in business processes can be propagated to the PIM level. Therefore VIDE supports frequently changed and unchanged business processes as well. But it's not economic reasonable to implement business processes that are changed faster than it took to implement them.

Regarding the granularity of business processes, two types are supported. Detailed business processes, which cannot be further refined from a business perspective, can be transformed into PIM models. Compounded business processes can be used for orchestration. The activities which can be further refined are regarded as black boxes and an appropriate application is invoked by the WfMS.

The value creation also addresses economic issues. From an implementation point of view business processes with a low value creation as well as processes with a high value creation can be implemented with VIDE. But the benefit that arises from an implementation of a business processes with a low value creation can be less than the effort for the implementation. Therefore VIDE especially supports business processes with a high value creation.

The process scope that VIDE regards is on intra-organisational business processes. The modelling languages VIDE uses on CIM and PIM level don't consider special information such as the description of interfaces or mechanisms for information hiding between different enterprises. Because of the fact modelling of inter-organisational business processes and software is an own field of research [RöSc01], [Schu02] [ScOr01], this kind of models are not in the scope of the project and the methodology being developed.

Concerning the usage of persistent information, VIDE is designed to handle persistent data. Most business applications use, create or manipulate data. Therefore the VIDE CIM level language has its own DATA view in order to describe data objects and their usage in the business process. PIM level language elements for data definition and queries on databases have been introduced. Therefore VIDE can handle all three types of business process in regard to their usage of persistent data.

The level of automation that business processes possess is crucial for their implementability. VIDE supports business processes that are fully automated. They can be described at CIM and PIM level and/or be orchestrated in the sense of the VIDE approach. Semi-automated business processes can be described on the CIM level, because the CIM modelling language also allows the description of activities that are executed manually. However, on the PIM level, manual activities cannot be described. Therefore in semi-automated business processes the whole business process is described at the CIM level but only the automated part is transferred to PIM level. Manual business processes can be described on the CIM level in VIDE, but are not implemented.

Regarding the number of participants and parallel instances that VIDE can support a limitation is only given by the target platform on PSM level. If the PSM level supports multiple instances and is able to handle a large number of users VIDE can be used for this kind of system.

Regarding the last classification criterion, the need of a business process for transaction support, VIDE partly supports transactions. As VIDE is based on databases, the transaction concepts of databases can be used. Therefore a transaction support for data is realised. But transaction support for the process steps itself is only partly possible. The VIDE CIM level language offers the concept of compensation. This does not allow a roll-back to be done, but defines actions that have to be undertaken in order to undo an activity. As the description of compensations on CIM level are done in the same way as the description of normal business processes they can be implemented at the PIM level as well. Therefore VIDE offers a full transaction concept for data and compensations for business process activities.

An overview of the type of business processes that are supported by VIDE gives the following table:

Attribute	Value		
repetition rate	singular	sometimes	frequently
degree of structure	unstructured	semi-structured	structured
alignment	strategic	tactical	operational
frequency of changes	never	sometimes	often
granularity	detailed	compounded	aggregated
value creation	low		high
process scope	intra-organisational		inter-organisational
usage of persistent information	none	low	high
level of automation	manual	semi-automated	automated
# of process participants	low		high
# parallel instances	one	some	many
transaction necessity	required		not required

Figure 39: Classification of business processes supported by VIDE

In general VIDE supports all types of executable business processes. The only requirement is that the business processes can be described by a set of actions, a control flow between them, and data objects the activities are working on. The type of business processes which are supported optimally are business processes which just display, create or change data. These actions are typically for administrative processes, such as booking a flight or the administration of a warehouse.

Other domains, such as real-time or embedded systems, are not in the focus of VIDE. Furthermore VIDE is not designed to depict very complex algorithms, such as those used in Artificial Intelligence systems, because these kind of systems require a large number of loops, branches and case differentiation which are difficult to describe in the control flow of the VIDE CIM level language.

5. Procedure Model for the VIDE software development process

This section describes the VIDE software development procedure model. It is structured in two parts. First existing software development procedure model are evaluated and then the VIDE software development procedure model is described.

5.1 Evaluation of existing software development procedure models

5.1.1 The waterfall model

This software development procedure model is one of the oldest existing process paradigms. It was elaborated in 1970 by ROYCE [Ro70] through deriving from more general system engineering process models (the stage-wise model by BENINGTON [Beni56] [Balz98]) and enriching it with feedback cycles [Somm07]. The waterfall model includes all the four basic activities – software specification, design and implementation, validation and evolution – and combines them in a sequential development process. This model was named after the waterfall-like diagram representing the process, where one phase flows into another, thus building a waterfall shape. This representation also means that the reports from the previous phase should be transferred to the next one, ensuring information is passed forward through the model.

In the following paragraph the main phases of the waterfall model are described. The classic model by ROYCE consists of five stages as shown in Figure 40. There also exist derivative models that extend these five phases over six and up to seven steps (BALZERT describes seven phases: system requirements, software requirements, analysis, specification, implementation, testing and maintenance [Balz98]). These latter models are not discussed here only the classic waterfall model by ROYCE.

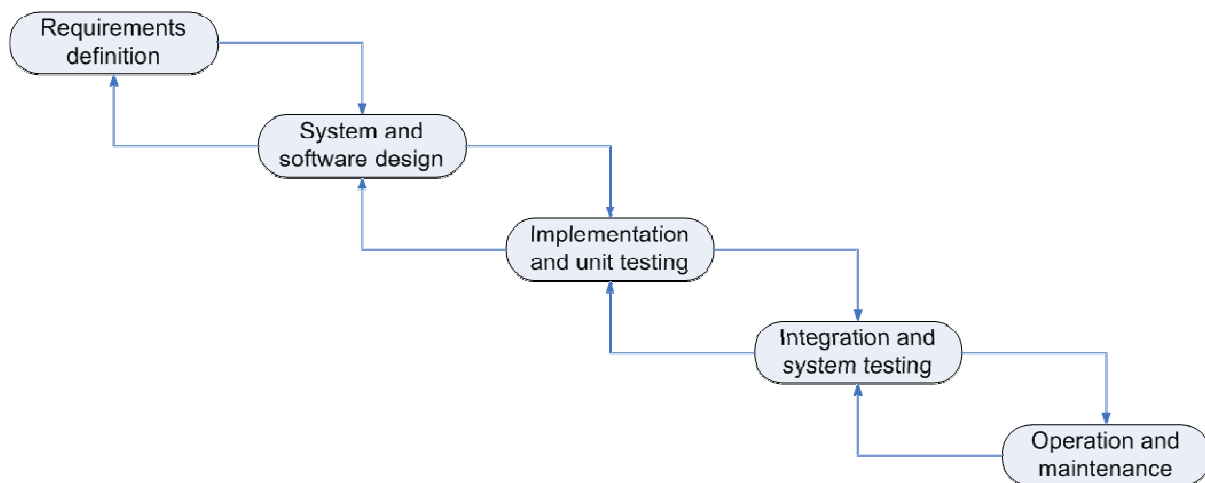


Figure 40: Waterfall process models [Somm07]

1. Requirements definition – in this phase, the initial requirements for the end product, its system environment and product users are defined and are later used in the next phases. This phase is sometimes divided into a phase of system requirements and a phase software requirements.
2. System and software design – in the first part the system and software requirements are extracted from the first phase. Then the overall system architecture evolves from these requirements. The next part is essentially designing of the software architecture.
3. Implementation and unit testing – as soon as software design is completed, it is implemented. The units of software code need to be tested for defects to see if its behaviour conforms to specification.

4. Integration and system testing – once software units have been verified, the units should be integrated in a complete system and tested as a whole. This procedure checks if the software meets its requirements as stated in the first phase.
5. Operation and maintenance – after the software has been delivered to the user, it should be installed and utilized. This covers the operation part, whereas maintenance part involves
 - correcting defects in the software product not revealed in the previous phases
 - designing new components that meet the changing user needs and
 - improving the existing implemented features.

It is stated that this phase could be the most expensive one compared to the previous four. If the complete software product exists for a long time, it is feasible to assume that there would appear new features needed to be included in the product, thus extending time spent on this stage. The same time expansion could be caused by the defective implementation, which in turn requires time efforts to correct these.

The advantages of the waterfall process model are as follows:

- each stage produces extensive documentation. Because all of the steps are carefully monitored, the process could resume at any stage in case of process interruption.
- this documentation suits other process models. It brings faster integration with other system projects if desired.
- the process is simply represented and requires less management efforts. The software product under development can be easily reviewed, stopped or enhanced in case of severe emergencies.
- user interaction is only needed at the beginning in the definition phase. User interaction usually takes much time and needs to be confirmed by both sides. In case these interactions are frequent it can result in slowing down the development.

The disadvantages of the waterfall process model are as follows:

- documentation produced at each stage could become overwhelming. If reports are more important than the development itself, then it is difficult to maintain the adequate software quality level at the same development velocity.
- requirements have to be stated in the very beginning resulting in less flexibility. If the development takes much time, as it is in case of large projects, it is almost impossible to anticipate all the needed features the system should possess at the end.
- each stage has to be executed sequentially which is sometimes inflexible. At the implementation stage, it might be desirable to integrate the units as soon as the next one is completed to check each and every relation. It would be prohibited by the waterfall model as the integration shouldn't be done until each and every unit is fully implemented and tested.

Hence, the waterfall process model should only be used if the requirements can be stated in the beginning of the process and are not going to change drastically during the development. This type of process model is also used in other engineering projects, which means that this process model could be used in large-scaled projects, especially for subprojects of larger system engineering projects [Somm07].

5.1.2 The V-model

This process paradigm proposes an extension to a waterfall model. It was introduced by BOEHM in 1979 and is enriched with quality assurance of the software process [Boeh79]. The V-model thus incorporates validation and verification in the software development process. Considering validation and verification this model could be represented in the V-formed diagram, which the V-model derives its name from (see Figure 41). The four basic activities of the system development process – software specification, design and implementation, validation and evolution – are sequentially executed with the possibility of changing cycles and constant creation of tests at different levels of abstraction.

Verification – the process that determines whether the components of the system and the system itself behave like they should after specification. The informal question asked during verification is as follows: "Is the product correct?"

Validation – the process that determines whether the software product is the appropriate one for the system requirements stated before and user interaction during system usage. Informally one asks the following question: "Is the product the right one?"

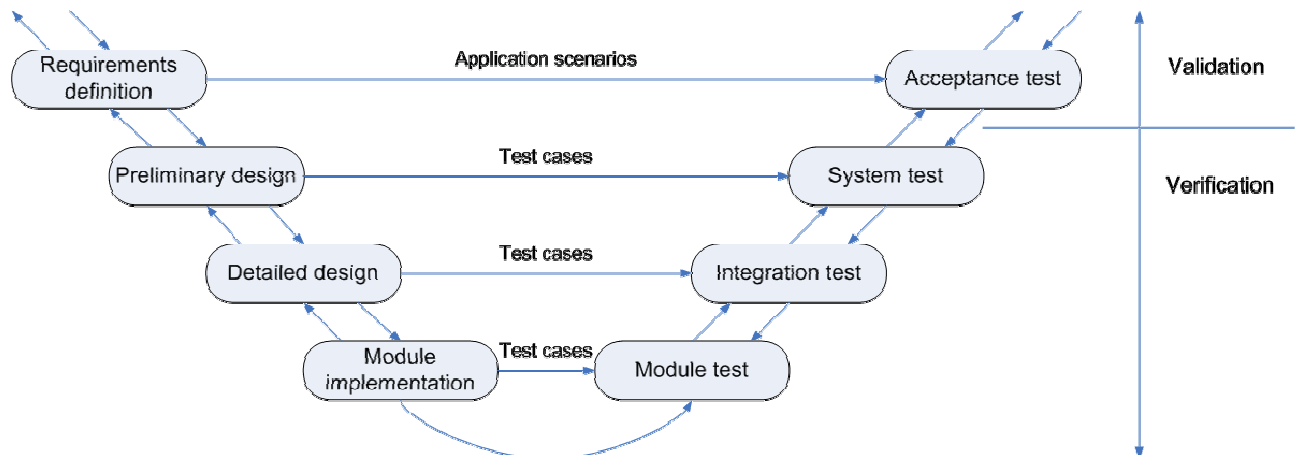


Figure 41: The V-model [Balz98]

The V-model consists of four sub-models:

- System engineering – deals with the development and maintenance of the system itself. This sub-model is in principle the core of the V-model that is very similar to the waterfall model. It indeed elicits requirements, makes a system design, implements the system, integrates components and documents each step extensively. In addition, each step has to be verified and the whole system has to be validated.
- Quality assurance checks whether products are conform to specification at the different levels of abstraction. It tests modules, relations between modules, the system as a whole and its acceptance by the users.
- Configuration management leads to adjustment of the system components. If some components are being refused or have to be more thoroughly elaborated, but have been included in the previous system releases, configuration management takes part and includes these changes in the next release.
- Project management supervises the project development as the whole. On the one hand, there are project plans that are being presented and later controlled by the means of comparison between "as-is" and plan data. On the other hand, project management also has to provide the software development environment to work with.

The V-model assumes that the software development and maintenance process consists of activities and products. The latter are the result of the former. The V-model describes the product states and relationships between products and activities. Activities can create products, change its state or the content, whereas some products run through state changes as shown in Figure 42. Firstly, it is planned to create a specific product – state "planned", secondly it is being elaborated and controlled through a certain developer – state "being processed". In the third stage later it is presented and taken in the configuration management – state "presented". In case of product refusal it has to be elaborated further and changes its state to "being processed" again, otherwise it passes the quality assurance and changes its state to "accepted". Feasible changes may only be incorporated in further product versions as stated by configuration management.

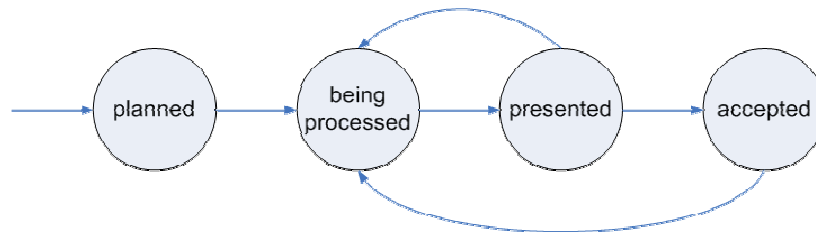


Figure 42: Product states in the V-model [Balz98]

There are also different roles defined in the V-model concept that can be associated with each activity. These roles describe the qualifications, experience and skills needed to conduct the associated activity. In each sub-model there is:

- one manager – presents the conditions of execution of an activity and is the highest arbitration.
- one responsible – plans, leads and controls the tasks of an activity.
- one or more executors – processes the planned tasks of an activity.

The V-model claims to be an universal one that is appropriate for different kind of processes. Because of vast number of different types that could not be unseen by the developers of the model there was a concept proposed that helps adjusting the model itself at the process requirements. This concept is called “tailoring” and takes place in two steps: request for proposal and technical tailoring. The former defines the activities and products before the development process begins through deletion of unnecessary entities. The latter defines the desired activities and products during the development process after the beginning of the containing activity. The goal of tailoring is to assure the are not too much unneeded documentations being produced, on the one hand, and that no important documents are missing, on the other hand.

The advantages of the V-model are as follows:

- integrated, detailed description of system engineering, quality assurance, configuration and project management. This division ensures every important aspect of the development process is documented and the product quality is assured.
- allows standardised system development. Through the universality claim it would be possible to apply this model to virtually every system engineering project. The tailoring concept should be useful to fulfil this.

The disadvantages of the V-model are the following:

- the concepts for large-scaled embedded systems are transferred onto other applications without criticism. These could be infeasible to maintain the role model proposed by this model in other kind of projects. For the small-scaled projects there is too much undesired administrative workload.
- threat of requiring certain software methods through division into data and function views. It may be desirable to introduce other views, to differentiate between function and output views.

Hence, the V-model is appropriate when developing large-scaled projects where extensive documentation and constant quality, configuration and project management are required. The projects with high security or other kind of risk would benefit from precise documentation and the possibility to qualitatively assure each step. The small- and middle-scaled projects may suffer from dominating reports needed to be written during the development. It is the same kind of danger the small projects would suffer when using the waterfall model, only that they would have to incorporate quality assurance in each step of the process.

5.1.3 Evolutionary development

The idea of this software development process model is that sometimes it is easier to understand what has to be developed through elaborating requirements by the means of customer try-out. The initial experimental product is exposed to the end user and feedback is collected directly in order to define the desired refinements. As soon as the refinement is implemented, it is being reviewed once again and the next portion of improvements is proposed. The final product is thus implemented through a series of stepwise refinement of the initial product draft. This development process combines, unlike the waterfall and V-model, the four basic activities of the software process in an interleaved manner rather than separately, allowing rapid feedback between development activities. This development model is illustrated in Figure 43.

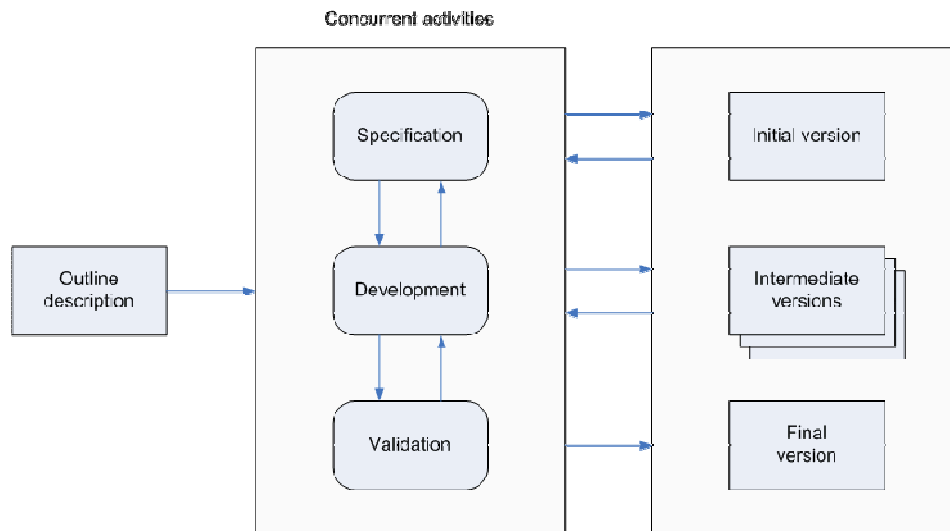


Figure 43: Evolutionary development process [Somm07]

There exist two essential types of evolutionary development:

1. Exploratory development – this approach has the objective to get the idea of user's requirements through stepwise development. It starts with the developing of the product at the parts that are known and then adds features demanded by the customer. The goal is to deliver the final product through gradually exploring customer's requirements.
2. Throwaway prototyping – this evolutionary development type deals with experimenting with customer's requirements that are poorly understood. The primary objective is not to develop a system, but to get the idea of system requirements through improvement of the existing customer's requirements.

The advantages of the evolutionary model are:

- the possibility of developing the system that meets the immediate needs of the customer. In this respect the evolutionary model is often more effective than the waterfall model.
- system specification can be developed incrementally. Thus, customers don't have to possess the complete knowledge of the system requirements they would like to achieve with the product. Instead, the rough image is enough at the beginning, allowing gradual refinement process.

The disadvantages of the evolutionary model are as follows:

- the process is not visible. In the systems that develop over many intermediate versions, it is often inefficient to produce documentation for each of the small changes. However, managers need to have feedback to be able to assess the process progress, which leads them to elaborate workarounds in order to get enough information about the process.
- systems are often poorly structured. Even a perfect software model can eventually get corrupted, when many continuous changes are made. New requirements could interfere with the existing structure, making it difficult to incorporate the according changes into the system.

Based on the advantages and disadvantages, it is preferable to choose the evolutionary software development model when small- and medium-scaled projects are taken into consideration. It is also useful for refining initial

poorly understood customer requirements and can help clarifying them so that another more structured approach could be chosen. On the contrary, the evolutionary development process model is not best suited for large-scale long-life projects, because it is difficult to consider all possible changes that could be made so that the final product can be unstable using this approach. A combination of the initial requirements refinement through evolutionary modelling and application of the more structured model afterwards may be desirable [Somm07].

5.1.4 Component-based software engineering

In large projects in which with many system parts have to be integrated, there might be source code sections in one system part that could be of interest for another one. In case the responsible developers know about the existing similar source code they might want to use, they can adjust the existing sections for their needs, thus saving efforts for development and related testing and maintenance. The evolutionary approach described in previous section and rapid system development that is connected to it, may profit from software reuse [Somm07].

Informal reuse of software code is sometimes called software cloning [BYMSB98]. The definition underlying this concept suggests that the copied source code is identical, but it is in fact misleading. Another advantage of is that this concept is able to find similar code and adjust it to the different needs. This concept differentiates between four different types of software clones, beginning with identical code segments that are relatively easy to identify and ending with semantically equivalent segments which identification may be difficult [Be02]. The reasons for cloning range from inability to develop a source code segment due to program or time limitations to text templating, where the copied text is customized for later use [KSNM05].

In the recent years, the process of software development based on software reuse has appeared and became more extensively used. This approach is called component-based software engineering (CBSE) and relies on the fact that there is an extensive pool of software components that can be integrated through a standardised framework. Some of the components may be so-called commercial off-the-shelf (COTS) products that provide their own functionality and can be further reused. The component-based software development process is shown in Figure 44.

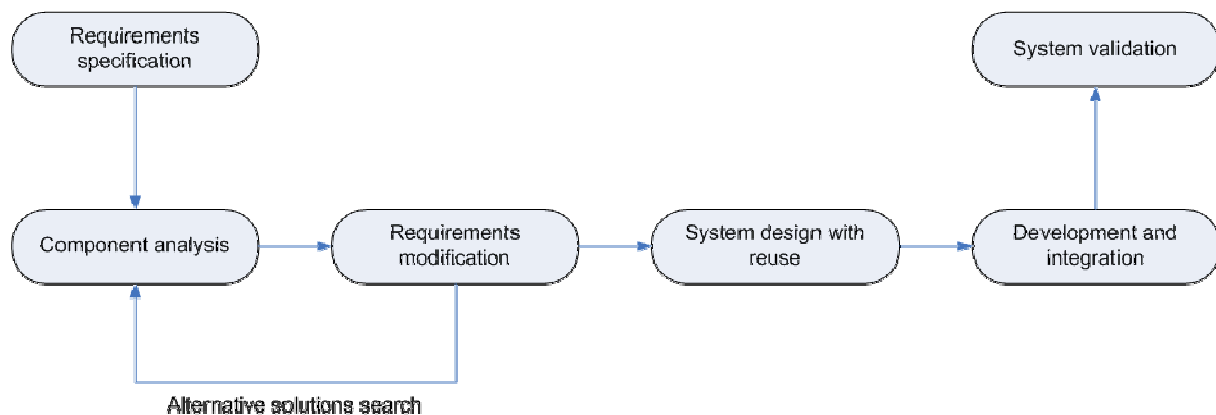


Figure 44: Component-based software engineering process [Somm07]

The first and the last phases of CBSE process are similar to the phases already described in waterfall model and others, the four phases in between differ from the rest:

1. Component analysis – after the requirements specification it is usually possible to search for components that implement this given specification. It is not guaranteed that there would be components that implement exactly the given specification, so the next similar ones are taken.
2. Requirements modification – depending on which components have been discovered and how well they fit into requirements implementation, the requirements specification itself should be reviewed in order to suit the available components. If the modification isn't feasible, the previous step may be repeated to find the alternative solutions.
3. System design with reuse – in this phase, the existing framework should be adjusted or a new one should be elaborated. Of course the components chosen for design are taken into account by system designers. If there have been no appropriate components found in the first phase and no alternative

solutions in the phase review after requirements modification, then some new system components should be developed.

4. Development and integration – new components are developed that cannot be obtained externally. After that, they are integrated with the existing ones (also COTS parts) into a final system. The system integration step is seen as a part of this step rather than a separate activity in this software development process model [Somm07].

The advantages of the CBSE process are as follows:

- reduced development costs and risks. Though some adjusting and development are made, these are relatively cost efficient because of the reduced time efforts. In addition, the future failure risks are reduced, because the reused modules have already been tested before. Of course, the changes components have to be tested again, but the test development time is saved.
- usually faster software development. Once again, it is related to reduced development time. Indeed, in case not that much time is spent on test development and maintenance, the software production should take place in rapid pace.

The disadvantages of CBSE process are listed below:

- requirement compromises may lead to a system that doesn't meet the user needs. If there are no components that fit into requirements specification and no alternative solutions have been found, then the components should be developed. If this is infeasible, then the requirements are not met, thus not providing for requirements.
- some control over the system evolution is lost. The newer version of the software components are not under control of the organisation using it [Somm07]. Therefore, it could be difficult to maintain the consistent development of this component further.

Hence, the CBSE development process is well suited for the projects that integrate many system parts from possibly different vendors. A service-centric project based on integrating web services from a range of suppliers can serve as a good example for this kind of the software development process [Somm07]. Where different components don't possess a standardised integrating framework or communication between different developer teams is complicated then this approach is not well suited. That is, much time and effort could be spent on evaluating the different software components in the search for appropriate source code segments that would fit into current needs. This search would not necessarily be successful and could result in time and resource issues for the project.

5.1.5 Incremental delivery

The waterfall model allows for structured software development, where every step is well documented and the input of the next step consists of the information from the previous one. It is complicated though to continue the development after completing the requirement stage if requirements are likely to change often, because every little change demands reworking of the requirement, design and eventually implementation stages. On the other hand, evolutionary development allows for delaying requirement and design stage completion, letting the important decisions be postponed until it is clear what is required. This kind of the development process may though lead to an unstructured and undocumented product that is difficult to modify and to maintain. In the worst case, if the new design or requirement decision is infeasible to incorporate, the product should be developed from scratch using the rest of the already developed system.

An approach called incremental delivery should combine the advantages of both waterfall and evolutionary models for software development. Essentially, it breaks the system development in a number of small increments that are each developed separately and delivered at the customer in turn. This process is illustrated in Figure 45.

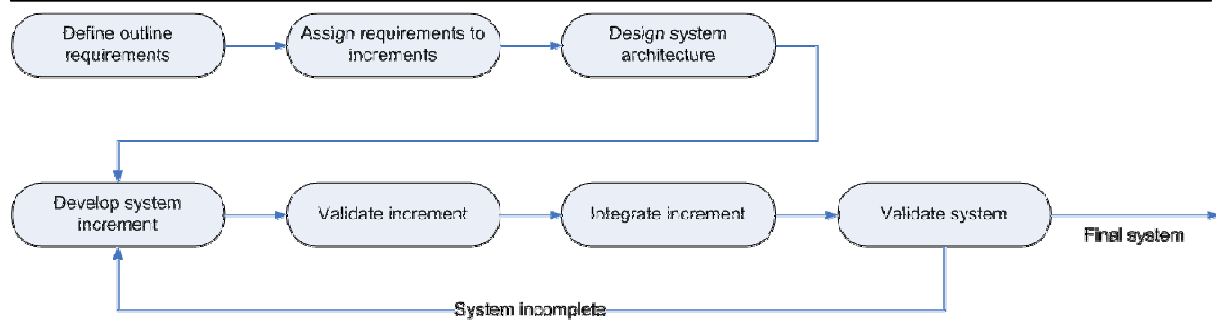


Figure 45: Incremental development process [Somm07]

Before breaking the development process down into a series of increments, the customers have to outline the services they would like the system to incorporate in the final version. These requirements are then distributed into a number of increments that deliver a part of system services each. The order in which services are assigned to increments depends on the service priority stated by the customer in the first phase.

As soon as the system architecture is designed and the increments are defined, the development of the first increment starts. Requirements for this increment are defined in detail and are sequentially elaborated through design, implementation and integration. During the development, new requirements to the system may come up, but changing the requirements for the current increment is not allowed.

After completing each increment, it is being immediately delivered at the customer. The services are put into realization, thus allowing the customer for experimenting with the product. With new increments implemented, the overall system functionality improves and customers' requirements to system and its components can change. As long as it doesn't disturb current increment development, new requirements are likely to be incorporated into system requirements and implemented in the later versions of the system or components.

The advantages of incremental delivery are as follows:

- customers benefit from the early system delivery. It helps identifying missing functionality, requirements and additional needs, which they would have expressed anyway even if the system would have been delivered as a whole. The customers are also able to use the system productively, even if it has temporarily limited functionality.
- customers can use the early increments as prototypes. Experimenting with prototypes gives the customers experience and helps them faster express their requirements for the later system versions.
- important system parts receive the most effort for testing. As the common services and system core are delivered first due to higher priority, these system components get the most time to be tested and be freed of defects. The important parts of a system are then more robust than those providing additional functionality.
- lower risk of overall project failure. Even if one or the other component is not working out well, the first part to be delivered is the system core and it is being made properly, so some of the later components are likely to be developed.

The disadvantages of incremental delivery are listed below:

- difficulties to map customer's requirement onto increments of the right size. Because each increment should be relatively small and deliver at least some functionality, it is hard to decide which part of the service is going to be developed with which increment.
- hard to identify common functionality before complete requirements specification. Later components use the same system interface and services as the former ones. It is thus important to identify common services used by all of the components, which is difficult if the requirements are not complete at the beginning.

Therefore, the incremental delivery development process is best suited to projects for which the requirements are not clearly defined in the beginning and may change in future. The customers should be prepared to be tightly involved in the development process, because at each step and especially after each increment their feedback is required. It is not suited for organisations that include the complete system specification in the contract, which is mostly the case with the government organisations.

This software development model belongs to a category called agile methods, referring to their early first draft system delivery and flexibility with respect to changing customer requirements. A variant of the incremental

delivery called extreme programming has been elaborated [Beck00]. It is based on a series of very small increments of the system services, customer interaction and pairwise programming. These should result in constant source code improvement according to new or changed customer requirements.

5.1.6 The software prototyping model

It is often impossible to apply the classic software development models such as the waterfall or the V-model to software development because of the incomplete system requirements or different design solution possibilities [Balz98]. In this case, developing a software prototype may be useful in order to solve these problems. This approach may be seen as a subcategory of the evolutionary software development process and is sometimes called software or throwaway prototyping [Somm07].

A software prototype is a first draft version of the final system product that demonstrates the implementation of the system requirements and possible design options. It can then be used for experimentation in the software development process in following ways [Somm07]:

1. During requirements engineering, a prototype can help to determine and evaluate the requirements for the system. The requirements should be reviewed from the technical and end user points of view to develop a set of requirements that satisfies them both in the best way.
2. During system design, a prototype can be used to explore the design solutions. Technical details should be hidden behind a user interface that can change depending on the end user design requirements. If those are incomplete or the end user is unsure what is better to include in the final system version, throwaway prototyping can help choosing the feasible design options.
3. During testing, a prototype can be used to run back-to-back tests with the final system that will be delivered to the end user. The test validation is the main goal of the prototype usage. The same tests are run with the prototype and the final system and depending on the difference in results the correctness of the final system can be measured.

The software prototyping process is shown in Figure 46 and describes the steps during the development. From the beginning the objectives of the prototype should be made explicit, because otherwise the goals of the prototype can be easily misunderstood by the end users and thus they will not be able to appreciate the benefits from the prototype development process. Defining the features to include or to leave out of the prototype is the next step in the process. In order to save time or costs for the prototype development, some uncritical requirements with respect to system functionality can be relaxed. Prototype development should produce an executable piece of software for the end users to experiment with, which is done in the last stage during prototype evaluation. The more the end users are getting comfortable with the system prototype, the more requirement adjustments they can find.

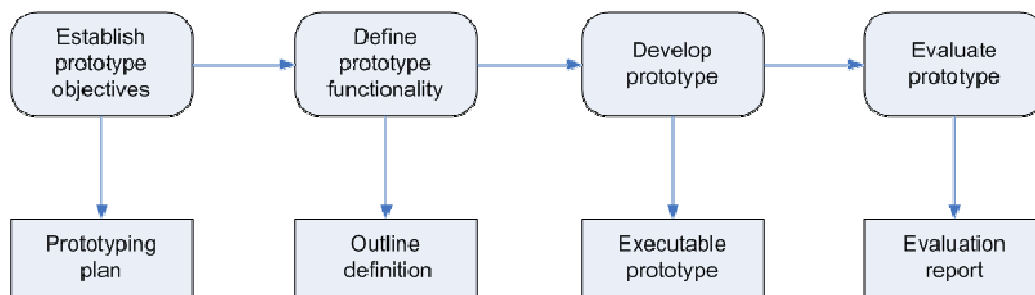


Figure 46: Software prototyping process [Somm07]

The advantages of the software prototyping development process are:

- software development risk reduction. Through early prototype application it is likely that the errors in requirements of design will be detected and eliminated in time, before the final product version is delivered to the end user.
- reduced development effort. Through application of the prototyping instruments it is possible to rapidly develop adjusted versions of the software product that incorporates changing customer requirements.
- a closer match of the system to user's needs. Through application of the prototypes in requirement engineering and system design more precisely tuned product emerges. Most of the

end user requirements are already implemented in the product, which in turn results in better product match.

The disadvantages of the software prototyping development process are the following:

- impossibility to incorporate the non-functional requirements. Quality standards are usually degraded for software prototypes. The system performance, security and reliability requirements may be relaxed during prototyping, which may result in incompatible changes to be made in order to implement those.
- prototypes are often seen as the documentation substitutes. Frequent changes lead to poorly documented software pieces that are extremely difficult to maintain. The only document provided is the source code, which is hard to use as the documentation basis in the long-term development.
- difficulties in prototype system maintenance. The changes during prototype development are likely to disturb the system structure, which in turn leads to problems in maintenance and possible need for system refactoring.

Hence, the software prototyping model can be used in the projects where the initial requirements are not clear or the design options are difficult to differentiate without further work. Experimenting thus, with poorly understood requirements, the better system requirements and design can be elaborated. Prototypes can also be incorporated into other software development models in order to clarify requirements or make a decision with respect to design options. This software process model is not suitable for projects where end users are not available for discussion during prototyping. Also, the software prototypes are not always included in the development contract, which makes it difficult to apply this software development model.

5.1.7 The object-oriented model

As already stated in the section describing component-based software engineering, software reuse may take place and sometimes it is reasonable to implement components by customizing the existing ones. This paradigm is also used in object-oriented software engineering, where the reuse is made through modularity, encapsulation, inheritance and polymorphism. This reuse can take place at the different levels of abstraction [Balz98]:

- requirements definition – Object-Oriented Analysis (OOA) is applied here, which results in reuse of subsystems and class hierarchy of the software products.
- technical design – Object-Oriented Design (OOD) takes place, which results in reuse of design options.
- implementation – this is the lower stage where classes and class libraries can be reused by the developers.

The reuse of the components can be based on the own developed software pieces or on the purchased class libraries or OOA concepts [Balz98]. All of the reused system components have to be made accessible through common reuse archives. The point of time at which the components are submitted into an archive is important for the object-oriented model. The system components may be filed directly during the development, after development completion or even after the dedicated developer team has analyzed and identified the reusable pieces of software. These aspects should certainly be taken into account during object-oriented development [Balz98].

The object-oriented software development process is shown in Figure 47. It starts with the definition of the system requirements, which in turn reuses possible existing approaches for the particular kind of a problem. As soon as an OOA model has been completed, it can be filed into the reuse archive, which then serves as reference for the future requirements definition. In the next step design decisions with additional help of the reuse archive are made and the OOD model is constructed, which in turn is stored into the reuse-archive for reference purposes. The following step is the implementation that is searching for the reusable classes in the archive. The output of this step is the object-oriented product, which can also serve as input for reuse archive. As soon as at least two object-oriented products are completed, they can be analyzed together in order to detect common generalised components for future reuse. During the development process, changes can be made to requirements definition (through design decisions) and design elaboration (through implementation issues) in case these are incompatible with the models that have been developed in following steps.

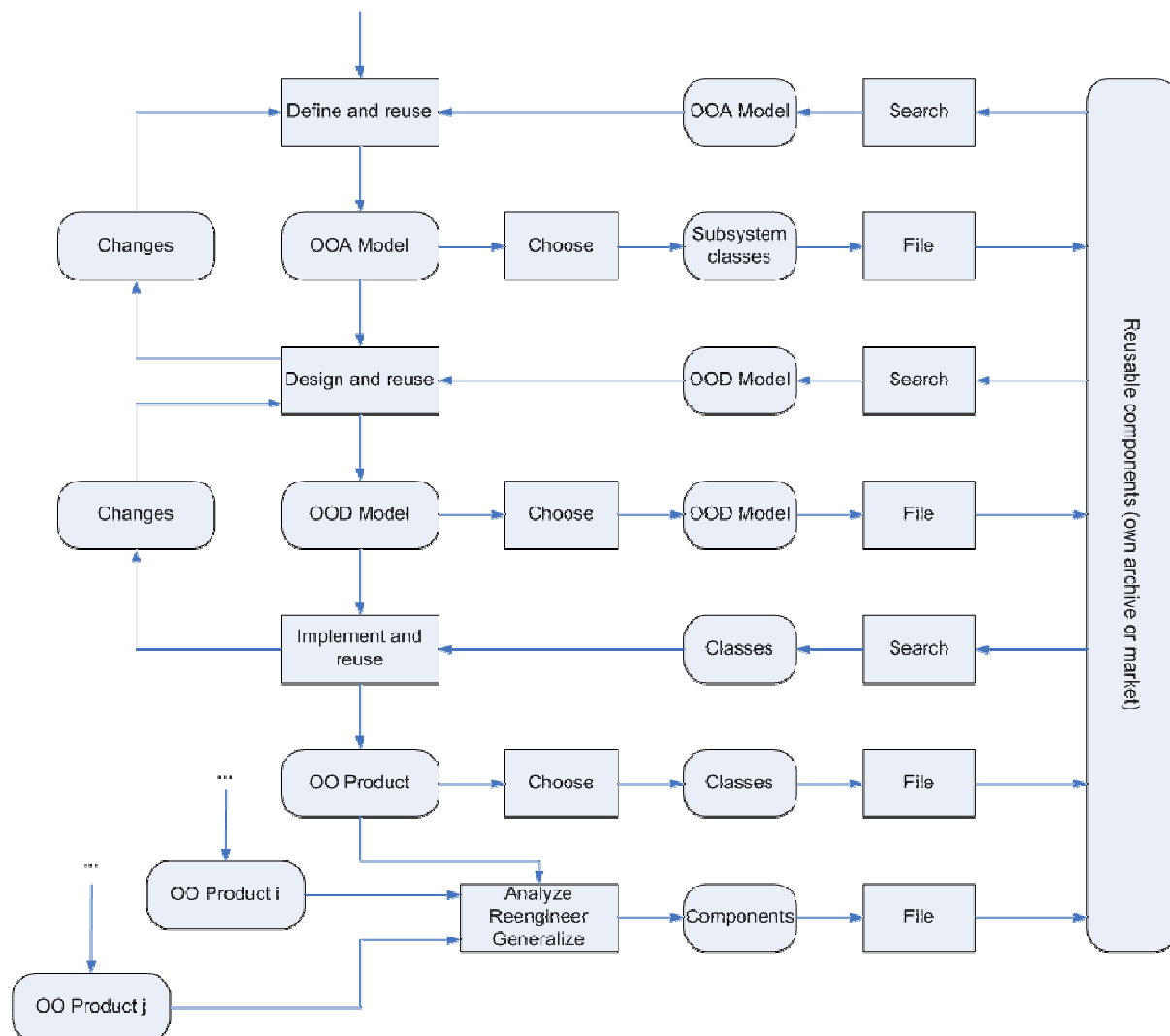


Figure 47: The object-oriented development process [Balz98]

The advantages of the object-oriented software development process are:

- productivity and quality improvement. Clearly the software reuse accelerates software development, because efforts for initial analysis and test development take not that much time. Quality gains through the additional testing after the adjustment of the reusable components.
- usage of semi-finished products, focus on own strengths. If it is too complicated for the team to develop a certain system component from scratch, it is then acquired from the external sources or the own reuse archive, which in turn consists of independently developed or purchased components.

The disadvantages of the object-oriented software development process are the following:

- product development is directly connected to object-oriented techniques. If the development team is using a different programming paradigm, say procedural programming, it is difficult to maintain the development process intact and at the same time to incorporate the reuse archive based on object orientation.
- appropriate infrastructure (reuse archive and organisation) is required. The development team has to have full access to reuse-archive and to request missing components to be purchased in case these are not available in the archive and can't be developed independently.

Therefore, the object-oriented software development model is best suited for projects developing products that can benefit from reuse. This approach can be well combined with evolutionary and incremental process models, because object orientation allows for rapid change conduction [Balz98]. It is also well suited for the projects applying object-oriented methods in the development process. It is not that suitable for projects that have to

develop all of the system components on its own, without purchasing or reusing existing software pieces due to financial limitations or inapplicability. As already stated, the projects that use another programming paradigm as object orientation would probably benefit a little from the object-oriented software development model.

5.1.8 Concurrent Engineering

Most of the classic software development models connect the development activities in a sequential manner. Rather than perform one development activity after ending another one, the concurrent engineering model proposes to allow for parallelization of those. This model stems from manufacturing industry, which is aimed at the production time minimization under separation of product design from product fabrication. It unites all of the involved departments in a team that is focused on a synchronous development [Balz98]. All of the aspects of the final product should be considered in advance, so that the reengineering doesn't take that much time. In addition, as long as it is possible, the development activities have to be conducted in parallel. That means that after completing only a part of requirements or even earlier, the design phase begins, which is directly followed or even overlapped by the implementation phase. The process of concurrent engineering is shown in Figure 48.

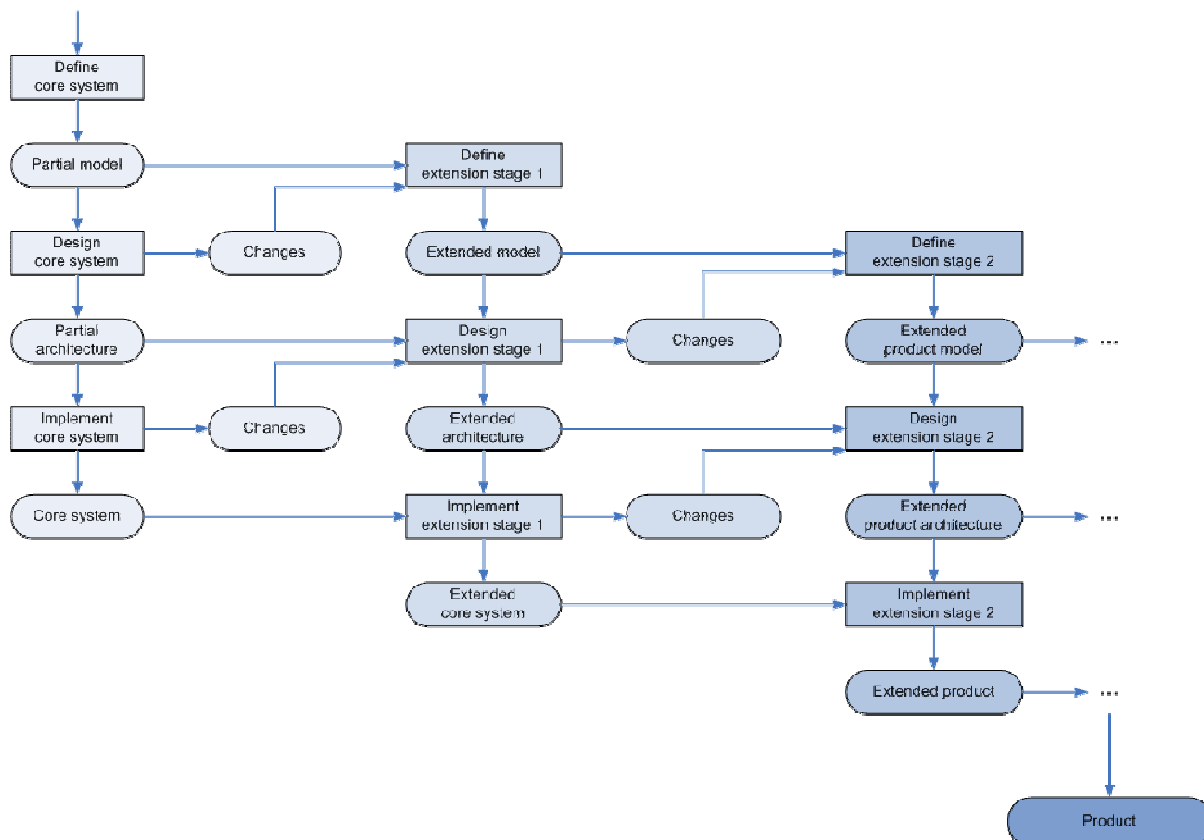


Figure 48: Concurrent engineering process [Balz98]

The goal of production time minimization and consequent development parallelization holds risks. If some instruments had been already built as the inevitable changes came up, the efforts to produce these instruments thus had been made in vain. This means that software development under the concurrent engineering paradigm should always consider whether it is financially feasible to make design decisions which may be changed in future [Balz98]. So the motto of this software development model sounds “right-the-first-time” unlike the motto of the prototyping model “redo-until-right”.

The advantages of the concurrent engineering software development process are:

- early problem detection and elimination. If all of the involved development departments participated in a decision process, it is likely that the risks would be minimized through early measuring of the alternatives.
- optimal time usage. Under assumption that everything goes right, this development model minimizes the development time, thus yielding shortest time-to-market.

The disadvantages of the concurrent engineering software development process are the following:

-
- unclear whether the goal “right the first time” can be achieved. As already stated, the parallel development may hold unpredictable risks that later lead to reengineering of the product and possible scrapping of the developed instruments.
 - risk if the decision made too late. If one requirements statement is made that confuses previously made design decision, then additional iterations needed.
 - high planning and personnel costs. If all of the involved development departments are taking part in the development as a team, then it is difficult to coordinate all of them at a time. This may augment the planning efforts for early error and problem anticipation, which then results in higher costs.

As a consequence, the concurrent engineering model can be applied in projects where it is possible to join all of the development departments in one team that conducts the development in a corporate way. Also, when it is likely that the requirements won't change in the future, it is feasible to take the concurrent engineering model as a development basis. Otherwise, if it is predictable that at least some of the requirements would change in the future and it can't be predicted in which way, then the concurrent engineering is not the best alternative. If the communication between departments is complicated, then it is not a good idea to develop the software product in a parallel way, because it demands a close connection between development groups in order to achieve its goal of minimal time-to-market.

5.1.9 Spiral development

The spiral software development model introduces as an alternative way of connecting the development activities. Rather than sequentially combining them, the spiral model represents them as a spiral-formed cyclic diagram (see Figure 49). Each next outermost loop of a spiral stands for a more detailed system model, beginning with requirements analysis and statement at the innermost loop and ending with specification and implementation with testing at the outermost loop. The area of the spiral shows the accumulated costs of the development. The angle of the spiral represents the development progress. This development model was introduced by BOEHM in 1988 and explicitly stresses the recognition of risk during software development [Boeh88].

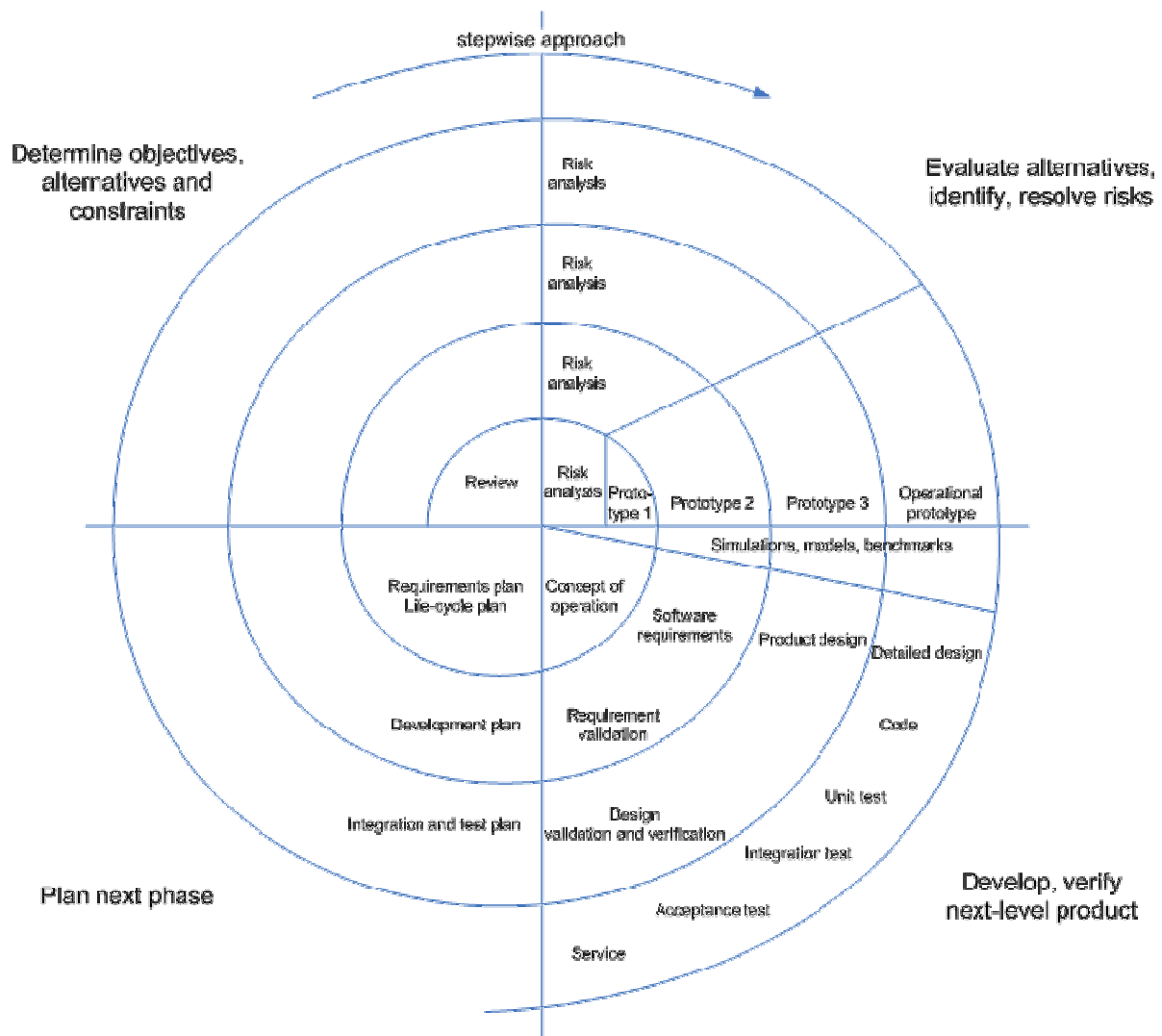


Figure 49: Spiral development process [Somm07, Balz98]

Each loop of a development spiral contains four steps of the according system stage elaboration [Balz98, Somm07]:

1. Objective setting – firstly, the goals of this spiral stage are stated. Alternative solutions are considered and the risks are outlined. During alternative solutions consideration the constraints on each solution are identified.
2. Risk assessment and reduction – secondly, each of the outlined risk kinds undergoes a detailed analysis. The possible, in the first step detected, alternative solutions are evaluated in order to identify whether there are risks that interfere with each solution. In case there are such risk kinds, appropriate strategies have to be elaborated to reduce the risk.
3. Development and validation – thirdly, after solution and risk evaluation, the software process model can finally be chosen. If it makes sense, a combination of different models can be taken due to risk minimization [Balz98, p.129].
4. Planning – finally, the previous steps and the project as a whole are reviewed. In case the decision is made to enter into commitment of continue the development with the next spiral loop, the according plans are drawn up. This ends the current loop and goes over to the next one.

The advantages of the spiral development model are as follows:

- periodical checking of the project progress. This can serve a good point of the project review in case some requirement changes come to light.

- risks, errors and inappropriate alternatives are identified in time. The according system design decisions can thus be made in advance, before the actual implementation takes place and too much workload has to be rolled back.
- integration of other software process models. On each spiral loop it is feasible to use different software process models to develop the system. Formal transformations are suitable for the parts where safety risks are the main consideration [Somm07].
- supports software reuse. It is especially important for development of different software components where separate spiral loops are taken for each component.

The disadvantages of the spiral development model are the following:

- high management complexity. The spiral development model requires relatively moderate user interaction. At the same time, much depends on management decision, because each cycle's proceeding is planned before the previous cycle ends. In addition, risk management is difficult to maintain, because there may be problems assigning and assessing risks for different alternative strategies.
- no distinction between development and maintenance. Once one spiral loop has been completed, the information and the system model made has to be maintained. The spiral model only stresses the product development.

Therefore, the spiral development model is appropriate for the projects where risk management and reduction are the prioritized goals. Also, it suits well large-scaled projects where decisions have to be made often. It comes in handy where it is unclear whether all of the system models are going to be compliant with single software development paradigms. In this case there is a possibility to use different process models for the next system model of run concurrent spiral loops for separate software components. The spiral development model works not that well for small- and middle-sized projects because of its extensive management efforts.

5.2 Vide Procedure model

Early discovery and correction of failures in the software development process reduces the cost and effort of the whole software development [Somm07]. One innovation of VIDE is the possibility of testing VIDE models on the PIM level (c.f. D.2.1). This means that the domain user gets the ability to verify whether all necessary aspects are covered by the software. Thus the VIDE language provides a means for rapid testing and user feedback. This strength of the language will be used in the VIDE software development procedure model. Additionally it allows a better involvement of domain users into the software development, as business users can influence the result of the development process earlier, which increases their acceptance of the software [Somm07]. These ideas are already considered in agile development methods such as extreme programming. SOMMERVILLE states the following principles of agile development method [Somm07]:

- Customer involvement:
 - Include customers in the development process
- Incremental delivery:
 - Software is splitted into increments, which are confirmed separately by the customer
- Embrace changes:
 - Expect changing requirements
- Maintain simplicity:
 - Avoid to complex procedure models
- People not Process:
 - Developers are free in choosing their way of working.

This process means that the customer is involved at an early stage in the development process and early testing is possible by incrementally delivering parts of the software. But, agile methods are usually not focussed on models, which is the main issue of an MDA project like VIDE. Additionally SOMMERVILLE argues that agile methods are not suitable for security critical applications and for large scale development. But security aspects are especially important for business software, as they often deal with financial data or sensitive customer and product data. Therefore agile methods bring important advantages (early customer involvement and incremental delivery) but can't be used without adoption of an MDA-based software development procedure for data-

intensive business applications. Therefore, the VIDE software development procedure model is based on a structured procedure model but includes the advantages of agile methods. Especially as incremental delivery is an important issue the spiral model is used as a base for the VIDE procedure model. The roles involved in the development process have already been defined in deliverable D.1.1. Five roles have been identified: the domain user, the business analyst, the VIDE analyst/designer, the VIDE programmer and the VIDE architect. As Figure 50 shows the VIDE programmer and architect serve also as testers for the developed VIDE application.

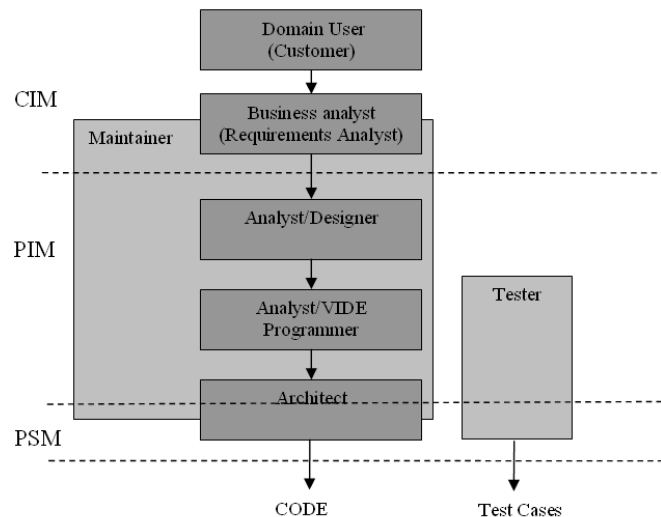


Figure 50: VIDE users in relation to MDA levels (c.f. D.1.1)

The VIDE procedure model consists of five phases. It is closely linked to the transformation process starting from requirement at the beginning and ending with PIM model, which will be described in deliverable D.5.1. The phases are executed sequentially. After all five phases have been executed one cycle of the spiral is completed and the development procedure starts with the first phase again until the software product is finished. This allows the usage and testing of parts of the whole software product. Because VIDE offers the orchestration of different applications, different parts of a software application can be implemented as separate VIDE applications which are later on orchestrated in order to achieve the full functionality of the whole software system. Each of these applications can be regarded as a module, which can be implemented in one cycle of the spiral procedure model. This modularization offers the chance to reuse existing VIDE applications and to integrate exciting legacy applications, if they provide a proper interface allowing their invocation by a workflow management system. The five phases (c.f. figure 50) are:

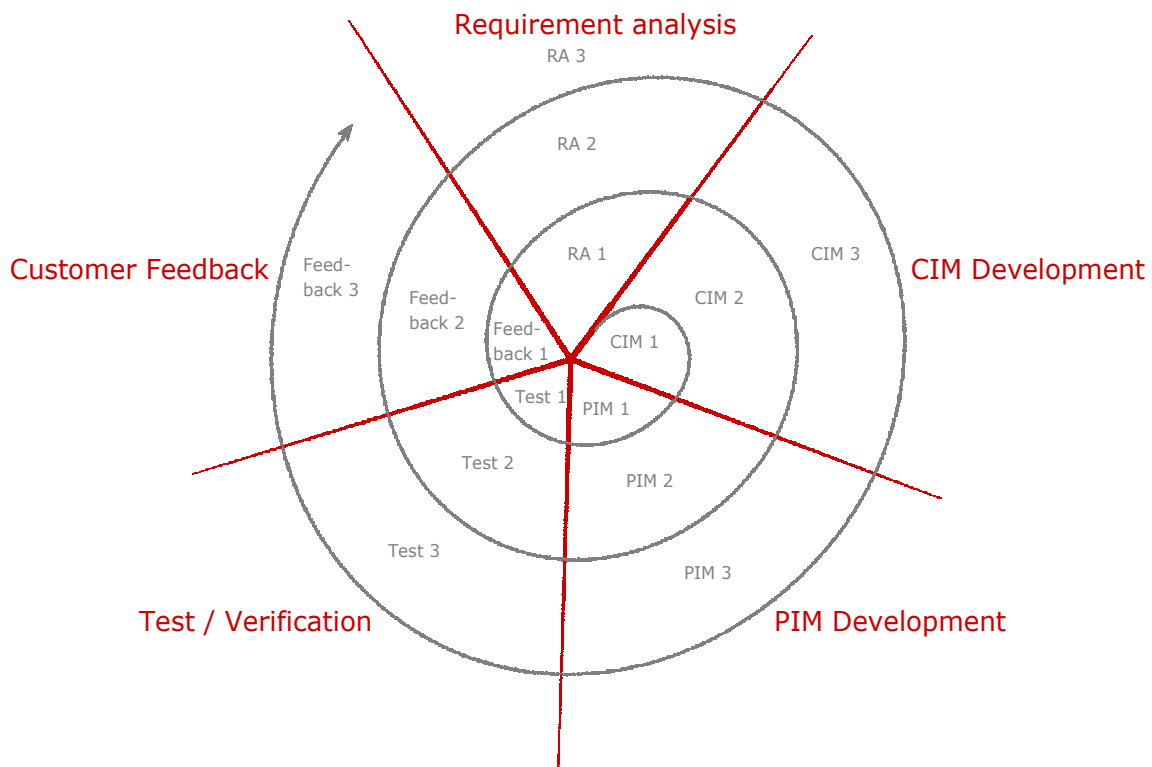


Figure 51: VIDE Procedure Model

Requirement analysis

The first major phase is the requirements analysis. This phase is conducted by the domain user and supported by the business analyst. The aim of this phase is to identify the goals that should be achieved with the software development and the requirements that derive from them. For the definition of requirements the domain user, who is usually a future user of the developed software, specifies business needs in a process centric way. This means the business processes that should be supported or executed by the software are depicted. Additionally questions are asked. Such as who will be using the software? When will it be used? What data are involved? The requirements resulting from this phase are goals, business process descriptions and requirements in natural language. Additionally existing forms that are relevant for the supported process are gathered. Here paper based forms and electronic forms, like PDF-documents or spreadsheets are collected. The role of the business analyst in this phase is to discover inconsistencies and contradictions in the gathered requirements and goals, to ask for missing pieces and to combine the requirements of different stakeholders of the software systems.

CIM modelling

The results of the requirements analysis are used to create CIM models. CIM models are semi-formal models which provide more structure than natural language but are still not executable. The CIM models are created by the business analyst and the domain user. The involvement of the domain user depends on his modelling skill and experience. But at least a validation of the business facts depicted in the CIM models should be done by the domain user. The creation of the CIM models usually starts with the modelling of the business process view. Here activities and the control flow between them are modelled. In this stage an important design decision is taken. The business analyst and the domain users decide which functionality is clustered to one application and describe the control flow between these applications for the workflow management system. The business processes which describe the refined activities and control flow of each application are further extended. Therefore other elements of the process view like groups, constraint business rules, etc. can be introduced. Then, based on the documents, forms and process description from the requirements analysis, the data objects that are used/edited or created are attached to each activity. They are later refined in the data view of the CIM level language by describing their attributes and their relations. Furthermore roles of employees who execute the activities are attached. Then the relation of these roles, their relation to departments, etc. is described in the

organisational view of the VCLL. In order to keep the relation and to allow changes in the next cycle of the spiral development procedure the relation of CIM level model elements and the artefacts of the requirements analysis they derive from are documented.

PIM Modelling

In this phase the CIM models are used to create the PIM level model. The creation of the PIM models is done by the VIDE designer and the VIDE programmer and supported by the business analyst. While the first two roles create the PIM models, the function of the business analyst is to provide information about the CIM models, if misunderstandings should occur. The PIM models are the highest executable level. Again the relation between CIM and PIM model elements is documented. This allows the identification of the impact of changes to both models in the next cycle of the development procedure.

Testing

After the PIM model is completed the tester (VIDE programmer and architect) verifies the technical functionality of the VIDE application. Therefore the functionality of the methods is tested and corrected if necessary. Additionally the transformability to target platforms is tested. This ensures that the completed systems can be used in the productive environment.

Customer Feedback

The technical running system is afterwards evaluated by the domain expert and the business analyst. In this phase the requirements that have been gathered in the first phase of the development procedure are checked. Additionally the business processes and data modelled on the CIM level can be used as test cases. These tests can have four possible consequences. Firstly, the application can meet all requirements and in this case no changes are needed. Secondly, the software reveals that the requirements are incomplete or they have changed by external influences. In this case the requirements have to be corrected and all models on the levels below have to be adapted. Thirdly, the requirements are correct but errors occurred during the creation of CIM models. In this case the CIM models have to be corrected and the changes propagated to the PIM level. Finally the creation of the PIM model based on a correct CIM level model leads to changes of the PIM models.

Every need for changes that occurs in this phase leads to a new development cycle in the VIDE procedure model. Furthermore if one application, which is only one part of the whole software system, is finished the next application leads to a new development cycle.

References

- [AaHe02] Aalst, W.M.P. van der, Hee, K.M. van (2002). Workflow Management: Models, Methods, and Systems. Cambridge, MA: MIT (2002).
- [Aals99] van der Aalst, W.M.P.: Formalization and Verification of Event-driven Process Chains. Information and Software Technology, 41(10):639-650, 1999.
- [Balz98] Balzert H.: Lehrbuch der Software-Technik. Berlin: Spektrum. (Ch. 3), 1998
- [Bax+98] Baxter, I. D.; Yahin, A; Moura, L; Sant'Anna, M, and Bier, L.: Clone Detection Using Abstract Syntax Trees. In International Conference on Software Maintenance, pp. 368-378. IEEE Computer Society Press, 1998.
- [Be02] Becker, J.: Konfigurative Referenzmodellierung. In: Becker, J.; Knackstedt, R. (Ed.): Wissensmanagement mit Referenzmodellen. Springer, Heidelberg, 2002, pp. 25-144.
- [Beck00] Beck, K.: Extreme Programming explained. Boston: Addison-Wesley. (Chs. 4, 17, 25, 26), 1998.
- [Bell02] Bellon S.: Detection of software clones – tool comparison experiment. URL <http://www.bauhaus-stuttgart.de/clones/>. Official homepage of the experiment, 2002.
- [Beni56] Benington, H.D.: Production of Large Computer Programs. Proceedings, Symposium on Advanced Computer Programs for Digital Computers sponsored by ONR. Republished in Annals of the History of Computing, Oct. 1956, pp. 350-361.
- [BeZu99] Becker, J.; zur Mühlen, M.: Rocks, Stones and Sand - Zur Granularität von Komponenten in Workflowmanagementsystemen. In: Information Management & Consulting, 17 (1999) 2, p. 57-67.
- [BFS00] Bisson, B., Folk, V. and Smith, M.E. (2000). Case study: How to do a business process improvement. Quality and Participation, 23 (1), pp. 58-63.
- [Boeh79] Boehm, B.W.: Guidelines for verifying and validating software requirements and design specifications EURO IFIP 79, North Holland 1979, pp 711-719.
- [Boeh88] Boehm, B.W.: A spiral model of software development and enhancement. IEEE Computer, 21 (5), 61-72. (Chs. 4, 5), 1988.
- [BRG06] Business Rules Group (Ed.): Business Rules Manifesto : The Principles of Rule Independence. <http://www.businessrulesgroup.org/brmanifesto.htm>, 2006
- [BZG02] Becker, J.; zur Muehlen, M.; Gille, M.: Workflow Application Architectures: Classification and Characteristics of Workflow-based Information Systems. In: Fischer, Layna (Hrsg.): Workflow Handbook 2002. Future Strategies. Lighthouse Point, FL 2002.
- [CA01] Cockburn, A., Writing effective Use cases. 2001: Addison-Wesley.
- [CB05] Badica, C, et.al. Integrating Role Activity Diagrams and Hybrid IDEF for Business Process Modelling using MDA; 6th International Workshop on Symbolic and Numeric Algorithms for Scientific Computing, 2005
- [Ch76] Chen, P.: The Entity-Relationship-Model : Toward a Unified View of Data. In: ACM Transactions on Database Systems 1 (1976), No. 1, pp. 9-36.
- [Ders99] Derszteler, Gérard: Prozeßmanagement auf Basis von Workflow-Systemen : ein integrierter Ansatz zur Modellierung, Steuerung und Überwachung von Geschäftsprozessen. Zugl. Berlin, Techn. Univ., Diss., 1999, Lohmar 2000.)
- [Deru96] Derungs, M.: Vom Geschäftsprozess zum Workflow. In: Österle, H. /Vogler, P. (Ed.): Praxis des Workflow-Managements, St. Gallen, 1996, p. 123-146.
- [DHLS96] Deiters, W.; Herrmann, T., Löffler, T.; Striemer, R: Identifikation, Klassifikation und Unterstützung semi-strukturierter Prozesse in prozeßorientierten Telekooperationssystemen. In:

- Krcmar, H.; Kewe, H.; Schwabe, G. (Ed.): Herausforderung Telekooperation, Berlin 1996, p. 261-274.
- [Endl04] Endl, R.: Regelbasierte Entwicklung betrieblicher Informationssysteme. (Wirtschaftsinformatik; 45). Eul, Lohmar et al., 2004.
- [GA98] Abeysinghe, G., et al., RolEnact: Enactable Models of Business Processes; Information and software Technology Journal, 1998. 40(3).
- [Giag01] Giaglis, G.M.: A Taxonomy of Business Process Modeling and Information Systems Modeling Techniques. International Journal of Flexible Manufacturing Systems;13(2) 2001, pp.209-228.
- [GKC99] D. Giannakopoulou, J. Kramer, and S.-C. Cheung. Behaviour analysis of distributed systems using the Tracta approach. Automated Soft. Eng., 6(1):7–35, Jan. 1999.
- [Gut+06] Gutierrez et al.: Using tactical and operational factors to assess strategic alignment. In European and Mediterranean Conference on Information Systems (EMCIS) Costa Blanca, Alicante, Spain 2006, p.7.
- [HaHe00] Hay, D.; Healy, K.: Defining Business Rules – What Are They Really. Final Report., 2000.
- [Haus96] Hauser, Christof: Marktorientierte Bewertung von Unternehmensprozessen. Zugl.: St. Gallen, Univ., Diss., 1996, Bergisch Gladbach 1996.
- [Heil94] Heilmann, H.: Workflow Management: Integration von Organisation und Informationsverarbeitung. In: HMD 31 (1994) 176. 1994, p. 8 - 21.
- [Jung] Jung, J.: Meta-Modelling Support for a General Process Modelling Tool. In Proceedings of the 5th OOPSLA Workshop on Domain-Specific Modeling (DSM'05), Tolvanen, J.-P., Sprinkle, J., Rossi, M., (eds.), Computer Science and Information System Reports, 2005.
- [Kale00] Kalenborn, A.: Prozeßorganisation und Workflow-Management. Organisationstheoretisches Konzept und informationstechnische Umsetzung. Zugl.: Trier, Univ., Diss., 1998, Aachen, 2000.
- [Ke00] Keller, S.: Entwicklung einer Methode zur integrierten Modellierung von Strukturen und Prozessen in Produktionsunternehmen. VDI Verlag, Düsseldorf, 2000, p. 32 et seq.
- [KiMa05] Kirikova M. and Makna J. Renaissance of business process modelling, In: Information Systems Development Advances in Theory, Practice, and Education. Vasilecas, O.; Caplinskas, A.; Wojtkowski, G. Wojtkowski, W. Zupancic, J. (Eds.), Springer, 2005. p. 403-414.
- [Kind06] Kindler, E.: On the semantics of EPCs: Resolving the vicious circle. Data & Knowledge Engineering, Special Issue on Business Process Management, 56 (1), 23-40, January 2006.
- [KNS92] Keller, G.; Nüttgens, M.; Scheer, A.-W.: Semantische Prozessmodellierung auf der Grundlage "ereignisgesteuerter Prozessketten (EPK)". In: Veröffentlichungen des Instituts für Wirtschaftsinformatik, No.89, Universität des Saarlandes, Saarbrücken, 1992.
- [KoLi07] Korherr, B., List, B.: Extending the EPC and the BPMN with Business Process Goals and Performance Measures. Proceedings of the 9th ICEIS, Madeira, Portugal, ACM Press, 2007.
- [Korh07] Korhonen, J.: Enterprise BPM: A Systematic Approach. Link: <http://www.bptrends.com/publicationfiles/01-07-ART-Enterprise%20BPM%20-%20Korhonen-Final.pdf> (2007), p.3.
- [KP98] Phalp, K., et.al RolEnact: role-based enactable models of business processes; Information and Software Technology, 1998 vol. 40.
- [KSNM05] Kim, M.; Sazawal, V.; Notkin, D. and Murphy, G. C.: An empirical study of code clone genealogies. In European Software Engineering Conference and Foundations of Software Engineering, pp. 187-196. ACM Press, 2005.
- [LeRo00] Leymann, F.; Roller, D.: Produktion Workflow. Concepts and Techniques. Upper Saddle River, Prentice Hall, 2000.

-
- [LWS] H. Foster, S. Uchitel, J. Magee, J. Kramer. LTSA-WS <http://www.doc.ic.ac.uk/~hf1/phd/papers/de04.pdf>.
- [Maur96] Maurer, G.: Von der Prozeßorientierung zum Workflow Management. Teil 2: Prozeßmanagement, Workflow Management, Workflow-Management-Systeme. Arbeitspapiere WI - Nr. 10/1996. Lehrstuhl für allgemeine BWL und Wirtschaftsinformatik. Universität Mainz. Mainz 1996.
- [Ment99] Mentzas, G. N.: Coupling Object oriented and workflow modelling in Business and Information Process Reengineering. In: Information - Knowledge - System Management, 1 (1999), p. 63 - 87.
- [MeNü04] Mendling, J.; Nüttgens, M: Transformation of ARIS Markup Language to EPML, <http://wi.wu-wien.ac.at/~mendling/publications/04-EPK-AML.pdf>, 2004.
- [MeNü06] Mendling, J.; Nüttgens, M.: EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC). Inf. Syst. E-Business Management 4(3): 245-263, 2006.
- [MJ95] Jackson, M., Software Requirements & Specifications: a lexicon of practice, principles and prejudices, 1995: Addison-Wesley.
- [MO92] An introduction to Process Modelling using RADs, in IOPTCLUB Practical Process Modelling, Mountbatten Hotel, Monmouth Street, Covent Garden, London.
- [MO95] Ould, M.: Business Processes Modelling and Analysis for Reengineering and Improvement, Wiley, New York, 1995.
- [MSC] S. Uchitel, R. Chatley, J. Kramer, J. Magee. LTSA-MSC <http://chatley.com/articles/tacas03.pdf>
- [OMG03] Object Management Group: MDA Guide Version 1.0.1, June 2003, www.omg.org/docs/omg/03-06-01.pdf.
- [OMG05] Object Management Group (OMG) (Ed.): Semantics of Business Vocabulary and Business Rules Specification, <http://www.omg.org/docs/dtc/06-08-05.pdf>, 2005.
- [OMG06] Object Management Group: Business Process Modeling Notation. Version 1.0, February 6, 2006, <http://www.bpmn.org/Documents/OMG%20Final%20Adopted%20BPMN%201-0%20Spec%2006-02-01.pdf>, 2006.
- [OMG07] OMG, Unified modelling Language: Superstructure, 2007.
- [OMG07-MDA] Object Management Group: Model Driven Architecture, visited 05.06.2007, <http://www.omg.org/mda>.
- [PiRe85] Picot, A.; Reichwald, R.: Bürokommunikation: Leitsätze für den Anwender. 2. edit., München 1985.
- [Rath94] Rathgeb, M.: Einführung von Workflow-Management-Systemen. In: Hasenkamp, U.; Kirn, S.; Syring, M. (Ed.): CSCW - Computer Supported Cooperative Work). Informationssysteme für dezentralisierte Unternehmensstrukturen. Addison-Wesley. Bonn 1994, p. 45 - 66.
- [Reij03] Reijers, H. A. Design and Control of Workflow Processes. Springer-Verlag New York, Inc. 2003.
- [RöSc01] Röhricht, J.; Schlögel, C.: cBusiness – Erfolgreiche Internetstrategien durch Colloborative Business am Beispiel mySAP.com, Munich et al., Addison-Wesley, 2001.
- [SBF] N. Pryce, J. Magee. SceneBeans Framework <http://www.dse.doc.ic.ac.uk/software/SceneBeans/>.
- [Sche94] Scheer, A.-W.: CIM – Towards the Factory of the Future Springer-Verlag, Berlin et al. (, 3rd. rev. and extended. edition. 1994.
- [Schm02] Schmidt, G.: Prozessmanagement. Modelle und Methoden. 2. edit., Berlin 2002.
- [Schu02] Schulz, K.: Modellin and Architecting of Cross-Organisational Workflows. PhD thesis, University of Queensland, Australia, 2002.
- [ScOr01] Schulz, K.; Orłowska, M.: Architectural issues for cross-organisational B2B interactions. In Proceedings of the International Workshop on Distributed Dynamic Multiservice Architectures (DDMA), 2001.

-
- [She+97] Sheth, A.; Georgakopoulos, D.; Joosten, S. M. M.; Rusinkiewicz, M.; Scacchi, W.; Wileden, J.; Wolf, A. L.: Report from the NSF workshop on workflow and process automation in information systems. In: ACM SIGSOFT Software Engineering Note 1 (1997), p. 28 – 38.
- [Somm07] Sommerville, I.: Software Engineering. Eight edition. Harlow, London et al.: Addison-Wesley (Chs. 4, 17), 2007.
- [SSW06] Seel, C.; Simon, B.; Werth, D.: Business Rule-enabled Process Modelling in: Proceedings of the e-Challenges 2006 conference, Barcelona, Spain, October 25-27, 2006.
- [UMB99] Umar, A.; Missier, P. ; Bellcore, NJ: A framework for analyzing virtual enterprise infrastructure. In: Research Issues on Data Engineering: Information Technology for Virtual Enterprises, 1999, p. 4-11.
- [W3C04a] W3C: RDF/XML Syntax Specification (Revised). Feb. 2004, available at: <http://www.w3.org/TR/rdf-syntax-grammar>.
- [W3C04b] W3C: OWL Web Ontology Language Overview, Feb 2004, available at <http://www.w3.org/TR/owl-features>.
- [Whit04] White, S.: Introduction to BPMN. In: Object Management Group/ Business Process Management Initiative, <http://www.bpmn.org/Documents/Introduction%20to%20BPMN.pdf>, May 2004.
- [XPDL05] Workflow Management Coalition, Process Definition Interface – XML Process Definition Language, WMFC-TC-1025, version 2.00, October 3, 2005, available at http://www.wfmc.org/standards/docs/TC-1025_xpdl_2_2005-10-03.pdf.
- [ZhCh03] Zhou, Y.; Chen, Y.: The methodology for business process optimized design. In: Industrial Electronics Society, 2003. IECON apos;03. The 29th Annual Conference of the IEEE Vol. 2 (2003) p.1819 – 1824.