Definition and Execution of Multiple Viewpoints on Workflow Processes

University of Los Andes

Darío Correal

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor in Engineering in the School of Engineering, Department of Systems and Computing

Supervised by
Rubby Casallas

Co-supervised by
Olga Marino

Jury

Rubby Casallas - Director
Olga Marino - Co-director
Jorge Villalobos - Reviewer
Demetrio Arturo Ovalle C. - Reviewer

July 2007

to Angie and Nata

Abstract

Process-centered organizations base the achievement of their goals on the correct definition, execution, monitoring, and control of their processes. The adoption of a process-centered strategy is a complex task because a large number of processes need to be taken into account and the relationships between them need to be defined. One strategy that process-centered organizations use to manage process re-engineering is to select a particular process (target process) that is strategic for the company and use it as a central axis around which others concerns are defined.

Business processes, when expressed as cross-organizational entities, involve different stakeholders, who define particular, and sometimes unrelated, concerns into the same target process. The absence of a mechanism to define concerns into independent modules with respect to the target process model makes it difficult to maintain process definitions and to help them evolve in a coherent manner.

The main goal of this research is to propose a mechanism to support process designers with the expression of concerns in workflow processes. We base our strategy on the use of viewpoints, model driven engineering (MDE) and aspect oriented (AOP) technologies.

Particularly, we developed a domain specific aspect language, called AspectViewpoint, to support the definition of viewpoints in workflow process models in a modular and non-intrusive way. In this manner, process designers can analyze viewpoints to detect inconsistencies and dependencies between them, and then integrate them into the target process model using a model weaving engine. Finally, we transform the resulting woven process model into a particular workflow platform to execute it.

Acknowledgement

The work presented in thesis was funded in large part by COLCIENCIAS - “Instituto Colombiano Para el Desarrollo de la Ciencia y la Tecnología - Francisco José de Caldas ” and by the Natural Sciences and Engineering Research Council of Canada (NSERC-CRSNG).

I would like to thank Heinsohn Software House for its support during the first year of this thesis.

Thank you to Jeff Gray and Jane Lee, members of the SoftCom team at the University of Alabama at Birmingham (UAB), for their time and support with ECL and C-SAW.

Thank you to the members of the LICEF Research Laboratory and the Télé-Université (Teluq) for giving me the opportunity to do my internship with them. Special thanks to Gilbert Paquette director of the LORNET project.

I owe a thank you to all my friends, especially Carlos and Gregorio, who have been there, no matter the distance or the time.

Thank you to my mother for her everyday support and love. I would not be who I am without her.

Thank you to the members of the Software Construction Group (TICS_w) at the University of Los Andes. Special thanks to Jorge Jimenez and Rob Vanmeert for their collaboration and feedback on this work, and also to the members of the development team of the Cumbia project.

Thank you to Demetrio Ovalle for agreeing to sit on the jury for this thesis.

Special thanks to Jorge Villalobos for being part of the jury, and also for giving me useful feedback at crucial moments.

I owe a special thanks to my co-advisor, Olga Marino, who gave me useful hints and clues and supported me with cooperative supervision during my internship. I also thank her for being part of the jury, and for the interesting conversations about many things in life.

I would like to thank my wife, Angela, for giving me the necessary emotional support not only during the period of writing this document but also during my years of studies. She was always there when I needed her the most.

My deepest thanks go to my advisor, Rubby Casallas, who over the years provided me with guidance and encouragement. Thanks to her for making all this possible.

Contents

1	Introduction	2
1.1	Motivation	2
1.2	Problem Statement	6
1.2.1	Representational Problems	7
1.2.2	Modularization Problems	8
1.2.3	Interference Problems	8
1.2.4	Execution Problems	8
1.3	Goals of this Thesis	9
1.3.1	Specific Goals	9
1.4	Structure of this Document	10
2	Process Modeling and Execution	12
2.1	Introduction	12
2.2	The Business Process Modeling Notation	13
2.2.1	Flow Objects	13
2.2.2	Connecting Objects	17
2.2.3	Swimlanes	17
2.2.4	Artifacts	18
2.2.5	Process Execution	18
2.2.6	BPMN Example	18
2.3	The Software Process Engineering Metamodel	19
2.3.1	Work Definitions	19
2.3.2	Work Products	20
2.3.3	Pseudo-states	21
2.3.4	Model Execution	22
2.3.5	SPEM Example	22
2.4	IMS-Learning Design (IMS-LD)	22

2.4.1	Method	24
2.4.2	Activity	24
2.4.3	Role	25
2.4.4	Local Properties, Global Properties, and Conditions . .	25
2.4.5	Notifications	25
2.4.6	Process Execution	26
2.4.7	IMS-LD Example	26
2.5	The XML Process Definition Language	27
2.5.1	Workflow Application Declaration	28
2.5.2	Data	28
2.5.3	Workflow Participant Specification	28
2.5.4	Activities	29
2.5.5	Subflows	29
2.5.6	Transition Information	29
2.5.7	Process Execution	29
2.5.8	XPLD Example	29
2.6	The Business Process Execution Language	30
2.6.1	Process Definition	31
2.6.2	Partner links	31
2.6.3	Variables	31
2.6.4	Orchestration Logic	31
2.6.5	Process Execution	34
2.6.6	BPEL Example	34
2.7	Cumbia-XPM	35
2.7.1	Model Execution	36
2.7.2	Cumbia Example	36
2.8	Workflow Patterns	37
2.8.1	Basic Control Patterns	38
2.8.2	Advanced Control Patterns	39
2.8.3	Structural Control Patterns	39
2.8.4	Multiple instance patterns	41
2.8.5	State-based patterns	41
2.8.6	Cancellation	42
2.9	Comparison	42
2.10	Summary	45
3	Model Driven Engineering	46
3.1	Introduction	46

3.2	MDE Fundamental Concepts	47
3.2.1	Models	47
3.2.2	Metamodel	48
3.3	Model Driven Architecture	50
3.4	The Eclipse Modeling Framework	54
3.5	Model Integrated Computing	56
3.5.1	Domain-Specific Modeling Languages	57
3.5.2	The Generic Modeling Environment	58
3.6	Summary	59
4	Representation of Concerns	61
4.1	Introduction	61
4.2	Aspect Oriented Technologies	65
4.2.1	Join Point Models	66
4.3	Aspect-Oriented Modeling	69
4.3.1	Join Points	69
4.3.2	Pointcuts	70
4.3.3	Advices	70
4.3.4	Weaving	70
4.3.5	AOM Example	71
4.3.6	The AOM Framework	72
4.4	Domain-Specific Aspect Languages	74
4.5	Summary	76
5	The AspectViewpoint Language	78
5.1	Introduction	78
5.2	Join Points	79
5.2.1	Static join points	79
5.2.2	Dynamic join points	79
5.3	Pointcuts	80
5.4	Advices	82
5.5	Viewpoint classification	83
5.5.1	Value Viewpoints	83
5.5.2	Complex Viewpoints	85
5.6	Viewpoints and Workflow Patterns	87
5.6.1	Sequence Pattern	88
5.6.2	Parallel Split Pattern	89
5.6.3	Synchronization Pattern	90

5.6.4	Exclusive Choice Pattern	91
5.6.5	Merge Pattern	92
5.6.6	Multiple Choice Pattern	93
5.6.7	Deferred Choice Pattern	93
5.6.8	Arbitrary Cycles Pattern	94
5.7	Viewpoint Interference	95
5.7.1	Explicit Interference	97
5.7.2	Detected Interference	97
5.8	The Verification Example	98
5.9	Summary	102
6	Process Viewpoints	103
6.1	Introduction	103
6.2	The Process Metamodel	104
6.2.1	Metamodel Structure	105
6.2.2	Example of Use	107
6.3	The Viewpoint Metamodel	109
6.3.1	Metamodel Structure	109
6.3.2	Example of Use	111
6.4	The Conciliation Metamodel	113
6.4.1	The Process Weaving Engine Metamodel	113
6.4.2	Example	115
6.4.3	The Interference Resolution Metamodel	115
6.4.4	Metamodel Structure	116
6.4.5	Example	117
6.5	General Overview	117
6.6	Summary	119
7	Definition and Execution of Viewpoints	120
7.1	Introduction	120
7.2	Process Modeling	121
7.2.1	The Process Metamodel	122
7.2.2	The Process Models	124
7.3	Viewpoint Modeling	126
7.3.1	The AspectViewpoint Grammar	126
7.3.2	The Viewpoint Metamodel	128
7.3.3	Viewpoint Models	129
7.4	Viewpoint Conciliation	131

7.5	Viewpoint Weaving	135
7.5.1	Generation of Weaving Strategies	135
7.5.2	Model Weaving	136
7.6	Process Execution	136
7.6.1	Transforming BPMN to Cumbia	138
7.6.2	Executing the Cumbia Process	145
7.7	Summary	147
8	Validation and Results	149
8.1	Introduction	149
8.2	The Software Development Process Scenario	150
8.2.1	Viewpoint Definitions	151
8.2.2	Crosscutting Concerns	165
8.3	The Learning Process Scenario	168
8.3.1	Viewpoint Definitions	168
8.3.2	Crosscutting Concerns	177
8.4	Analysis of Results	179
8.4.1	Target Process Definition	179
8.4.2	Viewpoint Definition	179
8.4.3	Viewpoint Conciliation	182
8.4.4	Viewpoint Weaving	183
8.4.5	Viewpoint Execution	183
8.5	Summary	185
9	Discussion	188
9.1	Thesis Summary	188
9.2	Conclusions	189
9.3	Main Contributions	190
9.4	Future Work	191
9.4.1	Multidimensional Viewpoints	191
9.4.2	Reusability	192
9.4.3	Interference	192
9.4.4	Workflow Patterns	192
9.4.5	Dynamic Weaving	193
9.4.6	Cumbiambas	193
9.4.7	Knowledge-Intensive Processes	193
9.4.8	Formal Verification	194

A The AspectViewpoint Grammar	205
B Viewpoint Transformation	209

List of Figures

1.1	Example of a typical cross-functional process. Source [Dav93]	3
1.2	CMMi - Engineering Category Process Areas. [CKS06] pg.82	5
1.3	Example of generic practices associated to the Technical Solution process area.	6
1.4	Structure of this document	11
2.1	BPMN graphical symbols used to represent activity and gateway elements. [Gro06]	14
2.2	BPMN graphical symbols used to model starting, intermediate and ending events. [Gro06]	16
2.3	BPMN graphical symbols used to define connections, swimlanes and artifacts [Gro06].	17
2.4	Example of a BPMN process model.	19
2.5	Graphical symbols suggested by the OMG to represent SPEM Work Definition elements. [OMG05a]	20
2.6	Work products represent input, and output parameters consumed and produced by activity elements.	20
2.7	SPEM class diagram used to represent the relationships present between work definitions and work products used as input and output parameters. [OMG05a]	21
2.8	SPEM symbols used to represent work products, performers, participants and pseudo-states. [OMG05a]	22
2.9	Example of a SPEM process model.	23
2.10	The elements of the IMS-LD Metamodel. Reprinted from [IG03]	24
2.11	Definition of an IMS-LD play composed of three acts.	26
2.12	Example of an IMS-LD process model.	27
2.13	The XPDL metamodel defined by the WfMC [Coa02].	28
2.14	Example of a XPDL process definition	30

2.15	Example of a BPEL process model.	34
2.16	Example of a Cumbia process definition	37
2.17	Basic control patterns [PWR05]	39
2.18	Advanced control patterns [PWR05]	40
2.19	Structural Control Patterns [PWR05]	40
2.20	Multiple instance patterns [PWR05]	41
2.21	State-based patterns [PWR05]	42
2.22	Advanced control patterns [PWR05]	43
3.1	UML class diagram that illustrates the concepts defined by Seidewitz [Sei03]	52
3.2	Example of the MOF layer architecture [OMG03]	53
3.3	Vertical transformations between models [Gra05].	54
3.4	Horizontal transformations between models [Gra05]	55
3.5	The Ecore Metamodel [Ecl06]	56
3.6	MGA Levels. Source [SKB ⁺ 95]	58
4.1	The Crosscutting Pattern. Source [vdBCH06] [vdBC05]	62
4.2	Crosscutting concerns example	64
4.3	Implementation of the Monitoring Aspect using the AspectJ language.	67
4.4	The implementation phase model before the weaving of a cross- cutting concerns.	71
4.5	The implementation phase model after the weaving of a cross- cutting concern.	72
4.6	The AOM framework. Source [SA06a] [SA06b]	73
5.1	Static join points in the AspectViewpoint language	80
5.2	The states associated to the execution of an activity are used as dynamic join points in the AspectViewpoint language.	81
5.3	Selection methods offered by the AspectViewpoint language	82
5.4	Target process model associated to the verification viewpoint.	84
5.5	Example of a value viewpoint.	84
5.6	Example of an observational viewpoint.	86
5.7	The implementation process model after the inclusion of the verification viewpoint.	87
5.8	Example of the weaving of a sequence pattern into a process model	88

5.9	Example of the weaving of a parallel split pattern into a process model	89
5.10	Example of the weaving of a synchronization pattern into a process model	90
5.11	Example of the weaving of an exclusive choice pattern into a process model	91
5.12	Example of the weaving of a merge pattern in a process model	92
5.13	Example of the weaving of a multiple choice pattern in a process model	94
5.14	Example of the weaving of a deferred choice pattern in a process model	95
5.15	Example of the weaving of an arbitrary cycle pattern in a process model	96
5.16	Implementation process model before the review and inspection of the codification activity	98
5.17	The implementation process after the weaving of the arbitrary cycle pattern	101
5.18	The implementation process after the weaving of the <i>SrcInspection</i> activity	101
5.19	The implementation process after the weaving of the <i>SrcReview</i> activity	102
6.1	Overview of the metamodels	104
6.2	BPMN metamodel [KV06] [MMA07]	105
6.3	Example of the implementation phase of a development process model defined using the BPMN language.	108
6.4	The viewpoint metamodel	109
6.5	Target process model associated with the verification viewpoint.	111
6.6	The implementation process model after the inclusion of the verification viewpoint.	112
6.7	Object diagram associated with the verification viewpoint model.	112
6.8	The conciliation metamodel.	113
6.9	Example of a model constructed from the transformation of the verification viewpoint model.	115
6.10	Example of a resolution model constructed from the transformation of the verification viewpoint model.	118
6.11	General view of the metamodels used during the definition and execution of process viewpoints.	119

7.1	The definition and execution of multiple viewpoints in process models.	121
7.2	Partial view of the class diagram of the BPMN process metamodel developed with the GME tool.	123
7.3	Partial view of the visualization diagram of the BPMN process metamodel developed with the GME tool.	123
7.4	Partial view of the attributes diagram of the BPMN process metamodel developed with the GME tool.	124
7.5	Partial view of the constraint diagram of the BPMN process metamodel developed with the GME tool.	125
7.6	Example of a process model definition using an implementation of the BPMN specification for the GME tool.	125
7.7	Partial view of the viewpoint metamodel generated by the oAW framework. This metamodel conforms to the Ecore metamodel.	129
7.8	The verification viewpoint model defined using the plug-in AspectViewpoint editor.	130
7.9	The verification viewpoint model visualized as a XMI file. . . .	130
7.10	The conciliation metamodel defined as an UML class diagram.	131
7.11	The conciliation metamodel transformed into an Ecore metamodel using the oAW framework.	132
7.12	XMI file representing the conciliation model obtained from the transformation of the verification viewpoint model.	134
7.13	The technical solution target process before the weaving of the verification viewpoint.	137
7.14	The technical solution target process after the weaving of the verification viewpoint.	138
7.15	Transformation of a BPMN Task.	139
7.16	Transformation of a <i>while</i> type BPMN LoopTask.	139
7.17	Transformation of an <i>until</i> type BPMN LoopTask.	140
7.18	Transformation of a Multi-instance BPMN Task executed in parallel.	140
7.19	Transformation of a BPMN subprocess.	141
7.20	Transformation of a BPMN Sequence.	141
7.21	Transformation of a BPMN Event.	142
7.22	Transformation of a BPMN <i>And</i> gateway.	143
7.23	Transformation of a BPMN <i>OR</i> gateway.	143
7.24	Transformation of a BPMN <i>DataXOR</i> gateway.	144

7.25	Transformation of a BPMN <i>EventXOR</i> gateway.	144
7.26	Transformation of a BPMN <i>Complex</i> gateway.	145
7.27	Transformation of a BPMN <i>DataObject</i>	145
7.28	Fragment of a BPMN software development process model. Source [Jim07].	146
7.29	Cumbia process model obtained from the transformation of a BPMN process model. Source [Jim07].	146
7.30	Visualization of the execution of a Cumbia process using the administration portal offered by the tool. Source [Jim07]. . . .	147
7.31	Traceability of a Viewpoint. Source [Jim07].	147
8.1	The BPMN process definition of the testing scenario.	150
8.2	The target process after the weaving of the PlanningProcess viewpoint.	154
8.3	The target process after the weaving of the TrainingPeople viewpoint.	157
8.4	The target process after the weaving of the ManageConfigu- ration viewpoint.	158
8.5	The target process after the weaving of the MonitorAndCon- trol viewpoint.	161
8.6	The target process after the weaving of the EvaluateAdherence viewpoint.	163
8.7	The e-learning process scenario.	168
8.8	The e-learning process scenario after the weaving of the Learn- ingPersonalization viewpoint.	170
8.9	The e-learning process scenario after the weaving of the As- sistanceDetection viewpoint.	174
8.10	Definition of the LernerCentered viewpoint using the BPMN notation.	175
8.11	The e-learning process scenario after the weaving of the Knowl- edgeOriented viewpoint.	176
8.12	Number of <i>FlowObjects</i> contained in the target process after the modifications introduced by each concern.	181
8.13	Number of gateways and execution paths present in the target process model after the introduction of each concern	182
8.14	The target process after the weaving of the software viewpoints.	184
8.15	The target process after the weaving of the learning viewpoints.	184

8.16 The Cumbia process model obtained from the woven process model of the software scenario.	185
--	-----

List of Tables

2.1	BPEL Activities [Pau04].	32
2.2	BPEL structured activities [Pau04].	33
2.3	Workflow Basic Patterns [PWR05]	43
2.4	Advanced Synchronization and Structural Patterns [PWR05] .	44
2.5	Multiple Instance Patterns [PWR05]	44
2.6	State Based and Cancellation Patterns [PWR05]	44
3.1	Main concepts defined for a model [Sei03]	49
3.2	Main concepts defined for a metamodel [Sei03]	51
3.3	Summary of the MDE implementation alternatives presented in this chapter.	60
4.1	Dependency Matrix for the example presented in figure 4.2 .	64
4.2	Crosscutting Matrix for the example presented in figure 4.2 .	65
5.1	Value adaptations supported by the AspectViewpoint language	85
8.1	Dependency matrix for the testing scenario	166
8.2	Scattering matrix for the testing scenario	166
8.3	Tangling matrix for the testing scenario	166
8.4	Crosscutting product matrix for the testing scenario	167
8.5	Crosscutting matrix for the testing scenario	167
8.6	Dependency matrix for the learning testing scenario	177
8.7	Scattering matrix for the learning testing scenario	177
8.8	Tangling matrix for the learning testing scenario	178
8.9	Crosscutting product matrix for the learning 1testing scenario	178
8.10	Crosscutting matrix for the learning testing scenario	178

Part I

*"As you set out for Ithaka
hope your road is a long one,
full of adventure, full of discovery.
Laistrygonians, Cyclops,
angry Poseidon-don't be afraid of them:
you'll never find things like that on your way
as long as you keep your thoughts raised high,
as long as a rare excitement
stirs your spirit and your body.
Laistrygonians, Cyclops,
wild Poseidon-you won't encounter them
unless you bring them along inside your soul,
unless your soul sets them up in front of you..."*

from *Ithaka* by C.P. Cavafy

Chapter 1

Introduction

1.1 Motivation

In 1993, Michael Hammer and James Champy published their book, *Reengineering the Corporation: A Manifesto for Business Revolution*. In it, they stated that “corporations must undertake nothing less than a radical re-invention of how they do their work.” [HC93]. This proposal originated as a recommendation to change the structural organization of most companies, which, until the early 90s, followed a pyramid shape, based on dividing complex tasks into smaller and simpler ones that are executed by specialized workers[HC93].

Hammer and Champy proposed a change in how business is done, oriented by business processes, which they defined as “the collection of activities that takes one or more kinds of inputs and creates an output that is of value to the customer.” In their proposal, business processes are applied across the company, changing functional departments into process teams [HC93].

In the same line as Hammer and Champy, Thomas Davenport proposed, in *Process Innovation: Reengineering Work through Information Technology*, using business processes to adopt a more cross-organizational structure [Dav93]. Rather than having isolated functional areas, business processes in this new structure are the responsibility of the whole organization. This idea is illustrated in figure 1.1.

Figure 1.1: Example of a typical cross-functional process. Source [Dav93]

The work of Hammer, Champy, and Davenport prompted a new interest in the study of business processes, not only in terms of their management, but also in the technology required to support them, such as languages to define the processes and workflow engines to instantiate and automate their execution.

Process-centered organizations base the achievement of their goals on the correct definition, execution, monitoring, and control of their processes. The adoption of a process-centered strategy is a complex task because a large number of processes need to be taken into account and the relationships between them need to be defined. One strategy that process-centered organizations use to manage process re-engineering is to select a particular process that is strategic for the company and use it as a central axis around which others concerns are defined.

This idea is illustrated by the Capability Maturity Model Integration (CMMI) for Development [Ins07]. This model, focused on the improvement of processes inside organizations, consists of best practices addressing development and maintenance activities that are applied to products and services [CKS06].

The CMMI model bases the software development aspect of process improvement on the definition of process areas (PAs), which are “cluster[s] of related practices in an area that, when implemented collectively, satisfy a set of goals considered important for making improvement in that area” [CKS06]. CMMI

version 1.2 proposes 22 process areas, grouped into four categories: process management, project management, engineering, and support. This classification is useful for study purposes; however, in practice, it is common to see relationships between PAs from different categories.

The CMMI model offers two different methods of implementation: continuous and staged. Process areas using the continuous representation follow a capability level classification. When the staged representation is selected, process areas are organized by maturity levels. The two representations are equivalent, and the results obtained after the complete implementation of the CMMI model are the same in either case. Companies that have to focus on the improvement of a particular process, because of strategic or inter-process dependency reasons, usually find the continuous path more suitable.

For example, consider the continuous representation of the engineering process areas presented in figure 1.2, as defined by the CMMI model [CKS06]. The category shown comprises five PAs: requirements definition (RD), technical solution (TS), verification (VER), product integration (PI), and validation (VAL). RD is in charge of the production of the requirements necessary to develop a product. TS addresses the design and development of the product and its components. VER comprises all the actions required to guarantee that the work products meet the specified requirements. PI addresses the integration of the components into complete products. Finally, the purpose of VAL is to demonstrate that the product satisfies its intended use in a production-like environment. During the whole process, the requirements management (REQM) process area, which is external to the engineering category, manages changes in the requirements as well as the new requirements created during product development, testing, and validation.

For each PA, the CMMI model defines generic and specific goals and practices. Specific goals describe the unique characteristics that must be present to satisfy a process area[CKS06]. Generic goals describe the characteristics that must be present to institutionalize the processes that implement a process area[CKS06].

The continuous representation classifies a process, at any given time, into six possible capability levels ranging from zero to five. Level zero is the incomplete level because the process does not satisfy all of the specific goals required by the model. Once the process is classified at level one, it must fulfill groups of generic goals to upgrade its capability.

Figure 1.2: CMMi - Engineering Category Process Areas. [CKS06] pg.82

According to this capability level classification system, there are two major assumptions made in the improvement of a process. First, different stakeholders are responsible for the fulfillment of different specific and generic goals. Second, the fulfillment of some generic goals is only possible through interaction with other processes in the organization.

To illustrate these ideas, consider a software development company that is working to improve and certify its development process using the continuous representation of the CMMI model. The first step is to associate the selected process with one or more of the PAs defined by the model. In this case, the technical solution process area, which considers the design and implementation activities, is chosen. Let us suppose that the selected process satisfies all the initial specific goals; therefore, it is classified at capability level one and the objective is to improve it to level two. The CMMI model defines ten generic practices that need to be implemented to reach this capability level. However, for the sake of this example, we consider only two: *evaluate adherence*, and *monitor and control* the process. The example is illustrated in figure 1.3. In this scenario, the implementation process needs to be adapted to satisfy new requirements. The *evaluate adherence* practice states that the work products produced by the process, like design documents and code, need to be reviewed and inspected. The *monitor and control* practice requires that information, such as the size of the products (KLOCs), defect

density (number of defects / KLOC), and cost of production, need to be known for some of the stakeholders of the process.

Figure 1.3: Example of generic practices associated to the Technical Solution process area.

In the example, we identify two main actors: the process designer and the process stakeholder. A process designer is in charge of the definition and modifications of the process. Process stakeholders, on the other hand, identify concerns that need to be implemented into the business process, and communicate these concerns to the process designer. Based on the interactions of these two actors, we present the following definitions, which formalize some key concepts used in this work.

Definition 1. A *target process* (TAP) is a business process selected for improvement inside an organization.

Definition 2. A *viewpoint* is the perspective from which a particular stakeholder interacts with a TAP in terms of the concerns relevant to him.

1.2 Problem Statement

In this section, we present some of the problems faced by a process designer during the improvement of a target process. We have classified these

problems into the following four categories: representation, modularization, interference, and execution.

1.2.1 Representational Problems

A process can be defined using different mechanisms, such as natural language, graphical languages, and textual languages. The use of natural language is a simple description of the activities, roles, and products involved in a process. The main drawback of this mechanism is the ambiguity of interpretation that is inherent to natural language, leading to misinterpretations appearing during the execution of the process.

Graphical languages are used to standardize and make explicit the relationships present between the components of a process, by providing a set of graphical symbols and editors to create process definitions. This alternative offers standardization and a common language, however, these languages do not offer a clear execution semantic, making their instantiation in a workflow engine difficult.

Textual languages describe in a programmatic way the control and data flows. The process is defined using a programming language which facilitates its execution. However, programs do not offer a friendly way to represent processes. Additionally, using this approach to define processes can demand considerable effort compared with using graphical languages.

The combination of these alternatives provides a hybrid approach that takes advantage of the simplicity and user-friendliness of graphical languages, and the lexical and semantic clarity of programming languages. Examples of such hybrid approaches are XPLD [Coa02], YAWL [vdAH02] and BPEL4WS [IBM03]. In these approaches, a process definition language supported by a graphical editor is used to design the processes. Process designers can then modify the process using either the graphical representation or the programming language.

The main problem with these alternatives is that they are dependent on specific technologies. The process definition is constrained to the expression power of the language and the execution capabilities of its corresponding workflow engine. Thus, it is desirable to raise the level of abstraction of the process model definition to one that is more independent of technology.

1.2.2 Modularization Problems

Business processes, when expressed as cross-organizational entities, involve different stakeholders, who define particular, and sometimes unrelated, concerns into the same target process. In the example presented in 1.3, we defined two concerns associated with the implementation process, which added new activities to the original process model. When multiple crosscutting concerns are added to the target process, the number of activities and relationships in the definition grows exponentially, potentially leading to unmanaged models.

The absence of a mechanism to define concerns into independent models with respect to the target process model makes it difficult to maintain process definitions and to help them evolve in a coherent manner.

1.2.3 Interference Problems

Interference problems arise when multiple concerns modify the target process at the same point and with incompatible actions. For example, one concern requires the deletion of an activity in a process definition, whereas another concern requires the execution of this activity to produce a monitoring metric. Most of the time, inconsistencies and deadlocks are only detected during the execution of the process, which redounds into higher correction costs. This problem results from the lack of a mechanism to analyze the dependencies and interferences in a process after the integration of multiple concerns.

1.2.4 Execution Problems

The integration of the concerns expressed by the stakeholders into the target processes produces an increment in the number of activities, work products, and control flow elements. The resulting process becomes more complex and difficult for the stakeholders and users to follow and control. For this reason, it is desirable to have a workflow engine to support the automatic execution of the process and to guide users during the execution of their activities.

However, the automated execution of business processes is usually tied to particular technological platforms, and, as a consequence, the process definition

inherits the advantages, but also the limitations, of the selected technology. Hence, the adoption of a specific platform for the execution of the process needs to be postponed as long as possible in favor of a more abstract model, which could be transformed multiple times into different cases.

1.3 Goals of this Thesis

The main objective of this thesis is to assist process designers with a mechanism that allows the definition, in a modular and non-intrusive way, of multiple crosscutting concerns in a target process, thus facilitating its execution in a workflow engine. We propose a solution based on a Model-Driven Engineering (MDE) approach to assure independence from technology and the use of domain-specific aspect languages to express stakeholders' viewpoints.

1.3.1 Specific Goals

- Perform a study of the state of the art of the most important process definition languages and workflow engines.
- Study the Model-Driven Engineering (MDE) approach and its most representative implementations.
- Design and implement a Domain-Specific Aspect Language (DSAL) [WGRM05] [GBNT01] to support the definition of viewpoints as cross-cutting concerns in process models.
- Define a set of metamodels for the definition of target process and viewpoints models, which will enable the detection of interdependencies between viewpoints and facilitate the transformation of these models.
- Define a mechanism to facilitate the interweaving of target process and viewpoint models.
- Provide the transformations required to execute target process models and viewpoints in different workflow engines.
- Develop different testing scenarios to validate the implementations and to analyze the results obtained.

1.4 Structure of this Document

This document is organized in three main parts: introduction, approach and discussion. The first part, introduction, is composed of four chapters, chapter one presents the motivation of the thesis and the specific goals. Chapter two describes different mechanism used to model business processes. Chapter three presents the state of the art in the model driven engineering (MDE) area. Chapter four presents the different approaches to the representation of concerns in software development.

The second part of the document, the Approach, is composed of four chapters. Chapter five presents AspectViewpoint, an aspect oriented modeling language for the definition of viewpoints. Chapter six explains the use process viewpoints as a mechanism to define crosscutting concerns in process models. Chapter seven presents a complete overview of the definition and execution of viewpoints on workflow processes. Finally, chapter eight presents the testing scenarios developed to validate this thesis.

The third part, the Discussion, is composed of one chapter containing the conclusions and the proposed directions for subsequent research. The figure 1.4 resumes the structure of this document.

Figure 1.4: Structure of this document

Chapter 2

Process Modeling and Execution

2.1 Introduction

Business processes are inherent to every organization; they define and guide the activities executed by the members of a company.

There have been several efforts to better understand, analyze, and organize business processes, such as the classification framework developed by the American Productivity and Quality Center (APQC) [APQ06] and the Open Process Handbook Initiative (OPHI)[MIT06].

The APQC is a taxonomy of cross-functional business processes that helps organizations identify their best practices and benchmark these practices to manage and improve their processes [APQ06]. The OPHI is held by a group of organizations dedicated to developing an on-line collection of knowledge business processes freely available to the general public [MIT06].

Other approaches, like those lead by the Business Process Management Initiative (BPMI) [BPM06], offer specifications of models to support the definition of processes. The BPMI created the Business Process Modeling Notation (BPMN) [Gro06] as a mechanism to standardize the graphical notations used to model business processes. This notation has had a great impact on the community, and languages like the XML Process Definition Language (XPDL) [Coa02] developed by the Workflow Management Coalition (WFMC) now incorporate the BPMN specification. In addition, the

BPMN is specifically oriented to be compatible with the Business Process Engineering Language (BPEL) [BBS03].

The increasing number of available process definition languages makes it difficult to select the best technology to represent and execute business processes and facilitate their intercommunication. One method of tackling this problem involves raising the level of abstraction in the process definition and concentrating on the minimum number of relevant elements.

This chapter presents an overview of some of the most important process modeling languages currently in use, with a special emphasis on BPMN. We use the workflow patterns proposed by the Workflow Patterns Initiative [WPI07] as mechanisms to evaluate these alternatives in a uniform fashion.

This chapter is organized as follows: section two presents the BPMN, section three explains the Software Process Engineering Metamodel (SPEM), section four presents the IMS-Learning Design (IMS-LD), section five explains the XML Process Definition Language (XPDL), section six introduces the Business Process Execution Language (BPEL4WS), section seven presents Cumbia Extensible Process Modeling (Cumbia-XPM), and finally, section eight evaluates these models from the viewpoint of workflow control patterns.

2.2 The Business Process Modeling Notation

The BPMI, which is part of the Object Management Group (OMG), defines and maintains the BPMN. The BPMN specification aims at standardizing business process models using a graphical language to express the activities, products, and participants involved in a process definition. The BPMN is composed of a set of graphical elements that can be grouped into four main categories: flow objects, connecting objects, swimlanes, and artifacts [Gro06].

2.2.1 Flow Objects

Flow objects comprise the large set of graphical elements defined by the BPMI and are used to model the execution of a business process. These elements are classified into three categories: activities, gateways, and events.

Activities

Activities represent particular pieces of work executed inside a process. They facilitate the definition of business process at different levels of granularity. This subdivision considers two options: atomic tasks and sub-processes. Atomic *tasks* represent work activities that are not broken down into more specialized tasks. *Sub-processes* represent activities inside a process that has been decomposed into other activities. Figure 2.1(A) presents the different graphical notations used to represent these subdivisions.

Both atomic tasks and sub-processes can be specialized into loop, multi-instance, or compensation activities. The loop attribute models cycles in a process without the use of gateways and connections. The multi-instance attribute expresses the number of instances of an activity that will be executed. Finally, the compensation attribute indicates that an activity will be executed as a response to a cancellation event triggered by another activity [Gro06].

Activity (A)	Task
		Normal	Loop	Multi-Instance	Compensation	
	Sub-Process
		Collapsed	Loop	Multi-Instance	Compensation	
Gateways (B)
		Complex	Data XOR	Event XOR	OR	AND

Figure 2.1: BPMN graphical symbols used to represent activity and gateway elements. [Gro06]

Gateways

Gateways control the execution flow of a process. Essentially, they define diverging or converging points in the process when a decision needs to be

made. If the gateway acts as a converging element, the connections arriving to the gateway are called incoming flows. If the gateway acts as a diverging element, the outgoing connections are called gates. Depending on the incoming or outgoing options, gateways are classified as exclusive (data and event-based), inclusive, parallel, or complex. These elements are presented in figure 2.1(B).

Exclusive gateways (XOR) are used to model points in the process when a single execution flow needs to be selected from a set of diverging or converging options. In the case of data-based gateways, the selection of the sequence is based on the evaluation of a condition associated with each option. When event-based gateways are used, the selection is based on an event occurring at the time of execution [Gro06].

Inclusive gateways (OR) represent diverging or converging points in which one or more alternatives can be selected based on the condition expressions associated with them. In this case, one or more alternatives can be selected.

Parallel gateways (AND) represent either points where several activities can be executed concurrently (diverging) or synchronization points where the concurrent activities need to be finished before the next activity can be executed (converging).

Complex gateways model special situations. For example, a converging point where three of five concurrent execution paths need to be completed to continue with the process, and, at the same time, a particular expression must be evaluated as true.

Events

An *event* is something that happens during the execution of a process [Gro06], such as the initialization or finalization of an activity, a property value change, or the timeout of an operation. Events are classified into three categories - starting, intermediate, and ending - depending on when they occur. Figure 2.2 shows the graphical symbols proposed by BPMN to model events.

Start events trigger the execution of a process. According to the BPMN specification, they can be grouped into six different types: none, link, message, multiple, rule and time. A *none* type indicates that the process starts

Start Events
	None	Link	Message	Multiple	Rule	Time		
End Events
	None	Link	Message	Multiple	Comp.	Cancel	Error	Terminate
Intermediate Events
	None	Link	Message	Multiple	Rule	Error

	Time	Comp.	Cancel					

Figure 2.2: BPMN graphical symbols used to model starting, intermediate and ending events. [Gro06]

independently of any event. *Link* events represent the starting of a process that is based on the finalization of other processes. *Message* events model the instantiation of one message as a result of the arrival of another. A *multiple* start event indicates that the instantiation of a process depends on multiple options. A *rule* event starts a process based on the evaluation of a condition. Finally, *time* events start the execution of a process at a specific date and time [Gro06].

Intermediate events are used between the initialization and finalization of a process. They are classified into several types. A *none* event represents, in a generic way, any event that may happen during the execution of the process. An *error* event is used either to receive notification (catch) or to inform about the presence of an error (throw). A *cancel* event can be used with other sub-processes to indicate the cancellation of a transaction within an activity. A *compensation* event initiates a compensation handling activity. The rest of the intermediate events - message, timer, link, multiple, and rule - are similar to the starting events except that they cannot initiate a process [Gro06].

End events have the same semantics as intermediate events, but are generated at the finalization of the process.

2.2.2 Connecting Objects

Connecting objects facilitate the connection between flow objects. They are classified into three types: sequence flow, message flow, and associations. Figure 2.3 (A) summarizes this classification.

Figure 2.3: BPMN graphical symbols used to define connections, swimlanes and artifacts [Gro06].

Sequence flows define the order of execution of the activities defined in a process. *Message flows* model the interchange of messages between processes. *Association flows* associate data objects with flow objects.

2.2.3 Swimlanes

The primary function of *swimlanes* is to model private business processes or Business-to-Business (B2B) situations in which several processes participate in a collaborative manner [Gro06]. Swimlanes are subdivided into pools and lanes. A *pool* represents a participant that graphically separates processes. A *lane* is a sub-partition within a pool that organizes and categorizes activities executed by different roles in the same process. Figure 2.3 (B) shows the graphical symbols offered by BPMN to represent swimlanes [Gro06].

2.2.4 Artifacts

In BPMN, artifacts do not have any direct effect on the execution of a process. They provide information about the products used or produced by activities [Gro06]. BPMN distinguishes between three types of artifacts: data objects, text annotations, and groups. Figure 2.3 (C) shows the symbols for each type of artifact.

2.2.5 Process Execution

The BPMN does not explicitly state how to execute a process model, nor does it define standard storage or interchange formats. Thus, to execute a BPMN model, it is necessary to transform it into another representation with well-defined execution semantics, such as XPLD or BPEL4WS.

2.2.6 BPMN Example

The process presented in figure 2.4 gives an example of a software development process model. In this example, the process starts with the *CreateDLD* activity, which creates the detailed design. The next activity is *Codification*, which according to the process, can be executed at two different points: after the finalization of the *CreateDLD* activity or the *UnitTesting* activity. In the first case, *Codification* implements the design as soon as it is ready. In the second case, *Codification* is executed to correct any defects in the code found by the *UnitTesting* activity. To model this situation, a *DataXor* gateway is used to define the convergence of the *UnitTesting* and *CreateDLD* activities before the execution of the *Codification* activity. After *UnitTesting*, there are two possible execution paths: if defects are present, *Codification* takes place again; if no defects are found, a baseline of the product is created by the *CreateBL* activity and the process finishes. In this example, a pool element contains all the process activities, indicating that the role developer is in charge of these activities.

Figure 2.4: Example of a BPMN process model.

2.3 The Software Process Engineering Meta-model

The Object Management Group (OMG) defines and maintains the Software Process Engineering Metamodel (SPEM). The model's main objective is to facilitate the definition of related families of software development processes, taking into account the different phases, iterations, activities, and tasks performed during the development of a product. The model supports the definition of the artifacts produced and consumed by activities during the execution of the process. SPEM also offers a mechanism to associate roles and the participants responsible for them with the process activities. The main elements of SPEM are work definitions, work products, and pseudo-states [OMG05a].

2.3.1 Work Definitions

Work definitions define pieces of work in a software development process. They can have different levels of aggregation: lifecycles, phases, Iterations, and activities. *Lifecycles* define the major level of aggregation and model the software process itself. A Lifecycle is a sequence of *Phases*, which represent groups of activities, for example, the analysis, design, and implementation of a software process. *Iterations* represent minor milestones achieved during the execution of a phase, for example, the implementation of a product can be executed in two iterations. Iterations are composed of sets of *Activities*, which represent simple pieces of work executed during the development pro-

cess. SPEM defines the composition between activities; however, it suggests stopping the composition at this point and modeling the behavior of activity elements with atomic elements called *steps*. Figure 2.5 presents the graphical symbols used by the OMG to represent these elements.

Figure 2.5: Graphical symbols suggested by the OMG to represent SPEM Work Definition elements. [OMG05a]

2.3.2 Work Products

Work products represent artifacts, such as code or design documents, used during software development processes. Work products can be used as the input parameters required by activities to start their execution and as the output products produced by activities.

In the example presented in figure 2.6, the activity *CreateUnitTest* produces the *UnitTest* artifact, which is the input parameter to the *ExecuteUnitTest* activity. In turn, this activity produces a list of the defects found during testing.

Figure 2.6: Work products represent input, and output parameters consumed and produced by activity elements.

The class diagram presented in figure 2.7 illustrates the relationship between work definitions and work products. In this figure, each work definition ele-

ment has a list of parameters, which are work product elements that represent input and output parameters.

Figure 2.7: SPEM class diagram used to represent the relationships present between work definitions and work products used as input and output parameters. [OMG05a]

2.3.3 Pseudo-states

Pseudo-states express divergence and convergence points in the process model. Divergence points can be either forks or branches. Forks model multiple and parallel execution paths; branches model multiple possible execution paths, each of which requires the evaluation of a condition to determine its execution.

Convergence points are either joins or merges. *Joins* synchronize multiple and parallel execution paths into one. *Merge* elements synchronize execution threads of a set of paths in model.

The main difference between the fork/join and branch/merge elements is that the latter require Boolean guards to determine the branch or branches that will be executed or synchronized. Figures 2.8(D) and 2.8(E) present

the symbols used to represent the fork/join and branch/merge points in a process model.

Figure 2.8: SPEM symbols used to represent work products, performers, participants and pseudo-states. [OMG05a]

2.3.4 Model Execution

The SPEM specification does not define how to execute a process model. Similar to BPMN, process definitions created using SPEM must extend the model with the appropriate semantics to execute the process.

2.3.5 SPEM Example

In this section, we use SPEM to model the example presented in section 2.2.6. Figure 2.9 shows the relationships between work definitions and work products. In this example, work products like *DLDesign* and *Code* are explicitly defined and form part of the process design and execution. In SPEM, the data flow and control flow are presented simultaneously. In addition, the role in charge of the activities, in this case the developer, is shown by a dotted arrow.

2.4 IMS-Learning Design (IMS-LD)

IMS-LD is the standard educational modeling language approved by the IMS Learning Consortium [IG03]. IMS-LD was inspired by the Educational

Figure 2.9: Example of a SPEM process model.

Modeling Language (EML)[KM04] developed at the Open University of the Netherlands. The main goal of IMS-LD is to provide a framework for the formal description of education and learning process. IMS-LD helps describe specifications such as the Learning Object Metadata (LOM) [IEE02] and the Reusable Definition of Competency or Educational Objective (RDCEO) [IMS02] for competency modeling.

To aid the modeling of a process and the implementation of supporting tools, IMS-LD breaks the metamodel down into three levels: A, B and C. Level A supports the modeling of learning activities and the defines the sequential and parallel relationships between them. Level B adds properties and conditions to the model. This addition facilitates the modification of the process model, based on the evaluation of conditions at run time. Finally, level C enriches the model with notifications sent to the participants during execution of the process. Figure 2.10 presents the main elements of the IMS-LD metamodel.

In the following sections, we explain some of the most relevant elements of the IMS-LD metamodel.

Figure 2.10: The elements of the IMS-LD Metamodel. Reprinted from [IG03]

2.4.1 Method

A process is built into IMS-LD using a theatrical metaphor. A learning situation, or *method*, corresponds to a theatrical piece, which can be performed in different ways. A play is composed of sequential acts. In each act, different roles, or characters, execute their scripts in parallel. The scripts to be executed can be either one activity or a group of structured activities (activity-structure) [Pie06].

2.4.2 Activity

IMS-LD distinguishes between two types of *activity*: learning and support. A participant involved in a learning situation may be assigned either to a learning or a supporting role, such as a staff member. Each activity is executed by a role (role-part association) using an environment composed of learning objects and services. The product, or outcome, of an activity may be integrated into the environment of other activities. A set of prerequisites and learning objectives may be tied to both the method and the activities

[Pie06].

2.4.3 Role

The *role* of learner can be modeled in a richer and more explicit way using standards such as the portfolio, or competency model. IMS-LD facilitates the description of learner information through elements called properties. As explained in the next section, properties may be local or global and, thus, they describe learner information that is passed to or from the process and information that is generated and used only inside the process, for example, to personalize the training. During execution, the data that correspond to the learner properties are called the dossier [Pie06].

2.4.4 Local Properties, Global Properties, and Conditions

Properties and *conditions* are part of the level B specification. Local properties allow for the management of information about persons, roles, and the learning process during the delivery of a learning situation (the execution of a learning design run). Global properties act as parameters that interface the execution of the process with its context (institutional norms and conditions, learner portfolios, academic program data, etc.). Property values may be changed when an act or activity is finished or when a rule is activated. The modification of a property value may activate further changes, such as the ending of an activity, the hiding or showing of an activity, the generation of a notification, or the modification of other properties. In this way, by combining rules (conditions) and properties, one can personalize the learning process and synchronize the activities in an act [IG03].

2.4.5 Notifications

Level C adds *notification*, which is a message sent automatically to a role when a condition is satisfied. This condition may be i) the finalization of an activity, an act, a play, or the whole method, ii) the modification of the value of a property, or iii) the validation of a condition. [Pie06]

2.4.6 Process Execution

The execution semantics of an IMS-LD model are well defined. First, all the *plays* defined in a run are executed in parallel. Each *play* is composed of *acts* executed in the sequence in which they were defined. Inside an *act*, different *role-parts*, associated with a set of activities, are executed in parallel.

To execute learning designs that conform to the IMS-LD specification, it is necessary to use a workflow engine, like the Reload Learning Design Player, developed at the University of Bolton. This player is based on the Coppercore Learning Design Engine developed at the Open University of the Netherlands [OUN07].

2.4.7 IMS-LD Example

In this section, we model a simple learning scenario. Figure 2.11 presents the definition of a *play* element that is composed of three acts executed in sequence.

Figure 2.11: Definition of an IMS-LD play composed of three acts.

Figure 2.12 shows the structure of an act called *Part I*. In this model, two actors participate in the learning process: the learner and the tutor. The learner is in charge of executing the learning activity *ReadMaterial* and then the activity *ProduceText*. The product is used by the *CorrectText* activity, which is executed by the tutor. When the corrections are ready, they are sent back to the learner, who makes the corrections in the *CorrectText* activity. When this activity is finished, a support activity, *AnalyzeVideo*, is executed. The process ends with the learning activity *evaluation*.

Figure 2.12: Example of an IMS-LD process model.

2.5 The XML Process Definition Language

The workflow reference model [WfM95] is defined and maintained by the Workflow Management Coalition (WfMC). This model is described in terms of a central component, a system engine, or workflow enactment service, and five interfaces between this engine and the other components of the system. Interface one makes the link between the engine and the process definition tools. It is through this interface that a process description, an XPDL [Coa02] document, is passed to the engine to be executed. Interface two makes the link with the client applications. Interface three helps connect and launch the tools that are used in the different activities of the workflow. Interface four allows for the interaction between different enactment engines. Finally, interface five describes the process audit and management services [Coa02].

We are particularly interested in the study of interface one, that is, the XML Process Definition Language (XPDL) [Coa02]. The goals of XPDL are i) to provide a business process representation language that allows for the automated manipulation of modeling, instantiation, simulation, visualization, audit, and documentation; ii) to distinguish clearly between the process definition and the process execution; and iii) to provide an interface between process modeling and execution in order to link modeling tools to execution engines [Coa02]. XPDL describes the high-level entities that appear in a process definition, illustrated in figure 2.13).

In XPDL, a process is modeled using a workflow process definition. Inside this definition, process designers declare the activities, participants, and products contained in the process. In the following sections, we explain the

Figure 2.13: The XPDL metamodel defined by the WfMC [Coa02].

main elements of the model.

2.5.1 Workflow Application Declaration

A process can declare external applications that will be invoked during its execution.

2.5.2 Data

XPDL specifies two types of *data*: workflow relevant data and system data. Workflow relevant data are generated during the execution of a process, whereas system data describe the execution context. Both types can be used in activities and during the evaluation of condition expressions associated with transition elements.

2.5.3 Workflow Participant Specification

This section describes potential participants (human or machine agents) in the different activities of a process during its execution.

2.5.4 Activities

Activities represent portions of work to be performed in the workflow process. They can be classified into three categories: generic, route, and block.

- *Generic Activities* represent indivisible pieces of work in the process definition.
- *Route Activities* model the control flow of the process and synchronize other activities.
- *Block Activities* define sets of activities and transitions.

2.5.5 Subflows

Subflows define a new process definition, which can be invoked in either a synchronous or an asynchronous way.

2.5.6 Transition Information

A *transition* defines a link between two activities, and may contain a condition to determine whether or not the transition is executed.

2.5.7 Process Execution

XPDL process models can be executed using workflow engines such as the WfMOpen engine[WFM07] or the Enhydra Shark [Enh07a]. WfMOpen is a J2EE workflow component based on a set of Java interfaces. Enhydra Shark is an extendable and embeddable Java Open Source workflow engine framework.

2.5.8 XPLD Example

In this section, we use XPDL to define the software process example presented in previous sections. The process was edited using the Enhydra JaWE editor [Enh07b].

In an XPDL process model, converging and diverging points are modeled with route activities. In figure 2.14, the activities *Select* and *Defects* act as gateways to control the execution flow. In XPDL, a *Swimlane* is used to bind the execution of sets of activities to a particular role.

Figure 2.14: Example of a XPDL process definition

2.6 The Business Process Execution Language

The Business Process Execution Language for Web Services (BPEL4WS) is an XML programming language created to support the modeling of business processes using a combination of web services. A BPEL process can be seen as a control flow diagram, where execution is divided into services that can execute one or more activities. An activity can invoke operations of other services and receive messages for external sources [BBS03].

The execution of a process can be defined using primitives offered by the language, such as sequence, which defines an execution order; switch, for conditional routing; while, for looping; pick, for race conditions based on timing or external triggers; flow, for parallel routing; and scope, for grouping activities [BBS03].

A BPEL process is subdivided into four main sections: process definition, partner links, variables, and orchestration logic.

2.6.1 Process Definition

A BPEL *process* is an XML program. Even if it has several graphical editors, the program can be edited using a plain text editor. As with other XML files, the process definition elements declare the name spaces used and define the name of the BPEL process.

2.6.2 Partner links

As mentioned above, BPEL is a language specialized for the orchestration of web services, called partner links. A partner link definition has four attributes: *name*, *partnerLinkType*, *myRole*, and *partnerRole*. The *name* attribute identifies the participant service in the process definition. *PartnerLinkType* specifies the service offered. The *myRole* attribute describes the service implemented. *PartnerRole* defines the web service that calls this partner.

2.6.3 Variables

The variables define the list of messages and XML documents used within the BPEL process.

2.6.4 Orchestration Logic

Orchestration Logic defines the activities of the process and the message flow between the partners. We can identify two types of element: activities and structured activities.

Table 2.1 summarizes the activities supported by BPEL[Pau04]. The graphical symbols associated with each element were taken from the Oracle JDeveloper tool [Ora07a].

Table 2.2 summarizes the structured activities supported by BPEL[Pau04]. The graphical symbols associated with each element were taken from the Oracle JDeveloper tool [Ora07a].

BPEL Construct	Graphical Symbol	Semantics
assign		Copy data from one variable to another
compensate		Defines a compensation handler
invoke		Used by a client calling the Web Service interface of a partner
receive		Blocking instruction used to receive services from other participants
reply		Send a response as a result of a receive activity
terminate		Unconditionally terminates a workflow
throw		Starts an exception handling
transform		Transforms the type of variable
wait		Specifies a delay for a fixed period of time
empty		Void operation

Table 2.1: BPEL Activities [Pau04].

BPEL Construct	Graphical Symbol	Semantics
sequence	N/A	One or more activities performed sequentially
switch		Based on a condition exactly one operation gets selected and executed
while		Loop construct
pick		Awaits the occurrence of one of a set of events and then performs the activity associated
flow		Defines statements which are to be processed in parallel
scope		Defines an area within a workflow

Table 2.2: BPEL structured activities [Pau04].

2.6.5 Process Execution

The BPEL4WS specification offers clear execution semantics and a well-defined storage and interchange formats. These characteristics facilitate the execution of processes. There are several products that support the definition and execution of BPEL process models like the Oracle BPEL Process Manager [Ora07b] and JDeveloper.

The semantics associated with the execution of a BPEL process require that all the participants are web services. The invocation of these services can be done in either a synchronous or asynchronous way. Additionally, the passing of messages between partners and the use of events are supported by the execution engine.

2.6.6 BPEL Example

Figure 2.15 presents the BPEL version of the software process example. In this case, the process activities are invocations of services offered by other partners. Contrary to the XPDL and BPMN models, the cycle involving the *Codification* activity after *UnitTesting* is defined using a *While* structured activity. This is because the BPEL language does not support the definition of arbitrary cycles using XOR gateways.

Figure 2.15: Example of a BPEL process model.

2.7 Cumbia-XPM

Cumbia Extensible Process Modeling (Cumbia-XPM) is one of the components of the Cumbia project [Vil06] developed by the Software Construction Groups at Los Andes University. Cumbia-XPM models intensive control scenarios, such as workflow engines, e-learning systems, orchestration platforms, and collaboration tools, by providing a metamodel, three types of extension mechanisms, and a reflective Application Programming Interface (API) [Vil06].

Cumbia-XPM offers a minimal set of clearly defined elements to model the static and dynamic definitions of a process. Cumbia-XPM works from the concept of open objects, which are composition elements based on the object-oriented paradigm. The model is extended to facilitate the composition, coordination, and monitoring of these elements[oLA07].

The Cumbia-XPM metamodel is composed of a handful of specific open objects designed to express different aspects of the control component of an application. However, these elements are not expected to be used in their basic form; they offer the required capabilities for composition and coordination, but the specific semantics needed in each context must be inserted using one of the extensibility mechanisms that open objects offer [oLA07].

Cumbia can be seen as a virtual execution machine. In the same Cumbia engine, it is possible to execute BPEL, XPDL, and IMS-LD processes. To achieve this objective, it is necessary to develop a mechanism to transform each of these models into the Cumbia model. Additionally, process models defined in SPEM and BPMN can be enriched with execution semantics provided by the Cumbia-XPM metamodel that are to be executed natively in the Cumbia workflow engine.

Some of the most important elements of the Cumbia-XPM metamodel [Vil06] are

- *Activity*: An element, composed of two sets of ports and a work space, that represents an atomic step in a process definition. The first set of ports receives data to be used by the activity, and the second set sends out data produced by the activity.
- *MultiActivity*: An element that represents a multi-instance activity.

- *Port*: An element defining entry or exit points for processes and activities, and representing the connection points of data flows.
- *Workspace*: An element that allows communication between the activity and the external world of the process.
- *Dataflow*: An element that communicates activities and processes by connecting their corresponding exit and entry ports.
- *StateMachine*: A set of stages that define the behavior of a given element, which goes from one stage to another through a transition caused by an event. Element refers to any of the components of the metamodel presented above.

2.7.1 Model Execution

Currently, an implementation of the Cumbia specification, called JCumbia, is available. The Cumbia-XPM engine forms part of this implementation, and it allows for the instantiation and execution of process models. The platform also offers monitoring consoles to control the instances in execution [Vil06].

As mentioned above, the execution of a Cumbia process is based on the coordination of multiple state machines, by passing messages between them. Each element of the model has a state machine that represents its execution. Thus, executing a process in a Cumbia engine means starting the first state machine. This event triggers the initialization of other state machines, which execute in a coordinated way. In a Cumbia model, gateways are not present, hence, diverging and converging points are modeled using data flow.

A Cumbia process is modeled as a special activity. Each activity defined in a Cumbia process can communicate with other activities using data flows between pairs of ports. An activity starts its execution when the data required by its ports arrive.

2.7.2 Cumbia Example

Figure 2.16 shows the Cumbia version of the software development example used in the previous sections.

Figure 2.16: Example of a Cumbia process definition

The *Codification* activity defines two input ports, one for the reception of the design product from the *CreateDLD* activity and other for the reception of the defect product from the *UnitTesting* activity. This scenario illustrates the use of a data flow to model an *XOR* gateway.

2.8 Workflow Patterns

Since the publication of *Design Patterns* by the GoF [GHJV95], the term *pattern* has been used widely in different technology contexts. The main purpose of a pattern is to identify and provide clear documentation for successful solutions to well-known problems.

The term *workflow pattern* was first used by A.D. Van der Aalst after several years of research in the field of workflow process designs and execution. In his first publication [AHKB03], Van der Aalst identified 20 workflow patterns, which defined control flow situations encountered in process designs and their corresponding solutions. In the most recent review of workflow control patterns, 23 new patterns were added, for a total of 43.

Parallel to workflow control patterns, two complementary works proposed a new classification of patterns related to data [RtHEvdA05] and resources [RvdAtHE05] of process definition. Workflow data patterns depict situations in which the execution of the process is based on the flow of the data rather than the execution control of the activities. Workflow resource patterns are

based on the type of work assigned to the performers of the activities during the modeling and execution of a process. A total of 40 workflow data patterns and 43 workflow resource patterns have been proposed.

An added value of workflow patterns is their use as a unit of measurement for different process definition languages and workflow engines. We believe that this is one of their most important uses, particularly in the field of workflow control patterns. Nevertheless, producing a complete classification of all the process modeling languages, considering there are a total of 126 workflow patterns, is a cumbersome task.

In this section, we use the initial 20 workflow control patterns to compare the different process modeling languages presented in the first part of this chapter. We believe that this subset is powerful enough to give an idea of the main capabilities of each of these languages and the differences between them.

We present the workflow control patterns using the graphical syntax defined by BPMN and a textual explanation describing their purpose using the classification defined in [AHKB03]: basic patterns, advanced patterns, structural patterns, multi-instance patterns, state-based patterns, and cancellation patterns.

2.8.1 Basic Control Patterns

Basic control patterns define the elementary aspects of control flow. This category includes sequence, parallel split, synchronization, exclusive choice, and simple merge patterns. Figure 2.17 presents the BPMN definition of each [PWR05].

Sequence patterns define an ordered sequence of activities. Parallel split patterns (AndSplit) convert one thread of control into multiple parallel threads of execution. Synchronization patterns merge multiple threads of control (AndJoin). Exclusive choice patterns (XORSplit) define a decision point in the process where one or more branches of execution are selected. Simple merge patterns model a point in the process where two or more execution branches arrive but there is no synchronization (XORJoin) [PWR05].

Figure 2.17: Basic control patterns [PWR05]

2.8.2 Advanced Control Patterns

Advanced control patterns model complex branching and synchronization scenarios. Figure 2.18 presents the BPMN definition of this category.

Multiple-choice patterns model a point in the process where one or more parallel threads of execution are selected from a set of options (ORSplit). Synchronizing merge patterns model the convergence of two or more execution threads into one (ORJoin). Multiple merge patterns represent a point in the process similar to the merge pattern (XORSplit), but more than one execution thread can be active simultaneously. Discriminator patterns model a point in the process where two or more execution threads converge but only one is used to trigger the execution of the next activity [PWR05].

2.8.3 Structural Control Patterns

Structural control patterns model two scenarios: arbitrary cycles patterns, which model multiple entry and exit point loops, and implicit termination patterns, which model a situation in which the process must be finished if there are no possible execution paths. Figure 2.19 presents the BPMN definition of this group of patterns [PWR05].

Figure 2.18: Advanced control patterns [PWR05]

Figure 2.19: Structural Control Patterns [PWR05]

2.8.4 Multiple instance patterns

Multiple instance patterns represent situations in which more than one instance of an activity is executed 2.20.

Figure 2.20: Multiple instance patterns [PWR05]

Multi-instance without synchronization patterns represent multiple instances of an activity in a process. Multi-instance with a priori design time knowledge patterns represent situations in which the number of instances of an activity is known at design time. Multi-instance with a priori runtime knowledge patterns differ in that the number of instances is only known at runtime. In multi-instance without a priori runtime knowledge, the number of instances of the activity is variable at runtime [PWR05].

2.8.5 State-based patterns

State-based patterns model processes whose execution is determined by the state of the process at runtime. Figure 2.21 illustrates the BPMN definition of these patterns.

Deferred choice patterns model a point in the process where the selection of the execution thread is resolved at the last moment by the workflow engine. The decision is notified using an event. Interleaved parallel routing patterns model a set of activities that can be executed in an arbitrary order. Milestone

Figure 2.21: State-based patterns [PWR05]

patterns model a point where the execution of an activity depends on the execution state of the process [PWR05].

2.8.6 Cancellation

Cancellation patterns model the termination of the process. Cancel activity patterns model situations in which an activity should be disabled from the process. Cancel case patterns can model the cancellation of an entire process [PWR05]. Figure 2.22 presents the BPMN modeling of these patterns.

2.9 Comparison

The purpose of this section is to compare the process definition languages presented in this chapter from the viewpoint of their control flow modeling characteristics. We base our analysis on the work presented in [PWR05], in which workflow control patterns are used to compare different alternatives for the modeling of workflow processes.

We include in the comparison the following alternatives: BPMN, Activity Diagrams (UML), BPEL, XPDL, and Cumbia. We use the activity diagrams

Figure 2.22: Advanced control patterns [PWR05]

Pattern	BPMN	UML	BPEL	XPDL	Cumbia
1. Sequence	+	+	+	+	+
2. Parallel Split	+	+	+	+	+
3. Synchronization	+	+	+	+	+
4. Exclusive Choice	+	+	+	+	+
5. Simple Merge	+	+	+	+	+

Table 2.3: Workflow Basic Patterns [PWR05]

of UML as a generalization of SPEM.

Table 2.3 presents a comparison based on the workflow basic patterns. In this case, the patterns can be modeled with all the languages.

Table 2.4 presents a comparison based on the workflow advanced and structural patterns. In this group, BPMN and Cumbia support the most patterns.

Table 2.5 presents a comparison based on the workflow multi-instance patterns. BPMN, Cumbia, and the Activity Diagrams used in the UML notations support three of the four patterns.

Table 2.5 presents a comparison based on the workflow state and cancellation patterns. BPMN, Cumbia and BPEL support most of the patterns.

Pattern	BPMN	UML	BPEL	XPDL	Cumbia
6. Multiple Choice	+	+	+	+	+
7. Synchronizing Merge	+/-	-	+	+	+
8. Multiple Merge	+	+	-	-	+
9. Discriminator	+	+	-	-	+
Pattern	BPMN	UML	BPEL	XPDL	Cumbia
10. Arbitrary Cycles	+	+	-	+	+
11. Implicit Termination	+	+	+	+	+

Table 2.4: Advanced Synchronization and Structural Patterns [PWR05]

Pattern	BPMN	UML	BPEL	XPDL	Cumbia
12. MI without Synch.	+	+	+	+	+
13. MI Design Time KW	+	+	+	+	+
14. MI Runtime KW	+	+	-	-	+
15. MI without KW	-	-	-	-	-

Table 2.5: Multiple Instance Patterns [PWR05]

Pattern	BPMN	UML	BPEL	XPDL	Cumbia
16. Deferred Choice	+	+	+	-	+
17. Interleaved Parallel Routing	+/-	-	+/-	-	+
18. Milestone	-	-	-	-	+
Pattern	BPMN	UML	BPEL	XPDL	Cumbia
19. Cancel Activity	+	+	+	-	+
20. Cancel Case	+	+	+	-	+

Table 2.6: State Based and Cancellation Patterns [PWR05]

2.10 Summary

We have presented different process modeling technologies: BPMN, SPEM, IMS-LD, XPDL, and Cumbia. In particular, BPMN offers a technology-independent graphical language suitable for the description of the control flow of complex business processes.

We presented workflow control patterns for two reasons. First, to have a unit of comparison between the models and, second, to introduce the name and semantics of the patterns that will be used in the description of the work developed in this thesis.

From our comparison, we can conclude that, as a model definition language, BPMN fulfills most of the workflow control patterns. However, the lack of a standard storage and interchange format makes it necessary to translate the models into other languages, such as BPEL, XPDL, or Cumbia, to execute them. A workaround to this limitation is found in the Cumbia workflow engine, which allows the implementation of workflow engines, and particularly, the execution of BPMN processes.

Chapter 3

Model Driven Engineering

3.1 Introduction

One of the main challenges faced by software designers is the transformation of the requirements held by stakeholders into software systems. This transformation requires the construction of models to capture the essential characteristics of a problem and to assist in the development of a system. The life cycle of these models has usually been limited to the analysis and design phases, where the models are used as documentation artifacts to guide the initial phases of the development. However, during the codification phase, it is common to see discrepancies between the models and their implementations, due to changes introduced during the development of the products.

The Model Driven Engineering (MDE) movement proposes a framework to i) define methodologies clearly, ii) develop systems at any level of abstraction, and iii) organize and automate testing and validation, using models as first engineering artifacts [Fon04] [Wik07]. Thus, models are used not only as documentation but also as the main components throughout the software development process, until codification on a specific technological platform.

The objective of this chapter is to present the basic concepts of the MDE paradigm and some of its more representative implementations. This chapter is organized as follows: the first section describes the fundamental concepts and definitions required to understand the principles of MDE, and the second

section presents Model Driven Architecture (MDA) [OMG03], Eclipse Modeling Framework (EMF) [Ecl07b] and Model-Integrated Computing (MIC) [SK97].

3.2 MDE Fundamental Concepts

MDE uses models as first engineering artifacts. Unfortunately, as discussed in [Fav04], there is no clear and formal definition of MDE. The available definitions tend to be informal, in natural language, and, in the best of cases, Unified Modeling Diagrams (UML) without a precise semantic [Fav04]. This section defines three basic concepts of the MDE paradigm - models, meta-models and transformations - independent of a particular implementation.

3.2.1 Models

What is a model? Different authors have answered this question in different ways. In this section, we adhere to the definitions presented by Seidewitz in [Sei03] and summarized in table 3.2.1.

Definition 3. "A model is a set of statements about some system under study (SUS)" [Sei03]

Seidewitz states that a model can be used as either a description artifact or a specification artifact. Models used as descriptive artifacts abstract the behavior of a SUS. That is, they capture the essential characteristics of the system. Models used as specification artifacts define the desired behavior of a SUS under construction. For example, in the case of software development processes, a descriptive model can be used to represent the activities followed by a development team, whereas a specification model defines the activities that the development team must follow during the process.

Definition 4. "An interpretation of a model is a mapping of elements of the model to elements of the SUS" [Sei03]

Interpretations are transformations used to instantiate a model. For example,

a software development process model defines the activities to be executed during the codification of a component. For this model to be instantiated into a particular development process, it must be transformed into a platform-specific model. In this case, the interpretation transforms each element of the model into the language selected for the execution of the process.

Definition 5. “A theory is a way to deduce new statements about a SUS from the statements already in some model of the SUS.”[Sei03]

Theories enable the study of a SUS at a higher level of abstraction. Following the previous example, a theory can be used to detect inconsistencies in a process model, such as deadlocks or activities that never will be reached. Theories also permit the construction of *What if* scenarios to analyze the consequences of changes to the process.

Definition 6. “A modeling language is used to express the statements in models of some class of SUS.”[Sei03]

Due to the high level of abstraction used during modeling activities, it is common to create specialized modeling languages for the problem domain. In contrast to general purpose programming languages, Domain-Specific Languages (DSL) use the elements of the SUS as its main elements. For example, in software development processes, the UML profile developed to represent the Software Process Engineering Metamodel(SPEM) [OMG05c] facilitates the definition of a process model in terms of development concepts such as phases, iterations, activities and work products. Similarly, the Business Process Modeling Notation (BPMN) is a specialized modeling language that represents the control flow of a business process using concepts such as activity, process and gateway.

3.2.2 Metamodel

In the same way that models raise the level of abstraction of a SUS, meta-models raise the level of abstraction of models by specifying the language used to define them. The metamodel has the same elements as the model: interpretations, theories and modeling languages. However, these elements have different semantics. The definitions are summarized in table 3.2.2.

Model	Description
Definition	<p>A model is a set of statements about some SUS.</p> <ul style="list-style-type: none"> • A model used to describe a SUS is considered correct if all statements made in the model are true for the SUS. • If the model is used as a specification for one or a class of SUS, the SUS is considered valid relative to this specification if no statement in the model is false for the SUS.
Interpretation	<p>An interpretation of a model is a mapping between elements of the model and elements of the SUS in a way that the truth value of statements in the model can be determined from the SUS to some level of accuracy. If this mapping can be inverted, then a model can also be constructed as a representation of a given SUS.</p>
How is it used?	<p>A theory is a method of deducing new statements about a SUS from statements already in some model of the SUS. This method either extends the original model or determines whether the model conforms to the theory. Given a model that conforms to a theory, an interpretation of the model can be used to make deductions about a SUS.</p>
How is it expressed?	<p>A modeling language is used to express the statements in models of some class of SUS.</p>

Table 3.1: Main concepts defined for a model [Sei03]

Definition 7. “A metamodel is a specification model for which the SUS being specified are models in a certain modeling language.”[Sei03]

An example of a metamodel in software development is the Software Process Engineering Metamodel (SPEM) [OMG05b], which defines families of software processes using a modeling language designed for this purpose.

Definition 8. “An interpretation of a metamodel is a mapping of elements of the metamodel to elements of the modeling language”[Sei03]

Two of the main elements in the SPEM metamodel are WorkDefinition and WorkProduct. We can use an interpretation in which both elements are mapped to graphical elements of the SPEM language.

Definition 9. “A theory of a metamodel is a way to deduce new statements about a modeling language from the statements already in a metamodel of the modeling language”[Sei03]

Because a metamodel is a specification, it is possible to use theories to validate the correctness of models expressed with a modeling language. For example, the Rational Unified Process (RUP) [KK03], a process model defined using the SPEM UML profile as modeling language, can be validated with respect to the metamodel (SPEM) using a theory.

Figure 3.1 uses UML notation to summarize the definitions presented so far and the relationships between them.

3.3 Model Driven Architecture

Model Driven Architecture (MDA) is a set of standard definitions maintained by the Object Management Group (OMG). MDA promotes the use of models to facilitate the *separation of concerns principle*, that is, that the specification of the operation of a system must be independent of the capabilities of a specific platform. The primary goals of MDA are portability, interoperability and reusability, through architectural separation of concerns [OMG03].

The MDA strategy relies on three basic concepts: model, viewpoint and transformation. For the OMG, a model of a system is a “description or

Metamodel	Description
Definition	A metamodel is a specification model for which the SUS being specified are models in a certain modeling language. A metamodel makes statements about what can be expressed in the valid models of a certain modeling language.
Interpretation	An interpretation of a metamodel is a mapping of elements of the metamodel to elements of the modeling language, such that we can determine the truth value of statements in the metamodel for any model expressed in the modeling language.
How is it used?	A theory of a metamodel is a way to deduce new statements about a modeling language from the statements already in a metamodel of the modeling language.
How is it expressed?	A metamodel is expressed in some modeling language. A single modeling language might have more than one metamodel, each expressed in a different modeling language.

Table 3.2: Main concepts defined for a metamodel [Sei03]

Figure 3.1: UML class diagram that illustrates the concepts defined by Seidewitz [Sei03]

specification of that system and its environment for some certain purpose. A model is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language." [OMG03].

A viewpoint of a system is the "technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system" [OMG03]. The MDA specifies three types of viewpoints: computation independent, platform independent and platform specific. Each viewpoint produces a model of the system with particular characteristics.

The computational independent viewpoint produces a computational independent model (CIM), which does not show details of the system's structure. This model describes the situation in which the system will be used and is called the business model. The platform independent viewpoint allows for the SUS to be viewed from a platform independent model (PIM); in this case, the system is usually modeled based on a technology-neutral virtual machine. The platform specific model (PSM) results from the platform specific viewpoint of the SUS. This model is obtained as a result of the combination of a PIM and a particular platform model.

The Meta-Object Facility (MOF) technology provides a model repository that can be used to specify and manipulate models, encouraging consistency

in manipulating models in all phases of the MDA strategy. MOF 2.0 was designed by the OMG alongside UML 2.0 and reuses a common core infrastructure. A more simplified version of MOF, called Essential MOF (EMOF), is also part of MOF 2.0. EMOF is used for designing simple metamodels using object-oriented concepts [OMG03].

MOF is based on a four-layer metadata architecture used to metamodel conceptualization of the relationships between data and descriptions of them. These layers are information, model, metamodel and meta-metamodel. The information layer comprises the data to describe. The model layer contains metadata that describe the data in the information layer. The metamodel layer is composed of descriptions that define the structure and semantics of metadata. The meta-metamodel layer is composed of the descriptions of the structure and semantics of meta-metadata [OMG03]. Figure 3.2 shows an example of the four-layer MDA approach.

Figure 3.2: Example of the MOF layer architecture [OMG03]

These models represent different levels of abstraction during the modeling of a SUS. It is possible to move from one level to other using a transformation mechanism.

[Gra05] defines five types of transformations, grouped into vertical transformations and horizontal transformations.

Vertical transformations are refinement and abstraction transformations. Refinement transformations transform an abstract model into a more detailed one, whereas abstraction transformations transform a detailed model into a more abstract one. Figure 3.3 illustrates vertical transformations.

Figure 3.3: Vertical transformations between models [Gra05].

Horizontal transformations are migration, merge and identification transformations. Migration transformations transform a model into another one at the same level of abstraction. Merge transformations combine individual models, seen as different views or aspects, to form a complete software system. Model transformations show only part of a model using a selection filter. Figure 3.4 illustrates horizontal transformations.

3.4 The Eclipse Modeling Framework

The Eclipse Modeling Framework (EMF) [Ecl07b] is a modeling tool and code generation facility for building applications. EMF started as an implementation of the MOF specification. Currently, EMF is a Java implementation of a core subset of the MOF Application Programming Interface (API), Ecore. Ecore is an EMF model and its own metamodel [BBM03].

EMF is based on the MDE concept of using models as first-class entities during the development process: in EMF, a model is used to drive code

Figure 3.4: Horizontal transformations between models [Gra05]

generation and serialization for data interchange. In addition to the Ecore metamodel, EMF offers runtime support for models, including change notification, persistence support with XML Metadata Interchange (XMI) serialization, and a reflective API for manipulating EMF objects.

EMF also offers editing tools for manipulating models defined with Ecore. This support includes reusable classes for building editors and code generation capabilities. EMF allows the creation of a model using annotated java files, XML schema files or Rational Rose class models [IBM07b].

Figure 3.5 presents a general description of the Ecore metamodel as defined in [Ecl06]. The boxes in gray represent abstract classes.

The increasing acceptance of EMF and Eclipse as development tools has given rise to different commercial and open source projects to support model transformation languages as well as new creation and editing facilities, thanks to the reflective API offered by EMF. ATLAS Transformation Language (ATL) [Ecl07a], created by the *Institut National de Recherche en Informatique et en Automatique (INRIA)* laboratory as part of the ATLAS Model Management Architecture (AMMA) project, performs general transformations between models and is now part of the Eclipse development platform.

Another project that deserves attention is the OpenArchitectureware (oAW) [OAW07] framework, which transforms models. The framework offers three

Figure 3.5: The Ecore Metamodel [Ecl06]

important services to developers who work with EMF. The first is transformations of valid UML 2.0 models into Ecore models, which offer an alternative to the three creation mechanisms of Ecore models. The transformation of UML models into Ecore models is a specific case of the general purpose language *Xtend* [Eff07a], which supports transformations between arbitrary Ecore models.

The second service is the support for the definition of Domain-Specific Languages (DSL) using an extended Backus-Naur Form grammar and the *Xtext* [Eff07b] language. Based on the grammar, the framework automatically generates the corresponding Ecore metamodel. In this way, programs written using the DSL are valid Ecore models. *Xtext* also generates an Eclipse plug-in for editing the DSL.

The third service is the generation of code from Ecore models using the *XPand* [Eff06] language. For example, it is possible to transform models defined with a DSL into Enterprise Java Beans or Java code.

3.5 Model Integrated Computing

Another implementation of MDE is Model-Integrated Computing (MIC) [SK97]. This development paradigm is inspired by the necessity of modeling the integration of software and its physical environment [SKF96]. MIC

models capture not only a software’s architecture but also the environment’s [SK97].

MIC facilitates the modeling of complete domain-specific applications. For example, in the modeling of a router network, it is possible to construct a model of the physical device and all its relevant components, such as network interfaces, processors and control displays. It is also possible to model the relationships between these components. The resulting products are a metamodel and a DSL that can be used to construct models of particular applications.

MIC implementation relies on the MultiGraph Architecture (MGA) [SKB⁺95] framework developed at Vanderbilt University’s Measurement and Computing Systems Laboratory [SK97]. MGA allows for the development of model-integrated program synthesis (MIPS)[SKF96] environments, which include tools for building, testing and storing domain models using a graphic model builder (GMB), a model database, model interpreters and a multigraph kernel.

GMB permits the design of domain models, usually by domain experts, that graphically represent the concepts, relationships and constraints present in the SUS. The model database stores different views of the models constructed. Model interpreters produce executable programs from domain models specified in terms of directed graphs [SKB⁺95].

MGA is structured in a three-layer architecture composed of the following levels: application, MIPS and metalevel (see Figure 3.6) ¹

3.5.1 Domain-Specific Modeling Languages

A Domain-Specific Modeling Language (DSML) captures the concepts, relationships, integrity constraints and semantics of the application domain and allows users to program declaratively by means of model construction [CSN05]. A DSML facilitates interaction with domain experts by including concepts they already manage.

A DSML can be defined in five elements: concrete syntax (C), abstract syntax (A), semantic domain (S), and semantic and syntactic mappings (MS and

¹Simplified version of the diagram presented in [SKB⁺95] pg. 4

Figure 3.6: MGA Levels. Source [SKB⁺95]

Mc) [CSN05]. As defined in [CSN05], concrete syntaxes define the notation (graphical and/or textual) used to express models. Abstract syntaxes define the relationships and integrity constraints in the language. The semantic domain expresses the meaning of the models. Mc transformations express a mapping from concrete to abstract syntax, whereas Ms transformation express a mapping from an abstract syntax to semantic domain [CSN05].

3.5.2 The Generic Modeling Environment

One alternative to defining the elements of a DSML is to use the Generic Modeling Environment (GME)[Led01], a meta-programmable model builder for creating and modeling in domain-specific modeling environments. GME offers a graphical metamodeling language called MetaGME for the specification of DSMLs. MetaGME is an extension of UML class diagrams with UML constraints that manages the following set of modeling elements [ESB04]:

- Models: Compound objects, which contain other model elements
- Atoms: Elementary objects, which do not contain other model elements
- FCOs: First-class objects, which must be abstract but can serve as the base type of an element of any other stereotype

- Connections: Analogous to UML associations classes
- Aspects: Different views of the model

Metamodels can be defined using the MetaGME meta-metamodel. The editor also supports the definition of constraints using the Object Constraint Language (OCL). Finally, the tool offers a graphical selection system based on the creation of aspects that file the elements visualized during the editing of a model.

3.6 Summary

The Model Driven Engineering approach recommends the use of models as first-class entities during all phases of the development process. Different alternatives have been proposed to fulfill the objectives proposed by the MDE, such as Model Driven Architecture, the Eclipse Modeling Framework and Model Integrated Computing.

The MDA approach uses a layered architecture based on different levels of abstraction. The Meta Object Facility is at the top level of abstraction and is called the meta-metamodel.

The EMF alternative is supported by IBM as part of the Eclipse Development Platform. EMF was initially designed as a simplified version of MOF, but has since evolved independently. The meta-metamodel in EMF (Ecore) is used to create models and to generate code and editors for the manipulation of those models.

The MIC implementation is supported by Vanderbilt University and has its origins in the necessity of modeling not only the software but also its execution environment. It is possible to use the Generic Modeling Environment, which is based on the principle of abstraction, to implement the MIC approach and to create Domain-Specific Languages. In this hierarchy, the most abstract layer (meta-metamodel) is the MetaGME.

Implementation	MDA	EMF	MIC
Supported By	OMG	IBM Eclipse	Vanderbilt University
Meta-Metamodel	MOF	Ecore	MetaGME
Example Models	UML2.0, SPEM	BPMN	Avionics, Networking
Constraint Language	OCL	OCL	OCL
Transformations	QVT	Reflective API	Reflective API

Table 3.3: Summary of the MDE implementation alternatives presented in this chapter.

Chapter 4

Representation of Concerns

4.1 Introduction

Software development involves several major phases, such as analysis, design, coding and testing. Of these phases, design plays a fundamental role because it is where the requirements expressed by the users are grouped, subdivided and modularized according to a selected architecture. Thus, at design time, an important change occurs in the level of abstraction and representation of a system: the concerns expressed by the stakeholders are mapped onto one or more parts of the architectural design.

In software development, *concerns* have been defined in various ways. The Aspect-Oriented Software Development glossary defines concerns as “the primary criteria for decomposing software into smaller, more manageable, and comprehensible parts that have meaning for a software engineer.” [AOS07]. In [IEE00], a concern is “the interest which pertains to the system’s development, its operation, or any other matters that are critical or otherwise important to one or more stakeholders”. Thus, a concern is more general than a requirement; furthermore, it is possible that several requirements are needed to fulfill a particular concern.

One of the most challenging tasks in software development is the proper choice of concerns representation in the architecture of a system. The same concern can be represented in different ways. For example, a particular con-

cern could be mapped to different services in a service-oriented architecture, or it could be represented by a class in an object-oriented decomposition.

Due to the limitations imposed by traditional decomposition mechanisms, a system can usually be modularized in only one way at a time. This is the *tyranny of the main decomposition*, which states that once a decomposition mechanism has been selected to define the architecture of the system, all concerns of that system must be represented using that and only that type of representation.

During the definition of a system’s architecture, concerns are mapped onto elements of the design. Figure 4.1 (A) presents the crosscutting pattern defined in [vdBC05]. This pattern is used to express the different relationships present during the representation of concerns. In this figure, concerns are called sources and representations are called targets.

Figure 4.1: The Crosscutting Pattern. Source [vdBCH06] [vdBC05]

Figure 4.1 (B) gives a scenario with three concerns represented with four elements of the design. Concern S1 is represented by elements t1, t3 and t4, and concern S2 is represented by t2. Using the crosscutting pattern [vdBC05] in the context of a particular representation, it is possible to define three types of relationships between concerns: scattering, tangling and crosscutting.

According to [vdBC05], “i) Scattering occurs when, in a mapping between source and target, a source element is related to multiple target elements. ii) Tangling occurs when, in a mapping between source and target, a target

element is related to multiple source elements. Two source elements are tangled if these elements are mapped onto the same target element. iii) Crosscutting occurs when, in a mapping between source and target, a source element is scattered over target elements and where in at least one of these target elements, some other source elements are tangled”.

Based on these definitions, we can affirm that for the scenario presented in figure 4.1 (B) [vdBCH06], concern S1 crosscuts concern S3.

In [vdBC05], Conejero proposes a method to identify the relationships between concerns in a system. This method is based on the construction of various matrices - dependency, scattering, tangling, crosscutting product and crosscutting - to detect scattered and tangled relationships between source and target elements. Using these matrices, we can detect not only scattering and tangling relationships but also crosscutting concerns.

To illustrate the definitions presented above and the methodology proposed by Conejero, we use the design of a software tool to support software development processes. In this scenario, the stakeholders have three concerns: implementation, verification and monitoring.

The implementation concern is associated with the design, codification and testing activities during the development of the product’s components. Verification refers to the revision and inspection of artifacts produced during the different activities. Finally, the monitoring concern refers to the tracking and control of the development activities.

Figure 4.2 uses a class decomposition to present the example representation. Consistent with the terminology in the crosscutting pattern[vdBC05], the implementation, verification and monitoring concerns are the sources, and the product, component, activity and metric classes are the targets.

Following the method proposed in [vdBCH06], the first step is to define the dependency matrix between the source and the target elements. Using this matrix, we can determine the scattered elements (denoted by an S at the end of the row) and the tangled elements (denoted by a T at the bottom of the column). In table 4.1, the three concerns are scattered and the component and activity targets are tangled.

Table 4.2 shows the final crosscutting matrix with the details of the scattering, tangling and crosscutting product matrices omitted. In this matrix, a

Figure 4.2: Crosscutting concerns example

	Product	Component	Activity	Metric	
Implementation	1	1	1	0	S
Verification	0	1	1	0	S
Monitoring	0	0	1	1	S
	NT	T	T	NT	

Table 4.1: Dependency Matrix for the example presented in figure 4.2

	Implementation	Verification	Monitoring
Implementation	0	1	1
Verification	1	0	1
Monitoring	1	1	0

Table 4.2: Crosscutting Matrix for the example presented in figure 4.2

value of 1 in position $[i,j]$ indicates that concern i crosscuts concern j ($i \neq j$). For the proposed representation, it can be stated that the three concerns crosscut each other.

Scattering, tangling and crosscutting relationships guide fundamental decisions during the design and implementation of software systems. Different approaches have been proposed to aid in defining crosscutting concerns as well-identified separate units, such as aspect-oriented programming (AOP) [KLM⁺97], Hyperspaces [TOHJ99], adaptive programming [Lie96] and subject-oriented programming [HO93].

In this chapter, we discuss the use of aspects as a mechanism for representing crosscutting concerns as independent and non-intrusive modules in software systems. Section two gives an introduction to aspect-oriented technologies and the use of aspects in all phases of software development. Section three presents the use of aspects at the model level. Section 4 describes the use of aspects in domain-specific languages.

4.2 Aspect Oriented Technologies

Aspects were first introduced in [KLM⁺97] as a programming paradigm to encapsulate crosscutting concerns in a given software system using a new kind of module, an aspect [KS]. An aspect provides a unit of encapsulation that binds the behavior of a crosscutting concern with the points in the program where it applies. Then, an aspect weaver combines the base program with the concerns, producing a new executable program.

Initially, the AOP paradigm was applied to the codification phase using languages such as AspectJ [Pro06]. However, the necessity of encapsulating crosscutting concerns is not present only during this phase but also earlier

in the process. Aspects are used during the analysis of requirements and in the design and testing phases. In general, Aspect-Oriented Software Development (AOSD) has been gaining acceptance in the community, which has begun generating new research threads, supporting tools and methodologies around this concept.

AOP is based on some key concepts: join points, pointcuts and advices. The particular instantiation of these concepts is called a join point model. The following section explains these concepts in detail.

4.2.1 Join Point Models

To support the implementation of crosscutting concerns, AOSD techniques rely on join point models to specify where, when and how aspects observe, augment or alter a system [Ali05]. The following definitions are intended to formalize the basic concepts involved in aspect-oriented development.

Definition 10. “A base concern is a unit of modularization with respect to vertical decomposition of the system and is referred as non-crosscutting concern.” [Iqb06]

Definition 11. “An adaptation specifies in what way the concern’s structure or behavior is adapted, i.e., enhanced, replaced or deleted” [SA06a]

Definition 12. “An aspect is a unit of modularization formalizing a cross-cutting concern that otherwise would be scattered across other concerns” [SA06a].

Definition 13. “A join point specifies where an aspect might insert adaptations. Thus, a join point is a representation of an identifiable element of the underlying language used to capture a concern.” [SA06a]

Definition 14. “A join point selection, or pointcut, represents a set of join points selected for the purpose of specifying certain adaptations.” [SA06a].

Definition 15. “A join point model (JPM) comprises as possible join points all elements of a certain language where aspects are allowed to introduce

adaptations." [SA06a].

Definition 16. "Weaving is the act of composition of aspect and base concerns, which yields a woven model of the overall system." [Iqb06]

To illustrate these definitions, we revisit the example presented in section 4.1. We use the AspectJ language to present an implementation of this example using the separation of concerns principle. Aspects encapsulate the monitoring crosscutting concern. In this example, the implementation concern is represented by product, component and activity classes. These classes register the development state of each component of the project. In addition, the activity class has administrative information, such as who is responsible for the activity, the developers assigned and the requirements that the component implements. The monitoring concern tracks the evolution of the project. For example, each time an activity changes its internal state, a register is created to record the date and time of this event. Figure 4.3 presents a simplified implementation of this concern using the AspectJ plug-in for the Eclipse development platform.

```
1 package org.aspectviewpoint.project;
2
3 import java.util.Vector;
4
5 public aspect Monitoring {
6
7     private Vector<Metric> changes = new Vector<Metric>();
8
9     pointcut stateChange(Activity obj): target(obj) &&
10        call (void setState (String));
11
12     after(Activity obj):stateChange(obj) {
13         changes.add ( new Metric (obj.getName(), obj.getDate(), obj.getState()));
14         System.out.println ("New metric added !!");
15     }
16 }
```

Figure 4.3: Implementation of the Monitoring Aspect using the AspectJ language.

In this example, an adaptation is introduced into the base program. Each time an activity changes its current state, a new metric is created and the name of the activity, the date of the change, and the new state are registered.

The first step is to create a new aspect using the AspectJ language (line 5).

The second step is to define a set of join points where the adaptations will take place. In this case, we want to capture all calls of the *setState* method in the objects of the activity class. Line 9 shows the definition of a pointcut called *stateChange*. This pointcut takes as its argument the object in which the method *call* takes place (the context object) using the primitive method *target*. Pointcuts also allow the combination of several selection methods using logical operators such as *and*, *or*, and *not*.

The final step is to define how the adaptation is implemented. Line 12 presents the definition of the advice, referencing the pointcut defined in the previous step. The keyword *after* is used to express that the adaptation will take place after the occurrence of the pointcut; other possible values are *before* and *around*. The adaptation creates a *Metric* object with the information contained in the activity object as the context parameter.

From the example, it is clear that using aspects as separation of concerns mechanisms avoids the re-factoring of existing code (in this case the *Activity* class), which maintains the implementation of the concern in a separate module and prevents the scattering of this concern into other components of the system.

The adoption of AOP techniques does have some drawbacks. First, aspects could break object encapsulation because of the facility to define new attributes and methods of a class outside of the class definition. Second, most existing join point models define a pointcut selection based on so-called pointcut designators or selection methods, such as *target*, *call* and *execution*. These pointcut designators specify the location of their target(s) by naming elements of their corresponding base program [KS]. These explicit references introduce a coupling link between the base system and the aspect[KS], which is known as the *fragile pointcut problem*[KS]. Consider line 10 of figure 4.3. Here, the pointcut designator *call* looks for all calls to the method *setState (String)*. If re-factoring happens to the *Activity* class and the *setState (String)* method is deleted, the pointcut will fail.

4.3 Aspect-Oriented Modeling

AOSD can also be used in earlier stages of software development, including using aspects at the modeling level with the MDE approach. Aspect-Oriented Modeling (AOM) supports the separation of concerns that otherwise will be scattered between the components of a domain-specific model [Zha06][SA06a].

The purpose of AOM is to use aspect-oriented techniques close to the problem space (models) as opposed to the solution space (code) [JGN03]. Thus, AOM is based on the same concepts found in AOP: join points, pointcuts, advices and weaving. However, these concepts are used in a model context rather than a language programming environment. The following sections present examples of the application of AOP concepts in the AOM scenario.

4.3.1 Join Points

In AOP, join points can be static or dynamic. Static join points denote elements of the program that can be selected outside the execution context of the program. Dynamic join points denote elements of a program that are only visible during its execution. For example, a particular class or attribute in an object-oriented program can be used as a static join point. The execution of a method defined in this class is a dynamic join point, which can only be reached during the execution of the program.

Static and dynamic join points can also be defined in AOM. Here, static join points refer to entities defined in a model that do not depend on its execution, for example, an activity or an artifact in a software process model. Dynamic join points refer to elements of the model that are only present during its execution. In a process model, the states associated with the execution of an activity (i.e., active, ready, suspended, aborted, cancelled) are dynamic join points.

In AOP, join points are located in the program code (source, or binary). In AOM join points can be located in a graphic, an XML file, or any other type of model representation.

4.3.2 Pointcuts

In AOP, a pointcut is a set of join points obtained after the use of a filer, for example, the methods with names starting with the string *get* in a class definition. Similarly, in AOM, a pointcut is a selection of entities defined in a model, for example, all the activities whose names start with the string *Review*. In both cases, matching expressions can be used to obtain sets of join points.

4.3.3 Advices

In AOP, an advice defines the desired behavior at a join point. For example, each time the method *getName* is called, it can be intercepted and the response encrypted. In general, this behavior can happen *before*, *after*, or instead of (*around*) the occurrence of the join point.

Advices can define the desired behavior at a join point in AOM, for example, the addition of a new activity. However, the semantics of *before*, *after*, and *around* depend on the specific model. Following the example, it is possible to define that the new activity must be created after the finalization of another activity. In another domain, for example a logic circuit model, *after* could refer to the achievement of a particular voltage (i.e., +5V) in a circuit component.

4.3.4 Weaving

There are two types of weaving in AOP: static and dynamic. Static weaving is the integration of the advices into the base code of the program at compilation time. In dynamic weaving, this integration happens at runtime.

In AOM, static weaving is done at modeling time instead of at execution. There is also a difference between AOP and AOM in the result of static weaving. In AOP, the result of weaving is seen only at runtime. For example, if an advice adds a new attribute in a class using static weaving, the resulting code has the modification, but this can only be confirmed during the execution of the program. In AOM, the result of static weaving of an advice is a model,

which can be visualized and manipulated with the new modifications, even if it is not executed.

In AOM dynamic weaving, advices are integrated into the base model during its execution. This implies a more complex solution because it is necessary to have a model execution engine designed with this feature in mind.

4.3.5 AOM Example

To illustrate the use of AOM, we use a partial modeling version of the example presented in section 4.2.1. We are interested in the execution of a metric activity each time one of the activities finishes its execution. Figure 4.4 shows the proposed process model defined using the Generic Modeling Environment (GME). The following activities are defined: the construction of the detailed design (ProduceDLD), the codification of the design (Codification) and the execution of the unit tests (UnitTesting). These activities are executed following a sequence workflow pattern (see section 2.8).

Figure 4.4: The implementation phase model before the weaving of a cross-cutting concerns.

The next step is the definition of an advice to express that a metric activity

will be introduced after the execution of the ProduceDLD and Codification activities. This new activity is executed in a non-synchronized way, forming a Parallel Split workflow control pattern (see section 2.8). The advice is weaved with the base process model to obtain the process model shown in figure 4.5.

Figure 4.5: The implementation phase model after the weaving of a cross-cutting concern.

4.3.6 The AOM Framework

In this section, the AOM framework proposed in [SA06b] [SA06a] is used as a common reference architecture that makes explicit the AOP concepts and relationships used in AOM. The purpose of this framework is threefold: “i) to offer a common reference architecture to identify the basic ingredients of aspect-orientation, abstracted from certain AOP languages and AOM approaches; ii) to propose an evaluating and comparing mechanism for different AOM approaches; and iii) to offer a pivot model to the transformation of models developed with different AOM tools” [SA06b] [SA06a].

Figure 4.6 presents the AOM framework using UML notation.

Figure 4.6: The AOM framework. Source [SA06a] [SA06b]

The framework is organized into four modules: *ConcernComposition*, *AdaptationSubject*, *AdaptationKind* and *Language* [SA06b] [SA06a].

The *ConcernComposition* module presents the general decomposition of the system, making the relationships between concerns, base and aspects explicit without specifying the adaptations and join points. The *AdaptationSubject* module describes the concepts required for identifying where to introduce an aspect's adaptations. The *AdaptationKind* module comprises the concepts necessary to describe an aspect's adaptation. Finally, the *Language* module describes the language underlying the specification of base and aspect.

4.4 Domain-Specific Aspect Languages

The use of model driven techniques to support the process development software facilitates communication between software engineers and final users. Models abstract the main elements of a target domain in which the final users are usually experts; hence, engineers and users can speak the same language.

However, to define the particular business logic involved in a software system, developers need to use General Purpose Programming Languages (GPPL) such as Java or C++. This implies that there is a translation of the concepts defined in a model to the primitives offered by the GPPL, which means different vocabularies are being used by users and developers. Domain-Specific Languages (DSLs) are mechanisms that help users and developers use the same vocabulary by creating programming languages for a particular domain in which the primitives, data types and logic are all expressed in the users' vocabulary.

DSLs provide a programming language for developers while avoiding the translation of models and concepts of a particular domain to GPPLs. DSLs have more power of expression and more compact and declarative syntaxes [DR06].

General Purpose Aspect Languages (GPALs) handle the definition of cross-cutting concerns in a modular way with respect to GPPLs. For example, the AspectJ language supports the definition of crosscutting concerns in Java programs, and AspectC++ [Org07] does the same with respect to C++ programs.

The use of aspects in model-specific domains requires Domain-Specific Aspect Languages (DSALs). DSALs support the description of domain-specific crosscutting concerns that provide language constructs tailored to their particular representation [DR06] [Hut06].

DSALs were initially proposed way by C. Lopes [LK97] in 1997. Lopes defined a framework for distributed programming called *D*. As part of this implementation, she defined two aspect languages: the Coordination Aspect Language (COOL) and the Remote Interface Aspect Language (RIDL). COOL provides constructs for dealing with mutual exclusion of threads, synchronization state, guarded suspension and notification. RIDL defines how and what data are sent when methods are invoked between different execution spaces [LK97]. These languages are considered the seminal works in this area.

The definition of a DSAL is not an easy task; the design and implementation of such languages is demanding and costly work [DR06], and in many situations more than one DSAL must be defined to handle the decomposition of different concerns in a specific domain.

In [DR06], the authors state the main issues facing the designer of a DSAL, starting with the definition of a Domain-Specific Join Point Model for the aspect language. Some of these considerations are:

- What join points will be captured by the DSAL?
- Are the DSAL join points static or dynamic?
- What granularity is required for these join points?
- What is an appropriate pointcut language to describe these join points?
- What are advices in this domain?
- Is extension/refinement only about behavior or also about structure?
- How is information exchanged between join points and their associated advices (context exchange)? Is parameterization of advices needed?

Other challenges for DSAL designers are aspect ordering, aspect dependency and interference, and how are these factors are allowed and handled [DR06] [Zha06].

4.5 Summary

Aspects are suitable in the modularization of crosscutting concerns that are impossible to handle using General Purpose Programming Languages. The use of aspects has been extended to different phases of software development, in particular, to the earlier stages of analysis and design.

Aspect Oriented Modeling is a particular case of Aspect Oriented Software Development in which the separation of concerns principle is applied to the model of a system. The use of models promotes the incorporation of concepts and terms belonging to the vocabulary of the problem and used by the experts in the domain. Domain-Specific Languages define these terms, introducing constructs into the language that raise the generality of General Purpose Aspect Languages. Domain-Specific Aspect Languages combine the specialization of DSLs and the separation of concerns offered by GPALs.

Part II

*"Hope your road is a long one.
May there be many summer mornings when,
with what pleasure, what joy,
you enter harbors you're seeing for the first time;
may you stop at Phoenician trading stations
to buy fine things,
mother of pearl and coral, amber and ebony,
sensual perfume of every kind-
as many sensual perfumes as you can;
and may you visit many Egyptian cities
to learn and go on learning from their scholars."*

from *Ithaka* by C.P. Cavafy

Chapter 5

The AspectViewpoint Language

5.1 Introduction

Domain-Specific Aspect Languages (DSALs) support the definition of cross-cutting concerns using the vocabulary and concepts of a particular domain. In this chapter, we present AspectViewpoint, a DSAL designed for models of workflow processes. In this domain, stakeholders of a process express concerns regarding the activities, products and participants that affect both the structure of the model and its execution control flow.

In the AspectViewpoint language, a viewpoint defines the points in a process model where the concerns have an effect, as well as the strategies that implement these concerns. Within the design of the language, we created the Viewpoint Join Point Model (VJPM), which defines the set of valid join points, pointcuts and advices in a process model. The join points and pointcuts are defined in terms of the elements of a BPMN process model. The advices can be expressed in terms of the workflow control patterns presented in section 2.8 of chapter 2.

A complete reference of the AspectViewpoint language can be found in appendix A.

This chapter presents the AspectViewpoint language in detail. Section two presents the mechanism used to define join points in a process model. Section three explains how pointcuts are expressed using the AspectViewpoint lan-

guage. Section four presents the definition of advices. Section five describes the viewpoint classification based on the modifications introduced into target process by the advices. Section six details the workflow control patterns supported by the AspectViewpoint language. Finally, section seven illustrates the concepts using an example.

5.2 Join Points

In our model, a join point can be any of the *FlowObjects* elements defined by the BPMN specification (see section 2.2): tasks, subprocess, events and gateways. Join points can be classified as static or dynamic, based on whether or not they are related to the execution of the process model.

5.2.1 Static join points

Static join points are entities of the process model that do not depend on events occurring during the execution of the process, for example, a particular activity or work product in the process definition. Static join points can be selected using matching expressions such as

```
Name ("Codification")
```

or

```
Type("MultiActivity")
```

In these examples, the attributes *name* and *type* of an activity element are used as join points. These attributes are static because they are not the results of the execution of the process model.

Figure 5.1 shows two types of static join points: activities and gateways.

5.2.2 Dynamic join points

The Viewpoint Join Point Model (VJPM) supports the definition of dynamic join points to represent events that may happen during the execution of a process. To model dynamic join points, we use the states associated with the

Figure 5.1: Static join points in the AspectViewpoint language

execution of an activity as defined by the BPMN specification: none, ready, active, cancelled, aborting, aborted, completing and completed.

The selection of dynamic join points is based on the description mechanism for static join points plus a selection method to determine the desired state. For example, the following expression is used to select the point during the execution of the process where the activity *Codification* reaches the *active* state

```
name ("Codification") && status("active")
```

Figure 5.2 presents the state machine associated with the execution of an activity. These states can be used as dynamic join points to capture a particular event during the execution of the process model. Gateways and events are not associated with state machines; hence, they cannot be used to select dynamic join points.

5.3 Pointcuts

A pointcut is a subset of process elements in a process model definition. This set is formed from the elements returned by the applied selection filters. These filters select join points that share particular conditions. For example, if a concern states that all *Multiinstance* activities require a particular treatment, the first step is to define a pointcut that filters the activities of type

Figure 5.2: The states associated to the execution of an activity are used as dynamic join points in the AspectViewpoint language.

```

pointcut findtask (Task task) : target (task) &&
                                name ("ConstructDLD");
  
```

Listing 5.1: Pointcut definition

Multiinstance from the other elements of the process.

The filters used to construct pointcuts in our model are called selection methods. Filters are composed using Boolean operators (and, or, not), and several may be applied. Following the previous example, we specify two selection methods: one for selecting all activity elements in the process model, and the other for selecting all *Multiinstance* activities. This condition is expressed with an *AND* operator.

Listing 5.1 presents an example of a pointcut called *findTask*, which selects all join points in the process model with the name “ConstructDLD”.

In the AspectViewpoint language, every pointcut must have a name that uniquely identifies it. Pointcuts can also define parameters to reference the join point returned by the selection methods, using the selection method *target*.

Table 5.3 summarizes the selection methods supported by the AspectViewpoint language. *Activity* elements are selected by the name of the element, the activity type (task or subprocess), the loop type of the activity (none,

standard or multiinstance), or the status reached by the activity during the execution of the process model. *Gateway* elements have only one selection method, based on the type of the element (EventXor, DataXor, OR, AND or Complex). Finally, *event* elements have two selection methods: type (start, end or intermediate) and trigger (none, message, timer, error, cancel, compensation, rule, link or multiple).

FlowObject	Selection Method
Activity	<ul style="list-style-type: none"> • name (STRING) • type (Task SubProcess) • loopType (None Standard MultiInstance) • status (None Ready Active Cancelled Aborting Aborted Completing Completed)
Gateway	<ul style="list-style-type: none"> • type (DataXor EventXor OR AND Complex)
Event	<ul style="list-style-type: none"> • type (Start End Intermedaite) • trigger (None Message Timer Error Cancel Compensation Rule Link Multiple)

Figure 5.3: Selection methods offered by the AspectViewpoint language

5.4 Advices

An advice defines the adaptations applied to the join points selected by a pointcut. In the AspectViewpoint language, we implement advices with blocks of code called *strategies*, which bind a pointcut definition to a set of adaptations that modify the process model's structure and behavior. These adaptations can be as simple as changes in property values or as complex as changes in the control of the execution of a process.

The *target* method, which can be used in the pointcut declaration, facilitates the exchange of information between join points and advices. Thus, the AspectViewpoint language supports advice parameters that establish the passing of content between pointcuts and strategies.

Listing reflst:strategy shows the definition of an advice in the AspectViewpoint language using strategies. This example defines a viewpoint called *Verification*, which belongs to a group of Viewpoints called *Engineering*. The

```

1 viewpoint Verification
2 affiliation Engineering
3 targetprocess TechnicalSolution
4 {
5     pointcut findCodification (Task task) : target (task) &&
6                                           name ("Codification");
7
8     strategy addCycle (Task task) : around findCodification (task)
9     {
10         ... // Adaptations
11     }
12 }

```

Listing 5.2: Implementation of advices using strategies

target process is *TechnicalSolution*. Line five defines the pointcut *findCodification*, which selects all the elements of type *Task* with the name *Codification*. Line eight gives a strategy, *addCycle*, which is linked to the *findCodification* pointcut. Inside the strategy, we define the adaptations required to modify the process model in the context defined by the join point received as a parameter by the strategy.

5.5 Viewpoint classification

Depending on how viewpoints define adaptations of a target process, they can be classified into three main categories: value, observation, and transformation.

5.5.1 Value Viewpoints

Value viewpoints define adaptations by changing the value of some attribute of an element, for example, changing the expressions associated with a condition in a gate or the name of a task. Value viewpoints do not add or delete new elements, they only change attributes on the existing ones.

As an example of value viewpoints, consider the target process presented

in figure 5.4. In this model, after the execution of the *UnitTesting* task, a condition gateway is evaluated to determine the execution path.

Figure 5.4: Target process model associated to the verification viewpoint.

If after the execution of the *UnitTesting* activity there still defects, the *RegisterDefects* task is executed and the codification process starts again. If there are not defects, the process finishes with the execution of the *CreateBL* task. However, this condition may be changed due to a reorientation of quality standards of the company. A value viewpoint is used to select the condition gateway and change some of the condition expressions used during the evaluation. For example, if the quality standards state that an acceptable value for the number of defects per thousand lines of code (defects/KLOC) is five or fewer, we can change the conditions of the gates at this point, as shown in figure 5.5.

Figure 5.5: Example of a value viewpoint.

This type of change is a value adaptation. The methods offered by the language to support value adaptations are explained below.

Adaptation	Activity	DataObject	Gateway	Event
setName ()	✓	✓	-	-
setType ()	✓	✓	-	-
setLoopType ()	✓	-	-	-
setTrigger ()	-	-	-	✓
setCondition ()	-	-	✓	-

Table 5.1: Value adaptations supported by the AspectViewpoint language

- *setName ()* : Modifies the name of the attribute of an activity or data object (product).
- *setType ()* : Changes the type of a process element. An activity can be changed to either task or subprocess. A gateway can be changed to type DataXor, EventXor, OR, AND or Complex. An event can be changed to start, end or intermediate.
- *setLoopType ()* : Changes the loop attribute of an activity to none, standard or multiinstance.
- *setTrigger ()* : Changes an event to none, message, timer, error, cancel, compensation, rule, link or multiple.
- *setCondition ()* : Changes the conditions of the gates in a gateway that are associated with the evaluation expression.

Table 5.1 summarizes the value adaptations.

5.5.2 Complex Viewpoints

Advices may require the use of more complex adaptations, such as the addition of new activities or changes in their execution order. Complex adaptations affect the structure of the process model and its control flow during execution. They can be classified as either observational or transformational.

Observational Viewpoints

Observational viewpoints depend on the occurrence of events in the target process, such as the initialization or finalization of a task, to trigger the execution of parallel control execution threads. This type of viewpoint is observational because they capture the occurrence of an event without interfering with the target process execution. The execution of the viewpoint happens in parallel with the original target process, but they are not synchronized.

Observational viewpoints are illustrated in the following example. The target process model is the same as in figure 5.4. Here, we register the initialization and finalization of the *codification* activity to control the total time involved during the development of the product. Figure 5.6 shows the target process after weaving of the observational viewpoint.

Figure 5.6: Example of an observational viewpoint.

Transformational Viewpoints

Whereas observational viewpoints are started only by events occurring in the target process, transformational viewpoints do not depend on the occurrence of events. The main difference between observational and transformational viewpoints is that a transformational viewpoint introduces synchronization points into the process to join its execution path with the process, whereas observational viewpoints do not. This means that the target process controls the execution of viewpoints, but also that viewpoints control the execution of the target process.

Consider the verification viewpoint in which the *CreateDLD* and *Codification* tasks are reviewed and inspected. In this example, the execution of the transformational viewpoint is triggered by the target process, but the process must wait until the end of the viewpoint’s execution to continue. To model this situation, two synchronization gateways are introduced into the model, transforming its original structure. Figure 5.7 illustrates the process model resulting from the weaving of the verification viewpoint.

Figure 5.7: The implementation process model after the inclusion of the verification viewpoint.

5.6 Viewpoints and Workflow Patterns

The AspectViewpoint language supports the definition of complex viewpoints using a subset of the workflow control patterns presented in chapter 2 (page 37). The supported workflow patterns are sequence, parallel split, synchronization, exclusive choice, merge, multiple choice, arbitrary cycles and deferred choice. In the following sections, we explain the semantics associated with each of these patterns in the context of a workflow process model.

5.6.1 Sequence Pattern

purpose: A new FlowObject element is added into a process model following a sequence pattern.

use: `caller.Sequence ([objectid, <conditionexpression>])`. A caller can be a task, subprocess, gateway or event.

implementation: The pattern is implemented by adding a SequenceFlow between two FlowObjects in the following manner.

- The connection is added from the caller element to the *objectid* FlowObject.

exceptions: If the join point element is an end event, the operation is forbidden.

example: Figure 5.8 shows an example of this pattern. The pointcut selects activity A; a sequence pattern is used to weave new activity B.

Figure 5.8: Example of the weaving of a sequence pattern into a process model

5.6.2 Parallel Split Pattern

purpose: A process model is modified by introducing a divergence point with two or more parallel execution paths.

use: `caller.ParallelSplit ([obj1, <condition1>],...[objN, <conditionN>])`.

A caller can be a task, subprocess, gateway or event element.

implementation: The pattern is implemented by an *And* gateway

- *before*: This option is not supported by this pattern.
- *after*: A SequenceFlow is connected from the caller to a new *And* gateway. For each *object* used as an argument, a new gate is added to the gateway. If the pointcut element has a connection with another FlowObject, this connection is added to the gateway.
- *around*: This option is not supported by this pattern.

exceptions: If the join point element is an end event, the *after* option is forbidden.

example: In figure 5.9, the selected pointcut is activity A, and a parallel split pattern is used to add two new execution threads corresponding to activities B and C.

Figure 5.9: Example of the weaving of a parallel split pattern into a process model

5.6.3 Synchronization Pattern

purpose: A process model is modified by introducing a convergence point of two or more parallel execution paths.

use: `Caller.Synchronization ([obj1, <condition1>],...,[objN, <conditionN>])`.
A caller can be a task, subprocess, gateway or event.

implementation: The pattern is implemented with an *And* gateway.

- *before:* A synchronization pattern adds an *And* gateway before the caller, two or more execution threads are defined as IncomingFlows connections of the gateway. If the caller has a connection with a FlowObject before it, this connection is added to the *And* gateway.
- *after:* This option is not supported by this pattern.
- *around:* This option is not supported by this pattern.

exceptions: If the join point element is a start event, the *before* option is forbidden.

example: In figure 5.10, the pointcut is activity A and a synchronization pattern merges two new execution threads from activities B and C.

Figure 5.10: Example of the weaving of a synchronization pattern into a process model

5.6.4 Exclusive Choice Pattern

purpose: A process model is modified by introducing a divergence point in which only one of two or more execution paths is selected.

use: `Caller.ExclusiveChoice ([obj1, <condition1>],...,[objN, <conditionN>])`.
A caller can be a task, subprocess, gateway or event.

implementation: The exclusive choice pattern is implemented by adding a *DataXor* gateway.

- *before*: This option is not supported by this pattern.
- *after*: A *DataXor* gateway is added after the selected join point, with a minimum of two possible execution paths. If the pointcut element has a connection with another FlowObject after it, this connection is added to the gateway as the default gate.
- *around*: This option is not supported by this pattern.

exceptions: If the join point is an end event, the *after* option is forbidden.

example: In figure 5.11, the pointcut is activity A. An exclusive choice pattern is defined with two new execution threads to activities B and C.

Figure 5.11: Example of the weaving of an exclusive choice pattern into a process model

5.6.5 Merge Pattern

purpose: A process model is modified by introducing a convergence point in which only one of two or more execution paths is selected.

use: `Caller.Merge ([obj1, <condition1>],...,[objN, <conditionN>])`. A caller can be a task, subprocess, gateway or event.

implementation: The merge pattern is implemented using a *DataXor* gateway.

- *before*: The merge pattern is implemented before the selected join point is returned by the pointcut. It selects two or more IncomingFlows connections representing execution threads. If the pointcut element has a connection with other FlowObjects before it, this connection is added to the gateway.
- *after*: This option is not supported by this pattern.
- *around*: This option is not supported by this pattern.

exceptions: If the join point element is a start event, the *after* option is forbidden.

example: In figure 5.12, the pointcut is activity A. The merge pattern is implemented by adding two new execution paths from activities B and C.

Figure 5.12: Example of the weaving of a merge pattern in a process model

5.6.6 Multiple Choice Pattern

purpose: A process model is modified by introducing a divergence point in which multiple execution paths can be selected.

use: `Caller.MultiChoice ([obj1, <condition1>],...,[objN, <conditionN>])`.
A caller can be a task, subprocess, gateway or event.

implementation: The multiple choice pattern is implemented with the addition of an *OR* gateway.

- *before:* This option is not supported by this pattern.
- *after:* The multiple choice pattern is implemented after the join point, and two or more gates to represent the parallel execution threads. Each gate has an associated condition expression, which will be evaluated at execution time. If the pointcut has a connection with other FlowObjects after it, this connection is added to the gateway as the default option.
- *around:* This option is not supported by this pattern.

exceptions: If the join point is an end event, the *after* option is forbidden.

example: In figure 5.13, the pointcut is activity A. The merge pattern is formed with two new execution threads to activities B and C.

5.6.7 Deferred Choice Pattern

purpose: A process model is modified by introducing a divergence point in which only one of several execution paths is selected, but the decision can only be made during the execution of the process.

use: `Caller.DeferredChoice ([obj1, <condition1>],...,[objN, <conditionN>])`.
A caller can be a task, subprocess, gateway or event.

Figure 5.13: Example of the weaving of a multiple choice pattern in a process model

implementation: The deferred choice pattern is implemented using a complex gateway.

- *before*: This option is not supported by this pattern.
- *after*: The deferred choice pattern is implemented after the join point, and two or more gates are connected to it. In this case, the gates are connected to intermediate events. The decision of which execution path will be selected is only resolved at execution time and can be different for each instance of the process.
- *around*: This option is not supported by this pattern.

exceptions: If the join point is an end event, the *after* option is forbidden.

example: In figure 5.14, the pointcut is activity A. The merge pattern is added with two new execution paths to activities B and C.

5.6.8 Arbitrary Cycles Pattern

purpose: A process model is modified by introducing a cycle.

use: Caller.ArbitraryCycles ([obj1, <condition1>],[obj2, <condition2>]).
A caller can be a task, subprocess, gateway or event.

Figure 5.14: Example of the weaving of a deferred choice pattern in a process model

implementation: The arbitrary cycle pattern is implemented using two *DataXor* gateways.

- *before*: The *DataXor* gateways are weaved before the join point element returned by the pointcut, forming a cycle around the element connected before the pointcut. This pattern is weaved only if the previous pointcut is not a start event.
- *after*: If the pointcut is not an end event, the cycle is weaved around the element following the pointcut.
- *around*: A cycle is woven around the join point returned by the pointcut.

exceptions:

example: Figure 5.14 presents the three possible scenarios. New gateways, but not activities, are added to the model.

5.7 Viewpoint Interference

Viewpoints are used to represent, in a modular and independent way, concerns that otherwise will be scattered along the process model definition. However, this independence means that viewpoints can interfere with each other.

Figure 5.15: Example of the weaving of an arbitrary cycle pattern in a process model

[Zha06] et al. classify dependencies between aspects into three categories: *follows*, *hiddenby* and *dependenton*.

In *follows*, two aspects are defined on the same joinpoint, which establishes a precedence order. For example, if we have two aspects, *Aspect1* and *Aspect2*, and it is necessary to force the execution of *Aspect1* before *Aspect2*, the definition of the relationship is *Aspect2 follows Aspect1*. *Hiddenby* describes a scenario in which the occurrence of one aspect in a join point automatically invalidates the second aspect: *Aspect2 ishiddenby Aspect1*. *Dependenton* defines a situation in which the execution of one aspect takes place only in join points where the other aspect is defined. For example, the clause *Aspect2 isdependenton Aspect1* expresses that *Aspect2* is enabled only in join points where *Aspect1* is also present [Zha06].

Another approach, presented in [DSBA05], proposes the modeling of aspects relationships following a resource model, in which join points are compared with resources in databases and distributed systems domains, and are shared among several processes. In this approach, advices are expressed as a sequence of instructions that are analyzed to detect conflict situations corresponding to previously defined rules.

Both approaches permit the detection of interference between aspects, how-

ever, they have some limitations. The first approach demands the explicit definition of the dependence between aspects, assuming that these relationships are known at modeling time. We believe that this is not always the case and that doing this creates a dependency between the aspects. The second approach, based on an analysis of the models, represents an expensive solution and does not guarantee that all conflicting relationships are detected, because it depends on the correct definition of the conflict rules.

We propose an approach that is a hybrid between the explicit definition of some types of interference between viewpoints and the analysis of viewpoint models to detect conflicting situations. By making these dependencies explicit, process designers can manually define the necessary corrections to solve the problem.

5.7.1 Explicit Interference

In our approach, a viewpoint can define different adaptations of a target process model. For example, consider the viewpoints used in the examples (verification and monitoring). We define that after the execution of the codification activity, two new activities are added, review and inspection, which are related to the verification concern. In the same viewpoint, a new adaptation is defined on the codification activity that adds a monitoring activity to records its finalization time. In this case, there are two different adaptations defined on the same joinpoint. To make the solution to this interference explicit, the AspectViewpoint language offers the primitive *follows*, which gives an order for the execution of the viewpoints. We define that *monitoring follows verification*, so the monitoring strategy will be weaved into the target process after the verification strategy. These are the same terms proposed in [Zha06].

5.7.2 Detected Interference

Explicit interference allows process designers to define the desired behavior in an interference situation. However, if the interference is present as a result of the interaction of multiple viewpoints, process designers cannot specify the order of weaving of the strategies. Only after the definition of all viewpoints

is it possible to consolidate and detect potentially interference situations, by performing an analysis of the models associated with the viewpoint. This approach will be explained in detail in chapter 6.

5.8 The Verification Example

To illustrate the AspectViewpoint language and the application of workflow control patterns as weaving strategies, we revisit the verification viewpoint example presented previously. This viewpoint is used to review and inspect the work products produced during implementation. For simplicity, we will present only the code associated with the review and inspection of the code product.

Figure 5.16 presents a scenario in which the weaving of the *DLDReview* and *DLDInspection* activities is already done, concentrating on the *Codification* activity.

Figure 5.16: Implementation process model before the review and inspection of the codification activity

The objective of the verification viewpoint is the review and inspection of the products produced during the *Codification* activity. The concern represented in this viewpoint states that after the execution of the activity, a review of the code must be followed by the inspection of the code. If during the inspection

activity, the reviewers find defects, the codification activity must take place again. This cycle is repeated until all unconformities are eliminated.

In response to this concern, the process designers produce a viewpoint that detects the points in the process model that the concern affects and define the required adaptations to the model.

The code presented in listing 5.3 corresponds to a possible implementation of the verification viewpoint. In line 1, viewpoint *Verification* in category *Engineering* is created. The name of the target process is *TechnicalSolution*.

The viewpoint definition contains one pointcut, called *findCodification*. It is composed of two selection methods: the first is the method *target* used to capture the context object; the second is *name*, required to select all tasks called *Codification*(line 3 of listing 5.3).

Once the pointcut is defined, the strategies are defined. For the purpose of illustrating an interference scenario, we define three different strategies: *addCycle*, *addInspection* and *addReview*.

The first strategy, *addCycle*, presented in line 6 of listing 5.3, is associated with the *findCodification* pointcut with the option *around*. The only instruction defined in this strategy is the use of the workflow pattern *ArbitraryCycle*, shown in line 8. This instruction is to weave an arbitrary cycle around the join point, which is the task *Codification*. The two parameters that call the method represent the conditions used in the gates of the *Xor* gateway to form the cycle. The first condition is the finalization condition, whereas the second condition defines the cycle. The intermediate state of the process model after the execution of this strategy is presented in figure 5.17.

The second strategy is *addInspection*. This strategy is used to weave the *SrcInspection* activity in the pointcut *findCodification* using the *after* option. In this strategy, we create a new activity of type *Task*, called *SrcInspection*, presented in line 15. This activity is weaved following a sequence pattern after the selected join point (the *Codification* activity). The intermediate state of the process model after the weaving of this activity is shown in figure 5.18.

The definition of this strategy also presents an interference scenario between *addInspection* and *addCycle*. Both strategies define the adaptation of the *Codification* activity. Since there is no a predefined order for the execution

```

1 viewpoint Verification
2 affiliation Engineering
3 targetprocess TechnicalSolution
4 {
5   pointcut findCodification (Task task) : target (task) &&
6                                           name ("Codification");
7
8   //This strategy weaves the arbitrary cycle pattern
9   strategy addCycle (Task task):around findCodification (task)
10  {
11    task.ArbitraryCycles ([task,"defects=0"],[task,"defects>0"]);
12  }
13
14  // This strategy weaves the \emph{Inspection} activity
15  strategy addInspection (Task task):after findCodification (task)
16                                follows (addCycle)
17  {
18    Task inspection;
19    inspection=task.parent().newFlowObject("SrcInspection",
20                                           "Task","Normal");
21    task.Sequence ([inspection,""]);
22  }
23
24  // This strategy weaves the \emph{Review} activity
25  strategy addReview (Task task):after findCodification (task)
26                                follows (addInspection)
27  {
28    Task review;
29    review=task.parent().newFlowObject("SrReview","Task","Normal");
30    task.Sequence ([review,""]);
31  }
32 }

```

Listing 5.3: The verification viewpoint defined using the AspectViewpoint language

Figure 5.17: The implementation process after the weaving of the arbitrary cycle pattern

of the strategies, it is necessary to explicitly define it using the *follows* instruction, as shown in line 12. This instruction states that the *addInspection* strategy is executed after the *addCycle* strategy.

Figure 5.18: The implementation process after the weaving of the *SrcInspection* activity

The third strategy defined in this viewpoint is *addReview*, shown in line 20. In this strategy, we create a new activity called *SrcReview*, as shown in line 23, and then we define the sequence control pattern associated with the selected join point. The final process model is presented in figure 5.19.

Similar to the previous strategy, this case also introduces an interference between the *addReview* and *addInspection* strategies. The interference is present because the *findCodification* pointcut is used in both cases. Again, we use the *follows* instruction to establish the desired weaving order.

Figure 5.19: The implementation process after the weaving of the *SrcReview* activity

5.9 Summary

The AspectViewpoint language supports the definition of viewpoints in target process models. A viewpoint is defined using pointcuts, which are sets of join points, and strategies. The AspectViewpoint language allows the definition of both static and dynamic join points. Static join points are based on the selection of FlowObjects as defined by the BPMN specification: tasks, subprocess, gateways and events. Dynamic join points select states associated with the execution of the activities.

The strategies defined in the AspectViewpoint language are based on workflow control patterns. The language uses a subset of these patterns to define the adaptation of a process model with the addition of new elements and the modification of the execution control flow.

Chapter 6

Process Viewpoints

6.1 Introduction

A *viewpoint* represents the perspective from which a thing is viewed. One of the first uses of viewpoints was by Ian Sommerville [SKVS95], who was inspired by the Goal Question Metric (GQM) approach of Basili and Rombach [BCR94]. He defined a process viewpoint as “an encapsulation process description which includes a process model, the sources of model information, and the questions which were posed to derive that process model”[SKVS95].

In our work, we use the term *viewpoint* to define the perspective from which a process is viewed by the stakeholders. Our objective is to support the definition of viewpoints and the integration of these viewpoints into a target process.

The approach we use is model-driven development. Specifically, our work is based on three metamodels: process, viewpoint and conciliation. Figure 6.1 presents an overview of these metamodels and the relationships between them.

The process metamodel defines a target process model. The viewpoint metamodel defines the viewpoints representing concerns expressed by the stakeholders of the process. The conciliation metamodel integrates the different viewpoints and detects conflicts between them.

Figure 6.1: Overview of the metamodels

This chapter explains these metamodels and the relationships between them. Section two presents the process metamodel, section three explains the view-points metamodel, section four presents the conciliation metamodel, and section five summarizes the chapter with an integration of the metamodels used to support the definition and execution of process viewpoints.

6.2 The Process Metamodel

We base the definition of our process metamodel on the BPMN [Gro06] presented in section 2.2 (page 13). From the BPMN specification, we have inferred a process metamodel, which was inspired by the work of [KV06] and [MMA07]. Figure 6.2 presents the elements and relationships in this metamodel.

The process metamodel plays two important functions in our approach. First, it supports the definition of target process models conforming to the BPMN metamodel. Second, it is used during the transformation of target process models into platform-specific processes to be executed in a workflow engine. In the following sections, we present the elements of the metamodel and

Figure 6.2: BPMN metamodel [KV06] [MMA07]

an example of its use in defining a target process model. The use of the metamodel during the transformation of the target process is illustrated in chapter 7.

6.2.1 Metamodel Structure

Figure 6.2 shows the main elements of a BPMN metamodel.

Business Process Diagram

The business process diagram is the main element of the metamodel; it acts as a container for the processes defined in the diagram.

Object

An *Object* represents any of the graphical symbols of the BPMN language. This element defines the common root of *Flow Objects*, *Connecting Objects*, *Swimlanes* and *Artifacts*.

Process

The process entity represents the participants in a business process diagram. Each process is associated with a *Pool* element, which facilitates the modeling of business-to-business and collaboration scenarios. This entity is composed of *Object* elements, called *graphicalElements* in the metamodel.

Flow Objects

A *Flow Object* is derived from an *Object* and is used to represent the execution control flow elements of a process. Its main subclasses are *Event*, *Activity* and *Gateway*. *Events* are divided into start, intermediate and end classes, which model events occurring during the execution of the process. The *Activity* entity represents a piece of work inside a process. It can be classified into *Subprocess* and *Task* elements. Finally, a *Gateway* represents diverging and converging points in the process and is composed of *Gates*. *Gateways* are divided into *And*, *Or*, *Xor* and *Complex* classes.

Connecting Objects

This entity models the three methods defined by the BPMN to connect *Objects*: *SequenceFlow*, *Association* and *MessageFlow*. *SequenceFlow* connections are between *Flow Objects*, and they define the execution sequence of the process. *Association* connections are between *Activities* and *Artifacts*, and they model the data flow. *MessageFlow* connections are between *Swimlane* elements, and they model the communication between processes.

Swimlane

The *Swimlane* element differentiates process definitions in one business process diagram, or models the responsibility of execution of activities. For multiple process definitions, the *Lane* entity is used. For modeling responsibilities, the *Pool* element is part of a *Lane* element and is a mechanism to model sets of activities assigned to one particular role or participant.

Artifact

This class models additional information in a process model. It is divided into three main subclasses: *Data Object*, *Annotation* and *Group*. However, the BPMN specification does allow the definition of other types of *Artifacts*.

6.2.2 Example of Use

Figure 6.3 shows a process model, developed using the BPMN metamodel, that represents the implementation phase of a software development process. In this example, the following elements can be identified: tasks, gateways, incomingflows, sequenceflows and gates. Tasks represent pieces of work inside a process. Gates model points in a process where control of the execution must be expanded into several options (splits) or collapsed into one execution thread (merge). When a gateway has several incoming execution paths, they are *IncomingFlows*. If the gateway is used to split the execution of a process, each of these paths is called a *Gate*.

During the modeling of the process in figure 6.3, we use the BPMN task element to define the following activities: *CreateDLD* produces the detailed design of the system, then *CreateUTPlan* designs the test cases and unit tests. The *Codification* activity represents the code that implements the detailed design. The *UnitTesting* activity executes a unit test on the code produced during *Codification*. Any defects found during *UnitTesting* are registered in a defect tracking application (*RegisterDefects*). Finally, the *CreateBL* task checks the tested code in a version management application and registers it in a base line system.

Figure 6.3: Example of the implementation phase of a development process model defined using the BPMN language.

In this figure, the execution control of the process is modeled using SequenceFlows and Gates. SequenceFlows indicate that the finalization of a task is followed by the initialization of another task, for example, the *CreateUTPlan* is initialized when the *CreateDLD* task is finished. The control flow is determined with gateways. In the example, there are two situations that require the use of gateways. The first involves the *Codification* task and represents a point where it is necessary to select one of two execution paths to continue. The task can be executed in two different manners: the finalization of the *CreateUTPlan* task or the finalization of the *RegisterDefects* task. This situation is modeled using an *Xor* gateway, which takes the two tasks as *IncomingFlows* and defines the *Codification* activity as a gate.

Similarly, an *Xor* gateway can represent an exclusive split of the execution of the process. When the *UnitTesting* activity finishes, there are two possibilities: if the tests do not report any defects, the next task must be *CreateBL*. If defects are detected, the *RegisterDefects* task is executed.

The process model in 6.3 will be used as a base example in rest of this document as the target process on which to define different viewpoints. The BPMN metamodel gives us a technology-independent definition of the control flow of the process.

6.3 The Viewpoint Metamodel

In the introduction to this chapter, we defined a viewpoint as the perspective from which a particular stakeholder interacts with a target process in terms of the concerns relevant to him. We defined a viewpoint metamodel to aid process designers with the definition of viewpoints in target process models. In figure 6.4, we present the main elements of this metamodel.

Figure 6.4: The viewpoint metamodel

In the following sections, we explain the viewpoint metamodel and give an example of its utilization.

6.3.1 Metamodel Structure

The viewpoint metamodel supports the definition of crosscutting concerns related to process models. We base this metamodel on the Aspect-Oriented Modeling (AOM) principle. The viewpoint metamodel is inferred from the the AspectViewpoint language, therefore, the entities of the metamodel correspond directly to the elements of the language.

Viewpoint

The *Viewpoint* element represents a crosscutting concern expressed in a target process model. A *Viewpoint* is composed of join points in the process model that are affected by the concern and the strategies containing adaptations to the process model. A viewpoint has several attributes: name, affiliation (which defines families of viewpoints) and the target process associated with it.

Pointcut

The *Pointcut* element defines sets of join points that belong to the process model and are filtered using selection methods.

SelectionMethod

A *SelectionMethod* element represents the criteria used during the selection of join points in a pointcut definition.

Strategy

A *Strategy* models the actions or modifications introduced by the concerns in a target process model. Strategies are composed of statements, which represent the actions required to implement a concern. Statements can be used to create new process elements (NewStatements) or to change the behavior of the process according to workflow control patterns (PatternStatements).

PatternStatement

A *PatternStatement* represents the workflow control patterns supported by the model. It modifies the process model in a given join point according to the semantics of the respective workflow pattern.

6.3.2 Example of Use

To illustrate the viewpoint metamodel, we revisit the verification example. Figure 6.5 shows the target process for the verification viewpoint. This process defines the implementation phase of a software component.

Figure 6.5: Target process model associated with the verification viewpoint.

In this example, the concern defined by the stakeholders is the verification of some artifact produced during the implementation phase. Specifically, the concern states that after the finalization of the design activity, two new activities (*DLDReview* and *DLDInspection*) need to be included.

DLDReview is executed after *CreateDLD*, and then *DLDInspection*, following a workflow sequence pattern. The concern also defines similar behavior for the *Codification* activity. After its finalization, the activities *SrcReview* and *SrcInspection* must be executed, following a workflow sequence pattern. The process model obtained after the inclusion of the verification concern is presented in figure 6.6.

The verification viewpoint is defined using the AspectViewpoint language. As mentioned, the viewpoint metamodel was inferred from the grammar of the AspectViewpoint language; hence, a viewpoint definition created using the language is actually a viewpoint model. In chapter 5, we presented the language associated with the implementation of the verification viewpoint. Here, we present a version of this example based on the viewpoint model. Figure 6.7 presents an object diagram that represents a partial view of the verification model. This diagram shows an example of the viewpoint model, which will be used later to perform the weaving of this model with the target process model.

Figure 6.6: The implementation process model after the inclusion of the verification viewpoint.

Figure 6.7: Object diagram associated with the verification viewpoint model.

6.4 The Conciliation Metamodel

The conciliation metamodel is the third component in our solution, and it plays two important roles. First, viewpoint models, defined by the AspectViewpoint language, are transformed into language-independent models. Second, two or more viewpoints can be defined around the same join point in a target process model. This situation can produce interferences between the advices defined in each viewpoint. The conciliation metamodel is used to detect and solve interference problems between viewpoints. Figure 6.8 presents the components of the conciliation metamodel.

Figure 6.8: The conciliation metamodel.

6.4.1 The Process Weaving Engine Metamodel

The objective of the Process Weaving Engine (PWE) metamodel is to offer a platform-independent mechanism to represent process model aspects. This

means that aspects models defined using languages similar to AspectViewpoint could be transformed to the PWE metamodel to obtain a neutral representation. Additionally, the PWE metamodel serves as a pivot model to transform models defined using one aspect language into another.

The PWE metamodel follows the aspect model weaving proposed by Schauerhuber ([SA06b] [SA06a]). We reworked Schauerhuber's metamodel to obtain a process-oriented version that is suitable for our domain.

Metamodel Structure

TargetProcess: This element represents the target process model around which aspects are defined. It is defined as a set of aspects and interferences.

Aspect: This element represents a concern expressed by the stakeholders of the target process. It is composed of pointcuts and adaptation rules.

Pointcut: This element defines the method used to select particular join points in the target process model. Each *SelectionMethod* is defined as a set of join points associated with elements of the process model definition.

AdaptationRule: This element defines the mechanisms used to adapt the target process model based on the concerns defined in it. In terms of the AspectViewpoint language, an *AdaptationRule* is equivalent to a *Strategy*. Adaptation rules are composed of adaptations, representing atomic modifications of the target process model.

Adaptation: This element represents each of the modifications made to the target process model. The *Adaptation* entity is divided into two elements: *SimpleAdaptation* and *CompositeAdaptation*. *SimpleAdaptations* are classified into *StructuralAdaptations* and *BehavioralAdaptations*. *StructuralAdaptations* represent changes in the structure of the process model in terms of the *ValueViewpoints* of the viewpoint metamodel. *BehavioralAdaptations* represent data and control flow adaptations in the target process model, equivalent to *ComplexViewpoints* in the viewpoint metamodel.

6.4.2 Example

To illustrate the use of the PWE metamodel, we present a simplified version of it resulting from the transformation of the verification viewpoint shown in figure 6.7. This figure shows some of the elements of the corresponding PWE model, which represents the verification viewpoint in a platform-independent fashion. As explained previously, this model is used as a pivot to translate aspect models expressed in different languages. In particular, we use this model to translate the AspectViewpoint language into the Embedded Constraint Language (ECL) [LGZ06]. This step will be explained in detail in chapter 7.

Figure 6.9: Example of a model constructed from the transformation of the verification viewpoint model.

6.4.3 The Interference Resolution Metamodel

As explained in chapter 5, the viewpoint model considers two mechanisms to tackle interference between aspects. The first is the use of the primitive *follows* to indicate the order in which the aspects must be weaved to guarantee the consistency of the process model. However, this approach is only valid when the aspects are known in advance by the process designer.

When multiple viewpoints are expressed in a target process model, process

designers are unaware of potentially conflicting situations. To detect interferences of this type, we use the conciliation metamodel, in particular, the resolution metamodel. This strategy takes advantage of the transformation of the viewpoint model into the PWE model. During this transformation, we create a dependency graph, following the resource model, to explicitly define the join points involved in each adaptation and the type of conflict detected.

6.4.4 Metamodel Structure

In the following sections we explain each element of the resolution metamodel.

Interference

This element models any type of interference detected during the conciliation of viewpoints. It is related to the *AdaptationRule* of the PWE metamodel. These relationships, called *intervened* and *intervener*, model the conflicting scenarios between adaptation rules. Each *Interference* is associated with a resolution mechanism that indicates how the conflicting adaptation rules are weaved.

Resolution

This element represents a resolution to an interference conflict between adaptation rules. A resolution can be divided into one of two elements: incompatibility and precedence.

Incompatibility

In this type of resolution, an incompatibility relationship exists between the adaptation rules. This resolution states that the *intervened* rule is not taken into account in the presence of the *intervener* rule. Consider two adaptations in a target process. In the first adaptation, the *UnitTesting* activity is deleted from the target process model. At the same time, the second adaptation adds a new activity after the *UnitTesting* activity. These rules are incompatible,

and the resolution entity states that the intervened rule will be disabled in the presence of the intervener rule.

Precedence

In this type of resolution, a precedence relationship exists between adaptation rules. This resolution states that the intervened rule is executed after the intervener rule. Consider the scenario in which in two adaptation rules are defined in a target process. The first adaptation adds a new activity, *ReviewCode*, to review the code produced during the coding activity. The second adaptation measures the execution time of all review activities during the implementation phase. In this scenario, the second adaptation needs to be weaved after the addition of the review code.

6.4.5 Example

To illustrate a resolution metamodel, we continue with the verification viewpoint example, where two interferences are defined. First, the *addCycle* rule associated with the join point activity *Codification* interferes with the *addInspection* rule. The interference is resolved using a precedence resolution, which indicates that the *addCycle* rule is executed first. The second interference is defined between the *addInspection* and *addReview* rules. In this interference, the resolution method adopted is also a precedence, which indicates that the *addInspection* rule is executed first.

Figure 6.10 illustrates the resolution metamodel for the verification viewpoint example.

6.5 General Overview

In this section, we present an overview of the metamodels explained in this chapter, as well as how and when they define and execute process viewpoints. In chapter 7, we will explain in detail the technologies used in our implementation.

Figure 6.10: Example of a resolution model constructed from the transformation of the verification viewpoint model.

Figure 6.11 summarizes the metamodels used in each of the phases that are required to define and execute viewpoints. The metamodels are in the rows of the table, and the phases in the columns. The first step defines the target process model using the BPMN metamodel. The result of this phase is a process model defined in the BPMN language.

The second step is the definition of viewpoints relative to the target process, which is accomplished with the AspectViewpoint language. The result of this step is a set of viewpoint models that conform to the viewpoint metamodel.

The third phase is the conciliation of the viewpoint models. This step is done with two parallel activities: first, the viewpoint models are transformed into technology-independent weaving strategies that conform to the PWE metamodel. The viewpoint models are then analyzed to find interference problems using the interference resolution metamodel. The resulting model is used by the PWE model to organize the weaving strategies.

The fourth step is the weaving of the target process model and the viewpoints expressed in the PWE model. The result of this phase is a BPMN process model.

Finally, in the fifth step, the woven process model is transformed into a platform-specific executable process model, like BPEL or Cumbia.

Figure 6.11: General view of the metamodels used during the definition and execution of process viewpoints.

6.6 Summary

In this chapter, we presented a model-oriented approach proposed to support the definition and weaving of viewpoints in target process models. Our work is based on the definition of three metamodels: process, viewpoint and conciliation.

The process metamodel is based on the BPMN process specification. The viewpoint metamodel defines aspect models on a process model. It is composed of pointcuts and strategies, which define the desired adaptations to a process model. The conciliation metamodel plays two important roles. First, it represents a technology-independent process weaving model, used as a pivot model to translate viewpoint models defined with the AspectViewpoint language to platform-specific weaving engine models. Second, this metamodel detects interference situations resulting from the definition of multiple viewpoints on join points.

Chapter 7

Definition and Execution of Viewpoints

7.1 Introduction

This chapter presents the technologies used during the implementation of our aspect-oriented model proposal. In particular, we detail the languages, tools and frameworks used for the definition, transformation and execution of viewpoints on workflow processes.

To facilitate our presentation, we organized this chapter following the process activities presented in figure 7.1.

Following the sequence in figure 7.1, section two explains the implementation adopted for the definition of the target process model. Section three presents the technology used to support viewpoint definition. Section four discusses the transformation of viewpoint models into conciliation models. Section five describes the technologies used to support the weaving of the target process model and the viewpoints. Finally, section six presents the technologies selected to execute the woven process.

Figure 7.1: The definition and execution of multiple viewpoints in process models.

7.2 Process Modeling

This work is based on the premise that organizations are in a state of continuous process improvement. This premise implies that processes need to be clearly defined and well understood by the members of the organization, so that changes may be proposed and concerns expressed.

Our main motivation in this work is to support the definition of concerns, in a non-intrusive way, on a target process. However, to achieve this objective, we need to be clear about the process to be modified and its activities, participants and products. We start with a description of the tools and techniques that are used to define target processes in our implementation strategy.

In previous chapters, we presented examples of process models using the BPMN notation. The definition of these models requires that the selected process editor meets some specifications. First, the editor must support the definition of processes using BPMN. Second, the editor must offer more than a graphical visualization of the process: the result of the editing must be a process model that must have a well-defined structure, conform to a known metamodel, and be able to be traversed, queried and transformed.

Taking these requirements into account, we selected the Generic Modeling

Environment (GME) editor to define both the process metamodel used to describe our process models and the process models themselves.

GME provides a meta-metamodel (MetaGME) to support the definition of domain-specific metamodels. These metamodels can be registered into GME and used to define models that conform to the new metamodels. Our implementation strategy is based on two steps: first, we define a process metamodel; second, we define process models that conform to this metamodel.

7.2.1 The Process Metamodel

In GME, a metamodel can be defined following the separation of concerns principle. According to the principle, the metamodel is composed of four different views: class diagram, visualization, attributes and constraints. During the editing of the metamodel, each of these views can be used and visualized independently. When the metamodel is processed by GME, these definitions are integrated into one metamodel definition.

In the *ClassDiagram* view, the main elements of the metamodel, as well as the relationships between them, define the elements of the MetaGME metamodel: models, atoms and connections. Figure 7.2 presents a partial view of the class diagram of the BPMN process metamodel developed with GME.

After the class diagram has been created, we define the visualization diagram. GME requires that a metamodel has at least one *aspect* view. The term *aspect* is used here to express views of the models created with the metamodel. For example, we can define a view to select only the atoms present in a metamodel. In our case, we define only one aspect, *MainView*, to visualize all elements of the BPMN models created with this metamodel. Figure 7.3 presents a partial view of the visualization diagram of the BPMN process metamodel developed using the GME tool.

The next step is the definition of model attributes in the class diagram: atoms and connections. In GME there are three types of attribute: *BooleanAttribute*, *EnumAttribute* and *FieldAttribute*. Figure 7.4 shows the definition of the *Task* element attributes in the BPMN metamodel.

The last step in the creation of a metamodel is the definition of its con-

Figure 7.2: Partial view of the class diagram of the BPMN process metamodel developed with the GME tool.

Figure 7.3: Partial view of the visualization diagram of the BPMN process metamodel developed with the GME tool.

Figure 7.4: Partial view of the attributes diagram of the BPMN process metamodel developed with the GME tool.

straints. In GME, a constraint element is associated with an entity of the metamodel, and a property editor is used to select the triggering event for the validation, for example, the modification of the model and the OCL expression associated with the validation of the constraint. Figure 7.5 presents an example of a constraint definition. A verification condition is associated with the *XORGateway* elements of the BPMN metamodel to guarantee that they always have one default gate.

7.2.2 The Process Models

When the four views are completed, the metamodel is validated and generated by GME. The resulting metamodel is registered within GME to be used as a new metamodel during the definition of models.

We have created different process models to test our metamodel. For example, the process model shown in figure 7.6 presents the implementation phase of the development of a software component.

Our current implementation is not fully compliant with the BPMN speci-

Figure 7.5: Partial view of the constraint diagram of the BPMN process metamodel developed with the GME tool.

Figure 7.6: Example of a process model definition using an implementation of the BPMN specification for the GME tool.

cation, mainly because of limitations of the GME tool to represent *swimlane* elements. Consequently, we currently support only one process model per diagram.

7.3 Viewpoint Modeling

The second step in the process is the modeling of the viewpoints representing concerns expressed by the stakeholders. Similar to the process modeling strategy, we define a metamodel to support the definition of viewpoint models.

We selected the Eclipse Modeling Framework (EMF) [Ecl06] to define the viewpoint metamodel. EMF includes a meta-metamodel, Ecore, for describing metamodels and runtime support for them, including change notification, persistence support with XMI serialization, and a reflective API for manipulating EMF objects [Ecl05].

Our viewpoint definition implementation is supported on the OpenArchitectureware (oAW) framework [OAW07]. This decision was made for two reasons. First, this framework bases its functionality on a model-driven approach (based on EMF). Second, the framework offers the option of generating a specialized plug-in editor for the Eclipse development platform [IBM07a].

In our implementation, viewpoints are expressed using the domain-specific aspect language, AspectViewpoint. Each program written in this language is also a viewpoint model. In the following sections, we present the definition of our language and its implementation using the OpenArchitectureware (oAW) framework.

7.3.1 The AspectViewpoint Grammar

We defined an extended Backus-Naur Form (EBNF) for the AspectViewpoint language using Xtext [Eff07b], a specialized language for the definition of domain-specific language (DSL) syntaxes provided by the oAW framework.

```
1 Viewpoint :
2     "viewpoint" name = ID
3     "affiliation" affiliation = ID
4     "targetprocess" targetprocess = ID
5     "{"
6     (pointcuts+=Pointcut)+
7     (strategies+=Strategy)*
8     "}" ;
```

Listing 7.1: Example of the definition of the AspectViewpoint’s grammar using the Xtext language.

The Xtext language offers two basic functionalities: the mechanisms to define a DSL and the generation of a parser to validate programs created with this language.

DSL Grammar Definition

The main element in the Xtext grammar language is a *Rule*. Its main function is to describe elements of DSL grammars [Eff07b]. For example, listing 7.1 presents the definition of the main rule of the AspectViewpoint language. Here, a viewpoint is defined with the reserved word *viewpoint* followed by an identifier indicating its name and the reserved word *affiliation* followed by an identifier indicating the affiliation of the viewpoint. Then, the reserved word *targetprocess* is followed by an identifier associated with the name of the target process. Finally, a viewpoint is composed of one or more *pointcuts* and zero or more *strategies*. *Pointcuts* and *strategies* are defined using similar rules.

Rules can be classified into three types depending on their structure and purpose: simple, abstract and enum.

Simple Rules: Simple rules associate the name of a concrete metatype with the name of the rule. For example, in line 1 of listing 7.1, the viewpoint rule is defined as the metatype *Viewpoint* followed by a colon.

```

1 Abstract Pointcut :
2     UnnamedPointcut | NamedPointcut
3 ;

```

Listing 7.2: Example of an abstract definition rule.

```

1 enum BpmnType :
2     Task="Task" | SubProcess="SubProcess" | Artifact="Artifact"
3     | Gateway="Gateway" | Event="Event" | Sequence="SequenceFlow"
4 ;

```

Listing 7.3: Example of an enum definition rule.

Abstract Rules: Abstract rules allow the definition of different types of elements inside a rule. Listing 7.2 presents the definition of the abstract rule *Pointcut*, which states that a pointcut may be named or unnamed.

Enum Rules : Enum rules are useful for the definition of types in a DSL. Listing 7.3 presents the enumeration rule *BpmnType*, which states that a BPMN type can be any of the types defined in line 2.

Appendix A presents details of the AspectViewpoint language grammar.

The Parser Generation

With the grammar file containing the EBNF defined in the previous step, Xtext produces a parser for the DSL. Xtext also generates a metamodel that corresponds to the DSL and an Eclipse plugin-editor for the language. The parser can be enriched with syntactic and semantic extensions using other oAW tools to facilitate the definition of semantic constraints checked during editing.

7.3.2 The Viewpoint Metamodel

As explained previously, the grammar of the AspectViewpoint language is used to generate the viewpoint metamodel, using the Xtext language. This

metamodel conforms to the Ecore metamodel and implements the metamodel presented in section 6.3.

Figure 7.7 presents a partial view of the viewpoint metamodel generated with the oAW framework.

Figure 7.7: Partial view of the viewpoint metamodel generated by the oAW framework. This metamodel conforms to the Ecore meta-metamodel.

7.3.3 Viewpoint Models

When a viewpoint is defined using the AspectViewpoint language, it is processed by the parser to produce a viewpoint model that conforms to the viewpoint metamodel created from the AspectViewpoint grammar.

Figure 7.7 presents a verification viewpoint edited with the plug-in editor generated by the oAW tool. Figure 7.9 presents the XMI file associated with the viewpoint model obtained after the parsing of the verification viewpoint.

```

1 viewpoint Verification affiliation Engineering targetprocess TechnicalSolution {
2
3     pointcut findCodification (Task task) : target (task) && name ("Codification");
4
5     // Weaves the arbitrary cycle pattern
6     strategy addCycle (Task task) : around findCodification (task) {
7
8         task.ArbitraryCycles ([task, "defects=0"], [task, "defects > 0"]);
9     }
10
11    // Weaves the Inspection Activity
12    strategy addInspection (Task task) : after findCodification (task) follows (addCycle) {
13        Task inspection;
14
15        inspection = task.parent().newFlowObject ("SrcInspection", "Task", "Normal");
16        task.Sequence ([inspection, ""]);
17    }
18
19    |
20    // Weaves the Review Activity
21    strategy addReview (Task task) : after findCodification (task) follows (addInspection) {
22        Task review;
23
24        review = task.parent().newFlowObject ("SrReview", "Task", "Normal");
25        task.Sequence ([review, ""]);
26    }
27 }

```

Figure 7.8: The verification viewpoint model defined using the plug-in AspectViewpoint editor.

Figure 7.9: The verification viewpoint model visualized as a XMI file.

7.4 Viewpoint Conciliation

The next step is the integration of viewpoints into one model to analyze their interdependencies and schedule their weaving with the target process model. The result of this process is the conciliation model.

The strategy followed during the implementation of the conciliation model is similar to that of the viewpoint model. The conciliation metamodel is defined, then conciliation models are created that conform to this metamodel.

To define the conciliation metamodel, we create an UML class diagram modeled with the MagicDraw tool [NoM06]. This model is exported as an XMI file and then imported by the oAW framework. We use the *uml2ecore* utility provided by oAW to produce a conciliation metamodel that conforms to the Ecore meta-metamodel.

Figure 7.10 shows the conciliation metamodel defined as an UML class diagram. Figure 7.11 shows the Ecore metamodel obtained after the use of the *uml2ecore* tool.

Figure 7.10: The conciliation metamodel defined as an UML class diagram.

To produce the conciliation model, viewpoint models are transformed using the *Extend* language [Eff07a], which is also part of the oAW framework.

```

1 create pwe::Aspect createAspect (Viewpoint vp):
2     setName(vp. affiliation) ->
3     setPointcut (vp.createPointcut()) ->
4     rules.addAll (vp.allStrategies().createRule()) ->
5     dependencies.addAll (vp.allDependencies().createDependency())
6 ;

```

Listing 7.4: pointcut definition

“This language provides the possibility of defining rich libraries of independent operations, and no-invasive metamodel extensions based on either Java methods or oAW expressions” [Eff07a].

We use the *Extend* language to define a set of rules for transforming a viewpoint model into a conciliation model. The algorithm of the transformation is

1. The *Viewpoint* element of the viewpoint model is transformed into a *TargetProcess* element of the conciliation model.
2. Each *Strategy* element of the viewpoint model is transformed into an *AdaptationRule* element of the conciliation model.
3. Each *NewStatement* element of the viewpoint model is transformed into a *StructuralAdaptation* element of the conciliation model.
4. Each *PatternStatement* element of the viewpoint model is transformed into a *CompositeAdaptation* element of the conciliation model.
5. Each *SelectionFunction* element of the viewpoint model is transformed into a *SelectionMethod* element of the conciliation model.

Listing 7.4 shows an example of a transformation rule definition. The extension is *createAspect*, which creates an *Aspect* element in the conciliation model using a *Viewpoint* element of the viewpoint model.

Figure 7.12 presents the XMI format of the conciliation model obtained from the transformation of the verification viewpoint model.

The complete set of transformation rules is in appendix B.

Figure 7.12: XMI file representing the conciliation model obtained from the transformation of the verification viewpoint model.

7.5 Viewpoint Weaving

After the viewpoints are integrated into the conciliation model, platform-specific weaving strategies are generated.

Our implementation is based on the Crosscutting-Specification Aspect Weaving (C-SAW) engine [Sof07], a plug-in for GME [Led01] developed at the SoftCom laboratory at the Department of Computer and Informatics Sciences at the University of Alabama at Birmingham.

The C-SAW engine requires three inputs: the model to be transformed, the metamodel associated with the model to be transformed, and the weaving strategies containing the transformations.

C-SAW requires that the metamodel conforms to the MetaGME meta-metamodel. The model to be transformed must also conform to this metamodel. The weaving strategies must be defined in a particular language called the Embedded Constraint Language (ECL) [LGZ06], also developed and maintained at the SoftCom laboratory.

ECL describes model-to-model transformations of visual models. In ECL, the OCL [OMG06] is used to express model navigation and constraints in a modeling domain [LGZ06]. The language offers a mechanism of selection and aggregation of model elements, as well as transformation operations to modify these models.

The main modification element in ECL is a *strategy*, which is a "modular construct provided to specify elements of computation and the application of specific properties to the model entities"[LGZ06].

Generally, an ECL program consists of one or more strategies containing the transformations of the model, which can be applied by traversing a model and matching elements that satisfy a predicate [LGZ06].

7.5.1 Generation of Weaving Strategies

To use the C-SAW engine, the conciliation model is transformed into ECL strategies that will be used to transform the target process model.

The generation of ECL strategies is implemented using the *Xpand* language [Eff06] provided by the OpenArchitectureware framework. This language supports the generation of output files from the entities of an Ecore model. This process is supported by the use of templates, which are files containing instructions to process each element of the model.

We developed an Xpand template which takes the conciliation model as input. The model is traversed by searching each element to produce the associated ECL code according to the template definition.

In listing 7.4, a file is created with name `<TargetProcessName>.spc`. The name of the ECL file corresponds to the value of the *TargetProcess* element of the conciliation model.

We created a special strategy, called *main*, which is the first strategy invoked by the C-SAW engine during the model weaving process. This strategy triggers the calling of all the strategies generated in the previous step. We created one strategy for each pointcut found in the conciliation model. The order of the strategy generation is determined by the resolution model, to guarantee the precedence of the transformations.

For each *rule* in the conciliation model, an ECL strategy is generated. Each instruction containing a workflow control pattern is translated into a corresponding ECL code.

7.5.2 Model Weaving

Once viewpoints are translated from the conciliation model into ECL code, they are woven into the target process model by the C-SAW engine.

Figure 7.13 shows the technical solution target process before the weaving of the verification viewpoint. Figure 7.14 presents the target process obtained after the weaving of the models.

7.6 Process Execution

The final step in our strategy is the execution of the woven process model in a workflow engine. The BPMN does not propose a particular workflow

```

1 DEFINE main FOR TargetProcess
2 FILE name + ".spc"
3 FOREACH this.concern.rules AS rule
4     defines rule.name;
5 ENDFOREACH
6
7 strategy main()
8 {
9     declare context:model;
10    context := self;
11    strategy_this.concern.pointcut.name (context);
12 }
13 EXPAND ContextGeneration FOR this.concern
14     FOREACH this.concern.rules AS rule
15         EXPAND StrategyGeneration FOR rule
16     ENDFOREACH
17 EXPAND BPMNStrategies FOR this
18 ENDFILE

```

Listing 7.5: pointcut definition

Figure 7.13: The technical solution target process before the weaving of the verification viewpoint.

Figure 7.14: The technical solution target process after the weaving of the verification viewpoint.

engine. Thus, to execute the process model, we need to transform it into a technology-specific executable model.

We selected the Cumbia-XP workflow engine. In the following section, we detail the transformation implemented to obtain an executable Cumbia process model from a BPMN process model. This transformation was developed by Jorge Jimenez as part of his Master's thesis at the University of Los Andes [Jim07]. The semantic transformation presented below was extracted from [Jim07].

7.6.1 Transforming BPMN to Cumbia

The execution semantics of a BPMN process differ from those of a Cumbia process. In this section, we present the transformation strategy for each of the elements of the BPMN metamodel.

Task Transformation

A BPMN activity can be modeled directly as a Cumbia activity. The properties and attributes of a BPMN activity are stored in the memory of the Cumbia activity. The workspace of the Cumbia activity is determined according to the type of the BPMN activity 7.15.

Figure 7.15: Transformation of a BPMN Task.

LoopTask Transformation

The execution of a BPMN *LoopTask* is based on the *LoopCondition* expression. The workflow engine must execute the task if the Boolean value of the expression is evaluated *True*. The execution of this type of task also depends on the *TestTime* attribute, which supports the definition of until-loops and while-loops.

The transformation of a BPMN *LoopTask* into a Cumbia model depends on the value of the *TestTime* attribute. In a while-loop, the condition is stored into the memory of the Cumbia task. The evaluation of the condition is simulated with a *Dataflow* between two internal ports (figure 7.16).

Figure 7.16: Transformation of a *while* type BPMN LoopTask.

If the value of the *TestTime* attribute is set to until-loop, a new “dummy” activity is used to simulate a circuit with the transformed activity, as shown in figure 7.17.

Figure 7.17: Transformation of an *until* type BPMN LoopTask.

Multi-instance Tasks

There are two methods for transforming a BPMN multi-instance activity into a Cumbia activity, depending on the order of execution of the instances. If they are executed in sequence, the transformation follows the while *LoopTask*, as presented in the figure 7.16. If the instances are executed in parallel, the BPMN activity is translated into a Cumbia *MultiActivity*, as shown in figure 7.18.

Figure 7.18: Transformation of a Multi-instance BPMN Task executed in parallel.

SubProcess

A BPMN *Subprocess* is transformed directly into a Cumbia *Subprocess*. BPMN elements connected before and after the subprocess are translated to Cumbia

and then connected to the input and output ports of the Cumbia subprocess, respectively, as seen in figure 7.19.

Figure 7.19: Transformation of a BPMN subprocess.

Sequence workflow

A sequence is modeled in BPMN using *connection* objects, in particular, the attributes *ConditionType* and *ConditionExpression*. In the transformation to the Cumbia model, these attributes are stored in the memory of the source activity of the sequence. If the sequence in BPMN does not define data associated with it, the transformation to Cumbia must include the use of a *token* to activate the ports of the target activity. Figure 7.20 illustrates this scenario.

Figure 7.20: Transformation of a BPMN Sequence.

Events

In BPMN there are three types of events: starting, intermediate and ending. Simple starting events are translated directly to the input ports of Cumbia activities. Similarly, simple ending events are translated to the ending ports of Cumbia activities. Intermediate and specialized starting and ending events are also translated into Cumbia activities. This strategy uses the extension mechanism provided by Cumbia, which uses the activity workspace to simulate the event. In this specialization, some of the event's attributes are mapped to memory elements of the activity. Figure 7.21 illustrates this scenario.

Figure 7.21: Transformation of a BPMN Event.

AND Gateway

All BPMN *And* gateways are translated into one output port in Cumbia. Each of the gates associated with the BPMN gateway is translated as a data flow, which starts in the Cumbia port. Figure 7.22 illustrates this scenario.

OR Gateway

To translate an *OR* gateway into the Cumbia model, one port is created for each gate. The condition expression of each gateway is stored in the memory of the preceding (*ANDSplit*) or receiving (*ANDJoin*) activity. During the execution of the Cumbia process, the *Workspace* evaluates the conditions

Figure 7.22: Transformation of a BPMN *And* gateway.

expressions and sends data only to the output ports evaluated to *True*, as seen in figure 7.23.

Figure 7.23: Transformation of a BPMN *OR* gateway.

DataXOR Gateway

The translation strategy for a *DataXOR* gateway is similar to that followed for *OR* gateways. The difference resides in the behavior of the *Workspace*. During the execution of the Cumbia process, the *Workspace* is responsible for the activation of only one port. Figure 7.24 illustrates this scenario.

Figure 7.24: Transformation of a BPMN *DataXOR* gateway.

EventXOR Gateway

The translation for *EventXOR* gateways is based on two strategies: events and gateways. Each gate of the BPMN gateway is represented with a port in the Cumbia activity. In addition, each event associated with the gates is translated into a new Cumbia activity. The *Workspace* of the activity is classified according to event type, as shown in figure 7.25.

Figure 7.25: Transformation of a BPMN *EventXOR* gateway.

Complex Gateway

The translation of BPMN complex gateways follows the same idea. The gateway is translated into a Cumbia activity, with one port for each gate in the complex gateway. According to the evaluation of the conditions, the

Workspace of the new activity decides which ports to activate. Figure 7.26 illustrates this scenario.

Figure 7.26: Transformation of a BPMN *Complex* gateway.

Data Objects

BPMN *DataObjects* are translated directly into Cumbia *Dataflows*. The information associated with *DataObjects* is stored in the memory of the ports defining the *Dataflows* (7.27).

Figure 7.27: Transformation of a BPMN *DataObject*.

7.6.2 Executing the Cumbia Process

Figure 7.28 gives an example, taken from [Jim07], of the transformation of a BPMN process into an executable Cumbia process model. The example

presents a fragment of a software development process model obtained after the weaving of different viewpoints.

Figure 7.28: Fragment of a BPMN software development process model. Source [Jim07].

Once the weaving of the viewpoints with the target process is finished, the woven process model is transformed into a Cumbia process model. Figure 7.29 shows the process model obtained after the transformation.

Figure 7.29: Cumbia process model obtained from the transformation of a BPMN process model. Source [Jim07].

The resulting Cumbia process model is ready to be executed in the Cumbia workflow engine, which offers an administration portal to follow and control

the execution of a process. Figure 7.30 presents a screenshot of the administration portal corresponding to the execution of the process model.

Figure 7.30: Visualization of the execution of a Cumbia process using the administration portal offered by the tool. Source [Jim07].

As part of the visualization, the process can be filtered to follow only the activities associated with a particular viewpoint, called the *Traceability* of the viewpoint. This technique was developed by J. Jimenez [Jim07].

Figure 7.31: Traceability of a Viewpoint. Source [Jim07].

7.7 Summary

This chapter detailed the process and technologies used to support the definition and execution of viewpoints.

The process is composed of five phases: target process definition, viewpoint definition, viewpoint conciliation, viewpoint weaving and process execution.

Target process definition is done using a BPMN metamodel that conforms to the MetaGME metamodel. The GME tool is used to create the target process model.

Viewpoint definition is performed using the AspectViewpoint language. The OpenArchitecturaware framework was used to define the language grammar and to transform AspectViewpoint programs into viewpoint models conforming to the Ecore metamodel.

Viewpoint conciliation is done using two metamodels: PWE and Resolution. The PWE metamodel transforms viewpoints into platform-independent weaving strategies. The resolution metamodel supports the resolution of interferences between the viewpoints and guides the order in which the weaving strategies must be performed.

Viewpoint weaving is accomplished by transforming the PWE model obtained in the previous step into a platform-specific transformation language called the Embedded Constraint Language. These strategies are used by the C-SAW engine to produce a BPMN target process woven with the adaptation strategies.

Finally, the execution of the process is done by transforming the BPMN target process model into a platform-specific process model. We used the Cumbia process model to support the execution of the process.

Chapter 8

Validation and Results

8.1 Introduction

This chapter presents the testing scenarios conducted to validate our work. The first scenario is a software development company with a technical solution process area at level one of the CMMI model for process improvement. The test cases in this scenario demonstrate how the generic practices (concerns) associated with each process area of CMMI can be defined easily using our viewpoint language. Furthermore, we demonstrate that these concerns can be weaved with the process model and then executed in a workflow engine. In this scenario, the responsible of the software development process of the Qualdev [Uni07b] group, acted as stakeholders and process designers. The Qualdev group is conformed by students and professors of the Software Construction Research Group (CSw) [Uni07c] at Los Andes University [Uni07a] in Colombia.

The second testing scenario was conducted in the *Télé-Université (TÉLUQ)* [Tel07], the distance university of the *Université du Québec à Montréal (UQAM)* [UQA07]. We choosed this scenario to demonstrate that our solution can be easily applied to different domains other than software engineering. The TELUQ is a well-known distance university, in which all its courses are designed using e-learning process models. In our testing, a selected group of professors and tutors of the TELUQ acted as stakeholders and process designers.

The rest of this chapter is organized in the following way: section two details the software development test scenario. Section three presents the e-learning testing scenario. Finally, section four presents an analysis of the results.

8.2 The Software Development Process Scenario

The software process used as a testing scenario considers the creation of the detailed level design (CreateDLD), the codification of the product (Codification), the execution of the unit tests (UnitTesting), the registering of any defects (RegisterDefects), and the registering of the actual configuration of the system by a base line application (CreateBL). Figure 8.1 shows the BPMN process model definitions that correspond to these activities.

Figure 8.1: The BPMN process definition of the testing scenario.

In the testing scenario, the software process has been assessed at capability level one, and the objective is to modify the process to achieve level two.

In the CMMI for development, when a process is classified at a particular capability level, it means that the process fulfills a set of generic practices. “A generic practice is a description of an activity that is considered important in achieving a goal. A generic practice is a generic model component” [CKS06]. To improve a process, it is necessary to modify it to address the specific generic practices for the next level. In our example, the generic practices defined for the technical solution process area to reach capability level two are:

- GP2.1 Establish an organizational policy
- GP2.2 Plan the process
- GP2.3 Provide resources

- GP2.4 Assign responsibility
- GP2.5 Train people
- GP2.6 Manage configurations
- GP2.7 Identify and involve relevant stakeholders
- GP2.8 Monitor and control the process
- GP2.9 Evaluate adherence objectively
- GP2.10 Review status with higher level management

In this scenario, each generic practice is associated with a concern that needs to be addressed by the stakeholders of the process and the process designers. We present one mechanism to support the improvement of the process; however, this mechanism is not sufficient to move to capability level two.

The following sections present the concerns required to access capability level two. These concerns and their explanations were extracted from the *CMMI guidelines for process integration and product improvement* by Marie Christis [CKS06]. Some concerns defined by the model are excluded from the testing scenario because they can be achieved without the use of viewpoints: establish an organizational policy, provides resources, assign responsibility, identify and involve relevant stakeholders, and review status with higher level management.

8.2.1 Viewpoint Definitions

The purpose of the scenario is to test the use of viewpoints as a mechanisms to represent crosscutting concerns in a process model. We also want to test our models for detecting inconsistencies between the concerns, and finally, we want to test if the resulting woven process can be executed in a workflow engine.

The following sections detail the viewpoints used to address the generic practices presented above. Each section presents the activities proposed by the model, a list of the typical work products created, the viewpoint associated with the practice expressed with the AspectViewpoint language, and the BPMN process model generated after the weaving of the viewpoint.

The PlanningProcess Viewpoint

Concern: The PlanningProcess viewpoint determines what is needed to perform the process and achieve the established objectives, to prepare a plan for performing the process, to prepare a process description, and to reach an agreement on the plan with stakeholders [CKS06].

Activities required:

- wd1.1 Establish estimates
- wd1.2 Develop project plan
- wd1.3 Obtain commitment

Typical work products obtained:

- wp1.1 Work breakdown structure (WBS)
- wp1.2 Project effort estimate
- wp1.3 Project cost estimate
- wp1.4 Project schedule
- wp1.5 Inventory of skill needs

Strategy: The implementation strategy of this viewpoint is

1. Locate the *CreateDLD* activity.
2. Create the *Estimate* activity and a sequence pattern from it to the join point activity.
3. Create the *DevelopPP* activity and a sequence pattern from it to the join point activity.
4. Create the *ObtainCommitment* activity and a sequence pattern from it to the join point activity.

```

1 viewpoint PlanningProcess
2 affiliation Engineering
3 targetprocess TechnicalSolution
4 {
5   //Locate the CreateDLD task
6   pointcut findCreateDLD (Task task) : target (task) &&
7                                           name ("CreateDLD");
8   //Adds the 'addEstimate' activity and weaves
9   //a sequence before the CreateDLD task
10  strategy addEstimate (Task task) : before findCreateDLD (task)
11  {
12    Task estimate;
13    estimate=task.parent().newFlowObject("Estimate","Task","Normal");
14    estimate.Sequence ([task,""]);
15  }
16  //Adds the 'DevelopPP' activity and weaves
17  //a sequence before the CreateDLD task
18  strategy addDevelop(Task task) : before findCreateDLD (task)
19                                  follows (addEstimate)
20  {
21    Task develop;
22    develop=task.parent().newFlowObject("DevelopPP","Task","Normal");
23    develop.Sequence ([task,""]);
24  }
25  //Adds the 'ObtainCommitment' activity and weaves
26  //a sequence before the CreateDLD task
27  strategy addCommit(Task task) : before findCreateDLD (task)
28                                  follows (addDevelop)
29  {
30    Task commit;
31    commit=task.parent().newFlowObject("ObtainCommitment","Task","Normal");
32    commit.Sequence ([task,""]);
33  }
34 }

```

Listing 8.1: The PlanningProcess Viewpoint implementation

Listing 8.1 presents the AspectViewpoint code used to implement the test case. Figure 8.2 illustrates the result obtained after the weaving of the *PlanningProcess* viewpoint with the target process.

Figure 8.2: The target process after the weaving of the PlanningProcess viewpoint.

The TrainingPeople Viewpoint

Concern: The TrainingPeople viewpoint performs or supports the technical solution process as needed. Appropriate training is provided to the people who will perform the work. Training methods may include self-study, self-directed training, self-paced programmed instruction, on-the-job training, mentoring, and formal and classroom training [CKS06].

Activities required:

- wd2.1 Establish training needs
- wd2.2 Create organizational training plan
- wd2.3 Deliver training
- wd2.4 Assess training

Typical work products obtained:

- wp2.1 Training needs
- wp2.2 Assessment analysis
- wp2.3 Training materials and supporting artifacts
- wp2.4 Training course
- wp2.5 Training records
- wp2.6 Training surveys
- wp2.7 Training examinations

Strategy: The implementation strategy of this viewpoint is

1. Locate the *Codification* activity.
2. Create the *EstablishTN* activity to establish if there are training needs before starting the *Codification* activity. Create an *ExclusiveChoice* to model the possible answers. If there are no training needs, the process continues with the *Codification* activity.
3. Create the *EstablishTC* activity to determine the training material to use in the project.
4. Create an instance of the learning design with the *DeliverTraining* activity.
5. When training is finished, use the *AssessTraining* activity to determine whether or not the required competences were acquired by the actors of the process.

Listing 8.2 presents the AspectViewpoint code used to implement the test case. Figure 8.3 illustrates the result obtained after the weaving of the *Train-People* viewpoint with the target process.

The ManageConfiguration Viewpoint

Concern: The ManageConfiguration viewpoint places designated work products under appropriate levels of control. The base lines are formally reviewed

```

1 viewpoint TrainingPeople
2 affiliation Engineering
3 targetprocess TechnicalSolution
4 {
5   //Locate the Codfication task
6   pointcut findCodification(Task task) : target (task) &&
7                                           name ("Codification");
8
9   //Adds the 'EstablishTN' activity and weaves
10  //a sequence before the CreateDLD task
11  strategy addEstimate (Task task) : before findCodification (task)
12  {
13    Task needs;
14    needs = task.parent().newFlowObject("EstablishTN","Task","Normal");
15    needs.Sequence ([task,""]);
16  }
17  //Adds the 'CreateOTP' activity and weaves
18  //a sequence before the CreateDLD task
19  strategy addChoiceSplit (Task task) : before findCodification (task)
20  {
21    Task plan, deliver, assess ;
22
23    plan=task.parent().newFlowObject("CreateOTP","Task","Normal");
24    deliver=task.parent().newFlowObject("DeliverTR","Task","Normal");
25    assess=task.parent().newFlowObject("AssessTR","Task","Normal");
26    plan.Sequence ([deliver,""]);
27    deliver.Sequence ([assess,""]);
28    task.ExclusiveChoice ([plan,"training=true"]);
29    task.Merge ([assess,""]);
30  }
31 }

```

Listing 8.2: The TrainingPeople Viewpoint implementation

Figure 8.3: The target process after the weaving of the TrainingPeople viewpoint.

and agreed upon, and they serve as the basis for further development of the designated work products. A change request addresses not only new or changed requirements, but also failures and defects in the work products [CKS06].

Activities required:

- wd3.1 Create change request
- wd3.2 Analyze the impact of the changes
- wd3.4 Request creation/update authorization
- wd3.5 Check configuration items into the base line

Typical work products obtained:

- wp3.1 Baseline
- wp3.2 Change request
- wp3.3 Revision history
- wp3.4 Baseline version
- wp3.5 Audit result

Strategy: The implementation strategy of this viewpoint is summarized in the following list. The first adaptation takes place after the *RegisterDefects* activity. The re-factoring of the code must be authorized using a change request.

- Locate the *RegisterDefects* activity.
- Create the *CreateCR* activity to start a base line change request authorization.
- Analyze the change request in the activity *AnalyzeImpactCR* to determine the implication of the changes required to correct the defects found. This activity is followed by the *Codification* activity.

In the second case, the creation of a base line, which is performed in the *CreateBL* activity, requires a previous authorization.

- Locate the *CreateBL* activity.
- Create the activity *ReqAuthorization* to start the base line creation authorization.
- Create an exclusive choice workflow pattern to model the two possible answers. If authorization is granted, the *CreateBL* action is executed. If authorization is not granted, the process ends its execution without checking the changes.

Listing 8.3 shows the AspectViewpoint code used to implement the test case. Figure 8.4 illustrates the result obtained after the weaving of the *ManageConfiguration* viewpoint with the target process.

Figure 8.4: The target process after the weaving of the ManageConfiguration viewpoint.

```

1 viewpoint ManageConfiguration
2 affiliation Engineering
3 targetprocess TechnicalSolutionConfig
4 {
5     //Locate the RegisterDefects task
6     pointcut findDefects(Task task) : target (task) &&
7                                     name ("RegisterDefects");
8     //Locate the CreateBL task
9     pointcut findBaseLine(Task task) : target (task) &&
10                                     name ("CreateBL");
11     //Adds the 'CreateCR' and 'AnalyseImpactCR' activities
12     //and weaves a sequence before the CreateDLD task
13     strategy addChangeRequest (Task task):after findDefects(task)
14     {
15         Task change, analyze ;
16         change=task.parent().newFlowObject ("CreateCR","Task","Normal");
17         analyse=task.parent().newFlowObject ("AnalyseImpactCR","Task","Normal");
18         task.Sequence ([change,""]);
19         task.Sequence ([analyze,""]);
20     }
21     //Adds the 'ReqAuthorization' activity
22     //and weaves an ExclusiveChoice before the CreateBL task
23     strategy addAuthorization (Task task) : before findBaseLine (task)
24     {
25         Task aut;
26         Event end;
27         aut=task.parent().newFlowObject("ReqAuthorization","Task","Normal");
28         end=task.parent().newFlowObject ("End","Event","EndTerminateEvent");
29         aut.Sequence ([task,""]);
30         task.ExclusiveChoice ([end,"authorization=denied"]);
31     }
32 }

```

Listing 8.3: The ManageConfiguration Viewpoint implementation

The MonitorAndControl Viewpoint

Concern: This viewpoint monitors and controls the technical solution process against the plan for performing the process and takes appropriate corrective action. Its purpose is to perform direct day-to-day monitoring and controlling of the process [CKS06]. This viewpoint includes several sub-practices:

Activities required:

- wd4.1 Monitor progress against schedule
- wd4.2 Monitor project's cost and expended effort
- wd4.3 Monitor data management

Typical work products obtained:

- wp4.1 Project performance
- wp4.2 Record data management

Strategy: The implementation strategy of this viewpoint is

- Locate the *CreateDLD* activity.
- Create the *MonitorSchedule1* activity to control the progress of the project against the original schedule of the project. Using a parallel split workflow pattern, create the *MonitorCostAndEffort1* activity to measure the current cost and effort against the original plan. Create another gate of the parallel split pattern to add the activity *MonitorDataMgm1*, which controls changes, such as requirements modifications, to the data. These three activities are synchronized before the execution of the *RegisterMCDData1* activity using a synchronization workflow pattern. The monitoring and control activities are executed in parallel with the *Codification* activity.
- Locate the *Codification* activity.

- Create the same set of monitor and control activities as in the previous step: *MonitorSchedule2*, *MonitorCostAndEffort1* and *MonitorDataMgm1*, using a parallel split pattern.

Listing 8.4 presents the AspectViewpoint code used to implement the test case. Figure 8.5 illustrates the result obtained after the weaving of the *ManageConfiguration* viewpoint with the target process.

Figure 8.5: The target process after the weaving of the MonitorAndControl viewpoint.

The EvaluateAdherence Viewpoint

Concern: This viewpoint objectively evaluates adherence of the technical solution process to its process description, standards and procedures, and addresses noncompliance [CKS06].

Activities required:

- wd5.1 Objectively evaluate process conformance
- wd5.2 Solve noncompliance problems
- wd5.3 Establish records of the unconformities

```

1 viewpoint PlanningProcess
2 affiliation Engineering
3 targetprocess TechnicalSolutionMonitor
4 {
5   //Locate the CreateDLD task
6   pointcut findCreateDLD (Task task) : target (task) &&
7                                           name ("CreateDLD");
8   //Locate the Codification task
9   pointcut findCodification (Task task) : target (task) &&
10                                           name ("Codification");
11
12   //Creates a ParallelSplit for the
13   //control and monitoring tasks
14   strategy addControlDLD (Task task) : after findCreateDLD (task)
15   {
16     Task sch, effort, data, reg;
17     sch=task.parent().newFlowObject("MonitorSchedule1","Task","Normal");
18     effort=task.parent().newFlowObject("MonitorCE1","Task","Normal");
19     data=task.parent().newFlowObject("MonitorDataMgm1","Task","Normal");
20     reg=task.parent().newFlowObject("RegisterData1","Task","Normal");
21     task.ParallelSplit([sch,""],[effort,""],[data,""]);
22     reg.Synchronization([sch,""],[effort,""],[data,""]);
23   }
24
25   //Creates a ParallelSplit for the
26   //control and monitoring tasks
27   strategy addControlCod (Task task) : after findCodification (task)
28   {
29     Task sch, effort, data, reg;
30
31     sch=task.parent().newFlowObject("MonitorSchedule2","Task","Normal");
32     effort=task.parent().newFlowObject("MonitorCE2","Task","Normal");
33     data=task.parent().newFlowObject("MonitorDataMgm2","Task","Normal");
34     reg=task.parent().newFlowObject("RegisterData2","Task","Normal");
35     task.ParallelSplit([sch,""],[effort,""],[data,""]);
36     reg.Synchronization([sch,""],[effort,""],[data,""]);
37   }
38 }

```

Listing 8.4: The MonitorAndControl Viewpoint implementation

Typical work products obtained:

- wp5.1 Evaluation report
- wp5.2 Noncompliance report
- wp5.3 QA report

Strategy: The implementation strategy of this viewpoint is

- Locate the *Codification* activity.
- Create the *EvaluateConformance* activity to establish if the process was followed during *Codification*. If the result is satisfactory, the execution of the process continues with the *UnitTesting* activity. If there are non-conformities, they are reported by the activity *ReportNonCompliance*. The *SolveNonCompliance* activity is used to resolve the noncompliances found, and finally, these problems are registered as documentation artifacts during the *DocumentNonCompliance* activity.

Listing 8.4 presents the AspectViewpoint code used to implement the test case. Figure 8.6 illustrates the result obtained after the weaving of the *EvaluateAdherence* viewpoint with the target process.

Figure 8.6: The target process after the weaving of the EvaluateAdherence viewpoint.

```

1 viewpoint Adherence
2 affiliation Engineering
3 targetprocess TechnicalSolutionAdh
4 {
5   //Locate the Codification task
6   pointcut findCodification (Task task) : target (task) &&
7                                           name ("Codification");
8   //Locate the Codification task
9   pointcut findEvaluate (Task task) : target (task) &&
10                                      name ("EvaluateConformance");
11
12   //Creates a Sequence for the
13   //EvaluateConformance task
14   strategy addEvaluate (Task task) : after findCodification (task)
15   {
16     Task evaluate;
17     evaluate=task.parent().newFlowObject ("EvConformance", "Task", "Normal");
18     task.Sequence ([evaluate, ""]);
19   }
20   //Creates an ExclusiveChoice pattern
21   strategy addReport(Task task) : after findEvaluate (task)
22                                   follows (addEvaluate)
23   {
24     Task rep, solve, doc;
25     rep = task.parent().newFlowObject ("ReportNC", "Task", "Normal");
26     solve = task.parent().newFlowObject ("SolveNC", "Task", "Normal");
27     doc = task.parent().newFlowObject ("DocumentNC", "Task", "Normal");
28     rep.Sequence ([solve, ""]);
29     solve.Sequence ([doc, ""]);
30     task.ExclusiveChoice ([rep, ""]);
31   }
32 }

```

Listing 8.5: The EvaluateAdherence Viewpoint implementation

8.2.2 Crosscutting Concerns

In section 4.1, we introduced the crosscutting pattern proposed in [vdBC05], as a mechanism to determine scattering, tangling, and crosscutting relationships between different concerns. We retake this pattern in this section to illustrate these relationships in the viewpoints defined in the previous sections.

To construct the matrices required by the pattern, we define as targets each of the activities of the technical solution process area considered in our test case. The sources are the five viewpoints presented above.

The following list presents the elements used as targets of the pattern:

- CreateDLD (CD)
- Codification (Cod)
- UnitTesting (UT)
- RegisterDefects (RD)
- CreateBaseLine (CB)

The following list presents the elements used as sources of the pattern:

- PlanningProcess (PP)
- TrainingPeople (TP)
- ManageConfiguration (MC)
- MonitorAndControl (Mon)
- EvaluateAdherence (EA)

The first step defined by the crosscutting pattern is the definition of the dependency matrix (source x target). This matrix is used to establish the dependency relation between source elements and target elements [vdBC05]. Table 8.1 presents the dependency matrix for the testing scenario.

The second step is the definition of the scattering matrix (source x target) to represent dependency relations from source to target elements. Table 8.2 presents the scattering matrix for the testing scenario.

	CD	Cod	UT	RD	CB	
PP	1	0	0	0	0	NS
TP	0	1	0	0	0	NS
MC	0	0	0	1	1	S
Mon	1	1	1	0	0	S
EA	1	1	1	0	0	S
	T	T	T	NT	NT	

Table 8.1: Dependency matrix for the testing scenario

	CD	Cod	UT	RD	CB
PP	0	0	0	0	0
TP	0	0	0	0	0
MC	0	0	0	1	1
Mon	1	1	1	0	0
EA	1	1	1	0	0

Table 8.2: Scattering matrix for the testing scenario

The third step is the definition of the tangling matrix (target x source) to represent dependency relations from target to source elements. Table 8.3 presents the tangling matrix for the testing scenario.

The fourth step is the derivation of the crosscutting product matrix (source x source) obtained through the matrix multiplication of the scattering matrix and the tangling matrix. The crosscutting product matrix represents the frequency of crosscutting relations. Table 8.4 presents the crosscutting product matrix for the testing scenario.

The final step is the derivation of the crosscutting matrix (source x source)

	PP	TP	MC	Mon	EA
CD	1	0	0	1	1
Cod	0	1	0	1	1
UT	0	0	0	1	1
RD	0	0	0	0	0
CB	0	0	0	0	0

Table 8.3: Tangling matrix for the testing scenario

	PP	TP	MC	Mon	EA
PP	0	0	0	0	0
TP	0	0	0	0	0
MC	0	0	0	0	0
Mon	1	1	0	3	3
EA	1	1	0	3	3

Table 8.4: Crosscutting product matrix for the testing scenario

	PP	TP	MC	Mon	EA
PP	0	0	0	0	0
TP	0	0	0	0	0
MC	0	0	0	0	0
Mon	1	1	0	0	1
EA	1	1	0	1	0

Table 8.5: Crosscutting matrix for the testing scenario

from the crosscutting product matrix. In this case, the matrix is obtained in the following way: If the cell $[i,j]$ ($i \neq j$) of the crosscutting product matrix is greater than zero, then a value of 1 is assigned to the cell $[i,j]$ of the crosscutting matrix. Otherwise, a zero value is assigned. Table 8.5 presents the crosscutting product matrix for the testing scenario.

From the values found in the crosscutting product matrix, we can establish that there is high frequency of crosscutting relations between the monitor and control concern (Mon) and the evaluate adherence (EA) concern. From the crosscutting matrix we can conclude the following:

- The monitor and control concern (Mon) crosscuts the planning process (PP) concern
- The monitor and control concern (Mon) crosscuts the training people (TP) concern
- The monitor and control concern (Mon) crosscuts the evaluate adherence (EA) concern
- The evaluate adherence concern (EA) crosscuts the monitor and control (Mon) concern

8.3 The Learning Process Scenario

Modeling and supporting rich e-learning situations are a complex task. Not only different entities must be orchestrated, such as different actors (student and staff), resources, services, knowledge, and activities, but also the process itself should be flexible for personalization and evolution.

Educational modeling languages have been criticized for being either too simplistic or too complex for educational designers to use. Behind these critics, there is a search for a balance between expressive power and simplicity. We advocate that identifying and modeling in a separate way the different viewpoints of a learning process helps to keep the modeling of the main learning process simple, while richness of expression is obtained through the integration of those viewpoints at execution time. Moreover, learn-flow oriented educational modeling languages, such as IMS-LD could be used both to describe the main learning process and the viewpoints.

In this testing scenario we define a simple learning process composed of three activities. The first activity *Experiment* is performed by the student. Once this activity finishes, the tutor performs the *CorrectExperiment* activity, to analyze the data obtained from the experiment and to give some feedback to the student. Then, a third activity, called *Analysis*, is performed by the student to produce a critical analysis and conclusions. The figure presents the definition of this process.

Figure 8.7: The e-learning process scenario.

8.3.1 Viewpoint Definitions

We base the test cases of this scenario on the e-learning viewpoints presented in [Cor06] proposed as a mechanism to the adaptation of e-learning processes.

In the first case, a viewpoint is used to change activities to particular students so as to personalize their learning experience. The second case adds activities to the staff member in charge of supporting the student learning, to help him identify and deal with potential extra assistance situations. Case three shows how viewpoints can be used to relate different learning situations of a same learner, while the last situation is an example of the potential of viewpoints to make the link between the knowledge management and the learning process.

The LearningPersonalization Viewpoint

Concern: In this scenario, we propose the use of viewpoints as a mechanism of learning personalization in which a given course could be adapted by the viewpoint. In this case, we classify students into two groups: A and B. The selection is made according to the skills of each student.

In the first case, the students of the group A must accomplish a review activity before starting the experiment. As for the group B, the students start the experiment, after that, the reviewer sends them the corrections and the students produce the analysis. In this case, the process is extended with a complementing activity that requires the production of a more extended version of the analysis. This situation could arise, for instance, for masters degree students, allowed to take undergraduate courses with an additional charge.

Activities required:

- wd1.1 Determine competencies
- wd1.2 Review Activity
- wd1.3 Practice Activity
- wd1.4 Extended Analysis

Strategy: The implementation strategy of this viewpoint is presented in the following list

- Locate the activity Experiment

- Create the activity EvaluateCompetence1 to determine if the student belongs to group A or B. Create an ExclusiveChoice pattern to differentiate the two options.
- Create the Review activity for the group A
- Locate the Analysis activity and create the activity EvaluateCompetence2 to determine if the student belongs to group A or B. Create the activity ExtendedAnalysis and an ExclusiveChoice pattern to differentiate the two options.

Listing 8.6 presents the AspectViewpoint code used to implement the test case. Figure 8.8 illustrates the result obtained after the weaving of the *LearningPersonalization* viewpoint with the target process.

Figure 8.8: The e-learning process scenario after the weaving of the LearningPersonalization viewpoint.

The AssistanceDetection Viewpoint

Concern: In this scenario, we use the viewpoint mechanism to illustrate a control process case. In this example, a viewpoint is used to detect possible warning situations in which the behavior of a student shows early symptoms of extracurricular problems.

When the student starts the experiment, a new activity (*RegisterStart*) registers the starting date and time of the activity as well as the complexity level of the experiment. When the Experiment activity is finished by the student, the activity *RegisterEnd* registers the ending date and time of the activity. The viewpoint also recovers data from the evaluation (*CorrectExperiment*)

```

1 viewpoint LearningPersonalization
2 affiliation Learning
3 targetprocess ExperimentalCourse
4 {
5   //Locate the Experiment task
6   pointcut findExperiment (Task task) : target (task) &&
7                                           name ("Experiment");
8   //Locate the Analysis task
9   pointcut findAnalisis (Task task) : target (task) &&
10                                          name ("Analysis");
11   //Creates the Sequence Pattern
12   strategy addEvaluate (Task task):before findExperiment(task) {
13     Task evaluate;
14     evaluate=task.parent().newFlowObject("EvaluateCompetence1",
15                                          "Task", "Normal");
16     task.Sequence ([evaluate, ""]);
17   }
18   //Creates the Review task
19   strategy addReview(Task task):before findExperiment (task) {
20     Task rep;
21     rep = task.parent().newFlowObject("Review", "Task", "Normal");
22     task.ExclusiveChoice ([rep, "goup=A"]);
23   }
24   //Creates the ExtendedAnalysis task
25   strategy addExtended (Task task):after findAnalisis(task) {
26     Task rep;
27     rep = task.parent().newFlowObject("ExtendedAnalysis",
28                                          "Task", "Normal");
29     task.ExclusiveChoice ([rep, "goup=b"]);
30   }
31 }

```

Listing 8.6: The LearningPersonalization Viewpoint implementation

and after analyzing both the evaluation of the student's work and the time spent by the student doing the experiment (*AnalyzeWork*), the teacher can establish if the student requires a more personalized assistance. In that case, a new support activity (*AssistedAnalysis*) is added to the supporting process to better accompany the learner through the following learning activity.

Activities required:

- wd2.1 RegisterStart
- wd2.2 RegisterEnd
- wd2.3 AnalyseWork
- wd2.4 Assistance

Strategy: The implementation strategy of this viewpoint is

- Locate the Experiment activity
- Add the RegisterStart activity following a ParallelSplit pattern
- Locate the Experiment activity
- Add the RegisterEnd activity following a ParallelSplit pattern
- Locate the CorrectExperiment activity
- Add the AnalyzeWork activity following a Sequence pattern
- Locate AnalyzeWork activity
- If the student requires a personalized assistance then the AssistedAnalysis activity is created following an ExclusiveChoice pattern

Listing 8.7 presents the AspectViewpoint code used to implement the test case. Figure 8.9 illustrates the result obtained after the weaving of the *LearningPersonalization* viewpoint with the target process.

```

1 viewpoint AssistanceDetection
2 affiliation Learning
3 targetprocess ExperimentalCourse
4 {
5   //Locate the Experiment task
6   pointcut findExperiment (Task task) : target (task) &&
7                                           name ("Experiment");
8   //Locate the CorrectExperiment task
9   pointcut findCorrect(Task task) : target (task) &&
10                                      name ("CorrectExperiment");
11  //Creates a ParallelSplit pattern
12  strategy addStart (Task task):before findExperiment(task){
13    Task start;
14    start = task.parent().newFlowObject("RegisterStart",
15                                       "Task", "Normal");
16    task.ParallelSplit ([start,""]);
17  }
18  //Creates a ParallelSplit pattern
19  strategy addEnd (Task task):after findExperiment(task) {
20    Task end;
21    end=task.parent().newFlowObject("RegisterEnd",
22                                    "Task", "Normal");
23    task.ParallelSplit ([end,""]);
24  }
25  //Creates the ExclusiveChoice pattern
26  strategy addAssisted(Task task):after findCorrect(task) {
27    Task rep;
28    rep=task.parent().newFlowObject("AssistedAnalysis",
29                                    "Task", "Normal");
30    task.ExclusiveChoice ([rep,"goup=b"]);
31  }
32 }

```

Listing 8.7: The AssistanceDetection Viewpoint implementation

Figure 8.9: The e-learning process scenario after the weaving of the AssistanceDetection viewpoint.

The LernerCentered Viewpoint

Concern: Students studying various courses at the same time can develop competencies in one course that are worked on in other courses as well; This is very often the case for core competencies. The following example shows how viewpoints can help to make the link between all those learning situations so that the learner stays in the center of his/her learning and each learning process takes into account at every moment the learner’s competencies.

Figure 8.10 illustrates this scenario. In the learning process A, after the correction of the experiment, the tutor determines that the student has acquired a competency. This situation is modeled sending a message notification to a process in charge of the competencies of the students. This process receives the notification and updates the student’s profile. Then, the system process sends a notification to the student’s instance of the learning processes B to notify about the new competency. This notification is used by the learning process B for personalization purposes, for example, to change the level of difficulty in an evaluation that requires this competency.

As presented in figure 8.10, this scenario requires the modeling of different parallel processes and the use of messages between them. In the current implementation of our BPMN metamodel, we are constrained to the definition of one process per business diagram. Additionally, the AspectViewpoint language does not support the message connections required between multiple target processes. For these reasons, this scenario can not be implemented with our current proposal.

Figure 8.10: Definition of the LernerCentered viewpoint using the BPMN notation.

The KnowledgeOriented Viewpoint

Concern: The fourth viewpoint scenario presented here is related to the assignment of the adequate human or digital resources to an activity before its initialization.

Figure 8.11 shows an example in which the *Experiment* activity is related to two activities in a viewpoint. When the activity is ready to start its execution, the information related to the requirements and the goals defined in the activity are sent to the *SearchResource* activity. This activity searches for the appropriate resources that best fits the student's skills and the goals of the learning activity. Once the resources are located, activity *UpdateResource* sends this information to activity *Experiment* and enables the initialization of this activity.

Activities required:

- wd4.1 SearchResource
- wd4.2 UpdateResource

Strategy: The implementation strategy of this viewpoint is

```

1 viewpoint KnowledgeOriented
2 affiliation Learning
3 targetprocess ExperimentalCourse
4 {
5   //Locate the Experiment task
6   pointcut findExperiment (Task task) : target (task) &&
7                                           name ("Experiment");
8   //Creates a Sequence pattern
9   strategy addUpdate (Task task) : before findExperiment (task)
10  {
11    Task search , update;
12    search = task.parent ().newFlowObject ("SearchResource",
13                                           "Task", "Normal");
14    update = task.parent ().newFlowObject ("UpdateResource",
15                                           "Task", "Normal");
16    task.Sequence ([update, ""]);
17    task.Sequence ([search, ""]);
18  }
19 }

```

Listing 8.8: The KnowledgeOriented Viewpoint implementation

- Locate the activity Experiment
- Create the activities SearchResource and UpdateResource
- Connect the previously created activities with the experiment activity following a sequence pattern

Listing 8.8 presents the AspectViewpoint code used to implement the test case. Figure 8.11 illustrates the result obtained after the weaving of the *LearningPersonalization* viewpoint with the target process.

Figure 8.11: The e-learning process scenario after the weaving of the KnowledgeOriented viewpoint.

	EX	CE	AN	
LP	1	0	1	S
AD	1	1	1	S
KO	1	0	0	NS
	T	NT	T	

Table 8.6: Dependency matrix for the learning testing scenario

	EX	CE	AN
LP	1	0	1
AD	1	1	1
KO	0	0	0

Table 8.7: Scattering matrix for the learning testing scenario

8.3.2 Crosscutting Concerns

In this section we present the crosscutting pattern applied to the Learning scenario.

The following list presents the elements used as targets of the pattern:

- Experiment (EX)
- CorrectExperiment (CE)
- Analysis (AN)

The following list presents the elements used as sources of the pattern:

- LearningPersonalization (LP)
- AssistanceDetection (AD)
- KnowledgeOriented (KO)

Table 8.6 presents the dependency matrix for the learning testing scenario.

Table 8.7 presents the scattering matrix for the learning testing scenario.

Table 8.8 presents the tangling matrix for the learning testing scenario.

Table 8.9 presents the crosscutting product matrix for the testing scenario.

	LP	AD	KO
EX	1	1	1
CE	0	0	0
AN	1	1	0

Table 8.8: Tangling matrix for the learning testing scenario

	LP	AD	KO
LP	2	2	1
AD	2	2	1
KO	0	0	0

Table 8.9: Crosscutting product matrix for the learning 1testing scenario

Table 8.10 presents the crosscutting product matrix for the testing scenario.

From the crosscutting matrix we can conclude the following:

- The learning personalization (LP) concern crosscuts the assistance detection (AD) concern
- The learning personalization (LP) concern crosscuts the knowledge oriented (KO) concern
- The assistance detection (AD) concern crosscuts the learning personalization (LP) concern
- The assistance detection (AD) concern crosscuts the knowledge oriented (KO) concern

	LP	AD	KO
LP	0	1	1
AD	1	0	1
KO	0	0	0

Table 8.10: Crosscutting matrix for the learning testing scenario

8.4 Analysis of Results

This section presents the results obtained after the development of the testing scenarios. We start by describing the results obtained in each of the five phases proposed in chapter seven to define and execute viewpoints on workflow processes.

8.4.1 Target Process Definition

The Generic Modeling Environment (GME) proved to be a powerful tool in the creation of Domain-Specific models. Particularly, the BPMN meta-model developed with GME, facilitated us the definition of workflow process models in different domains. We succeeded in the definition of several process metamodels, like BPMN, SPEM, IMSL-LD, and Cumbia. In all the cases, GME proved to be stable and easy to use. Our strategy of using a model editor instead of a graphics editor to create BPMN processes was a success. Thanks to this decision, we could manipulate the model, reflexively, introducing adaptations in the process model in a programmatic way.

The main limitation found during this phase was the absence of *swimlanes* in the editor. This limitation made difficult the association of participants of the process with their activities. Another limitation introduced by the absence of *swimlanes* was the lack of modeling mechanisms for the interactions between different processes in the same diagram, as was the case with the *LearningCentered* test case. This problem could be solved, introducing a new graphical symbol to represent *lanes* and *pools*. This new element can be associated to the activities of the lane with a new association symbol, indicating in this way that they belong to the same process. Nevertheless, this solution wont be compliant with the BPMN specification.

8.4.2 Viewpoint Definition

The viewpoint definition was done using the AspectViewpoint language, which proved to be suitable for the definition of the join points, pointcuts and strategies required to solve each of the concerns. In our testing, we successfully used six of the proposed workflow control patterns: sequence,

parallel split, synchronization, exclusive choice, arbitrary cycles, and merge. By means of these patterns, we helped process designers to express in a Domain-Specific language the requirements expressed by stakeholders. Particularly, in the software testing scenario, process designers found useful the incorporation of the workflow patterns as part of the vocabulary of the language and the ability to define, in separate modules, the concerns expressed by the stakeholders. Process designers expressed that these characteristics facilitated to understand the effects of the changes introduced on the target process after the weaving of each viewpoint.

However, process designers suggested an improvement during the definition of sequences involving multiple activities. In our current implementation, the sequence pattern can be defined only between two activities. This limitation can be surmounted if the sequence pattern is applied to a list of activities. The associated semantics is the creation of a sequence of activities starting with the first element of the list and finishing with the last one.

Process designers compared their traditional way of changing a target process, introducing modifications directly into the process definition, with the use of viewpoints. Their conclusion, in both testing scenarios, was that the use of viewpoints reduces the complexity associated to the modification of the target process.

To illustrate this behavior, we present some metrics based on the software case scenario. First, we present in figure 8.12 a relation between the number of *FlowObjects* contained in the target process after the inclusion of a concern versus the number of concerns. The number of *FlowObjects* passed from nine after the inclusion of the first concern to 50 after the inclusion of the fifth concern. This means that each time a process designer needs to introduce a new modification directly into the process model, the complexity of this task augments because the size of the model in terms of the *FlowObjects* contained.

The definition of viewpoints as modular and independent units to define concerns on a target process model, reduces the complexity associated with the modification of the process because the size of the target process remains constant.

We propose two additional indicators to measure the complexity of introducing modifications in a process model. These indicators are: the number of

Figure 8.12: Number of *FlowObjects* contained in the target process after the modifications introduced by each concern.

gateways in the process model and the number of potentially execution paths in the target process.

In the first case, we used the number of gateways as a measure of the complexity associated with the modification of a process. They represent control points in the execution of the process where potential side effects of the modifications may be introduced, as infinite cycles or execution paths never reached. When this changes are done directly into the process definition, it's difficult to manipulate the model to introduce the changes. With the use of viewpoints, the number of gateways in the process model remains constant reducing the complexity and facilitating the labor of process designers.

The graphics on the left of figure 8.13 show the number of gateways in the target process after the introduction of each concern. The original target process starts with two gateways, and after the introduction of the fifth concern, the process has 12 gateways.

The graphics on the right of the figure 8.13 show the increment in the number of execution paths in the target process model after each modification. We consider this measure also as an indicator of the complexity associated to the modification of a target process because process designers must understand how the introduction of a change in the process model could affect the overall process execution. For example, in the software development scenario, before the introduction of the first viewpoint, the process presents two possible execution paths, and with the introduction of the fifth concern, the process has 18 possible execution paths.

Figure 8.13: Number of gateways and execution paths present in the target process model after the introduction of each concern

To make more evident the complexity associated to direct modification on a process model versus the use of viewpoints, consider a variation on our software testing scenario. Initially, we considered only one process area composed of five activities. In a more complex scenario, a process designer need to face the simultaneous evolution of the 22 process areas defined by the CMMI, each of them with more than five activities. Introducing direct modifications in this context can lead rapidly to unmanaged models. However, this would not be the case with the use of viewpoints because the complexity of the changes in this case remains constant and independent of the total number of concerns.

8.4.3 Viewpoint Conciliation

The conciliation metamodel was created with two objectives in mind: first, to create a language-independent model of the adaptations defined in a target process. Second, to make explicit the interference relations between these adaptations.

The first objective was successfully completed. We were able to create a conciliation model representing the weaving strategies of the testing scenarios. This model was used as a pivot to transform the adaptations defined in the viewpoint models, using the AspectViewpoint language, into ECL weaving strategies.

The second objective was also successfully completed. During the construc-

tion of the resolution model, we could make explicit relationships such as scattering, tangling, and crosscutting, which helped users to better understand their process. For instance, one interference was detected during the definition of the *EvaluateAdherence* viewpoint. The solution to this conflict was the use of the *follows* instruction to define the precedence of the strategies.

We need to further investigate how the resolution metamodel can be used to detect more complex interference scenarios. For example, how the crosscutting frequency obtained from the crosscutting product matrix can be used to detect interference situations between crosscutting concerns. The master's thesis of Rob Vanmeert will continue the research on this topic.

8.4.4 Viewpoint Weaving

We used the C-SAW weaving engine to effectively transform the target process by weaving the strategies defined in the PWE model. The weaving process was performed successfully in both cases, and the resulting woven process contained the modifications defined by the stakeholders of the process.

The figure 8.14 presents the resulting process woven with the five viewpoints defined in the software testing scenario.

The figure 8.15 presents the resulting woven process for the learning scenario.

As a final remark, we are aware that a formal verification of the resulting woven process models must be provided to guarantee that the weaving strategies preserve the coherence of the process. However, this objective was out of the scope of this thesis and it's proposed as future work.

8.4.5 Viewpoint Execution

We succeeded in the transformation of the resulting woven process model of the software scenario, into an executable Cumbia model. We proved in this way, that it is possible to support the execution of complex workflow processes in which several crosscutting concerns have been defined. Figure 8.16 presents the Cumbia model executed in the Cumbia workflow engine using

Figure 8.14: The target process after the weaving of the software viewpoints.

Figure 8.15: The target process after the weaving of the learning viewpoints.

the model transformation developed by Jorge Jiménez [Jim07] as part of its Mater's thesis at the Software Construction Group in Los Andes University. In his work, Jiménez developed a transformation mechanism for process models, conform to the BPMN metamodel, into Cumbia models defined with the GME environment .

Figure 8.16: The Cumbia process model obtained from the woven process model of the software scenario.

In the case of the learning scenario, we limited our test to the weaving of the viewpoints with the target process. We did not perform the translation of the woven process into an executable Cumbia process model. This task is proposed as future work.

8.5 Summary

This chapter presented two testing scenarios for the validation of our work: a software process model and a learning process model. In the first case, we defined a minimal target process model representing the implementation phase during the development of a component. As part of the testing scenario, we defined five viewpoints modeling the concerns associated with the improvement of a software development process under the CMMI scenario. Particularly, we modeled some of the generic practices associated with capability level two. In the second case, we model an e-learning process model

and some concerns related with the personalization of courses according with the requirements of each student.

We developed metamodels and tools and tested them as to whether or not they are appropriate to support the proposed testing scenarios. We used the BPMN metamodel to support the definition of the target processes and the AspectViewpoint language to support the definition of viewpoints.

At the end of the process, we successfully transformed the viewpoints into platform-specific weaving strategies. These strategies were used by the C-SAW engine to make modifications to the target process model.

As a result, we obtained a process model enriched without any direct intervention on the original BPMN process model. The transformations were defined in a modular and non-intrusive way using the AspectViewpoint language.

In the case of the software testing scenario, the resulting woven process model was transformed to a Cumbia process model and instantiated by the Cumbia workflow engine.

Part III

*"Keep Ithaka always in your mind.
Arriving there is what you're destined for.
But don't hurry the journey at all.
Better if it lasts for years,
so you're old by the time you reach the island,
wealthy with all you've gained on the way,
not expecting Ithaka to make you rich.
Ithaka gave you the marvelous journey.
Without her you wouldn't have set out.
She has nothing left to give you now.*

*And if you find her poor, Ithaka won't have fooled you.
Wise as you will have become, so full of experience,
you'll have understood by then what these Ithakas mean."*

From *Ithaka* by C.P. Cavafy

Chapter 9

Discussion

The main motivation for the work presented here was to propose a mechanism to assist process designers during the definition of multiple crosscutting concerns in business process models, facilitating their evolution and their execution in a workflow engine.

We followed an aspect-oriented modeling approach, and we designed and implemented a domain-specific aspect language, *AspectViewpoint*, to support the definition of crosscutting concerns as modular and independent entities. We also provided the transformations required to execute the resulting woven process model into a workflow engine.

This chapter presents a summary of this work, its main contributions, our conclusions and potential future research directions.

9.1 Thesis Summary

In the introduction, we presented the state of the art of the three main technologies involved in this work: process modeling, model-driven engineering and aspect-oriented software development. We gave several examples of process definition languages, with an emphasis on the Business Process Modeling Notation. We also presented the main concepts of model-driven engineering and some of its more important implementations: MDA, MIC and Ecore.

Finally, we presented the main elements of aspect-oriented software development and the application of these concepts to the aspect-oriented modeling domain.

In the second part of the thesis, we presented our proposal and the testing scenario we used to validate it. The proposal was explained in five phases: target process definition, viewpoint definition, viewpoint conciliation, viewpoint weaving and process execution. The first three phases were implemented using process, viewpoint and conciliation metamodels. The process metamodel was based on the BPMN specification and developed to define a target process model. The viewpoint metamodel made possible the definition of viewpoint models using the AspectViewpoint language. The conciliation metamodel was used to construct a pivot model to transform viewpoint models into platform-independent weaving strategies.

We used a testing scenario defined in the context of the CMMI for development, in particular, the technical solution process area at capability level two. We developed different viewpoints to express some of the concerns associated with this level. These viewpoints were transformed and weaved into the target process model, which was executed into a Cumbia workflow engine.

9.2 Conclusions

The model-driven engineering strategy has proven to be a supporting tool for process designers, who must deal with constant changes in process definitions. MDE made possible to be technology-independent, giving us the possibility of transformations from notations such as BPMN to Cumbia or other executable models.

Aspect-oriented modeling techniques have proven suitable mechanisms for the separation of concerns at a higher level of abstraction. In the context of process modeling, viewpoints help process designers manage the complexity of tasks by achieving the separation of concerns. However, from the beginning of our validation, we saw how difficult it is for process designers to be aware of possible overlap between viewpoints because each stakeholder focuses on his own particular concern. Because of this, we developed the intermediate process weaving metamodel to help users identify and solve conflicts. Fur-

thermore, during the construction of the intermediate model, we can make relationships, such as scattering, tangling and crosscutting, explicit among viewpoints, which can help users to better understand the whole picture.

The initial prototypes and test cases developed to validate our models have produced satisfactory results. First, we accomplished our objective of defining concerns as viewpoint models. Second, we transformed these viewpoints into platform-specific process models and executed them in the Cumbia workflow engine.

We need to go further to produce a complete executable business process. Thus far, we have focused on the modeling and control of activities, but to execute a process we need to consider other elements involved, such as resources, products and tools. We are working toward this goal by testing workflow engines and trying to understand the other elements in order to propose extensions to the AspectViewpoint language. We are working on the generation of a traceability model that could be interpreted at execution time to give feedback to the stakeholder responsible for each viewpoint [Jim07].

9.3 Main Contributions

The main contributions of this work are:

- An original approach to support the evolution of process models based on aspect-oriented modeling.
- A business process metamodel, based on the BPMN specification and conforming to the MetaGME meta-metamodel, that can be used in different process modeling domains, such as e-learning processes, process intelligence and process simulation.
- A novel domain-specific aspect language called AspectViewpoint, which includes a basic set of workflow control patterns.
- An eclipse plugin editor for the AspectViewpoint language.
- A viewpoint metamodel conforming to the Ecore meta-metamodel to support the definition of crosscutting concerns in software process models.

- A conciliation metamodel conforming to the Ecore meta-metamodel that has two main uses. First, it can be used to create a platform-independent pivot model to express weaving strategies. Second, it makes relationships explicit between the concerns expressed in a process model, such as scattering, tangling and crosscutting.
- A set of transformations to support the conversion of viewpoint models into conciliation models.
- A mechanism of automatic code generation to create the ECL strategies required to weave the viewpoints with the target process from the conciliation model.

9.4 Future Work

We have integrated several research domains, including model-driven engineering, aspect-oriented modeling, and business process modeling and execution. We propose a coordinated use of these technologies to solve some of the problems associated with enriching process models. However, this is only a first step. The following sections present some logical continuations of this work.

9.4.1 Multidimensional Viewpoints

We focused mainly on the control flow aspects of business process modeling. However, the concerns defined by stakeholders frequently involve other aspects of the problem, such as work products and the management of resources, including the different roles, people, applications and participants in the activities. We believe that a multidimensional approach is necessary to model a complete business process scenario in which concerns can be defined in the control, data or resource dimensions.

Borrowing terminology from the Hyperslices [TOHJ99] proposal, we support the definition of concerns in only one dimension: control flow. The AspectViewpoint language has adopted the vocabulary of workflow control patterns, and our metamodels accurately represent this dimension. However, a process can be seen as the composition of slices of different dimensions,

meaning that in the resource management dimension, we could use resource patterns to define crosscutting concerns, and in the data management dimension, we could use data patterns to define crosscutting concerns. In this approach, a single process could be defined as the composition of multidimensional viewpoints.

9.4.2 Reusability

The current implementation of the AspectViewpoint language does not support the reuse of strategies or viewpoints. An improvement could be additional references to external strategies and pointcuts in the grammar of the language.

9.4.3 Interference

We support the identification of dependency between strategies, using *follows* to instruct the weaving engine about precedence. However, a more elaborate analysis must be done to detect interferences added as a consequence of strategies being defined in multiple viewpoints. Rob Vanmeert's current research is focused on the detection of interferences between viewpoints.

9.4.4 Workflow Patterns

There are several future directions that could be followed in the use of workflow control patterns. Currently, we support only a subset of the workflow control patterns, but it would be interesting to implement more complex patterns, such as discriminators, interleaved parallel routing, milestones and multi-instances. Furthermore, a future version of the AspectViewpoint language could implement some of the 23 control flow patterns that have recently been created.

9.4.5 Dynamic Weaving

The AspectViewpoint language supports the definition of static and dynamic join points as well as the static weaving of viewpoint models. The next step in this direction is support for the dynamic weaving of viewpoints. However, the potential for this improvement is limited by the weaving engine used in our current implementation (C-SAW). Future work could be a collaboration with the SoftCom team at the University of Alabama to explore the viability of a dynamic weaving version of the C-SAW engine.

9.4.6 Cumbiambas

We consider the execution of a process model resulting from the weaving of viewpoints with a target process model. This execution takes place in a workflow engine, which does not differentiate between the origins of the activities in a process during its execution. During the execution of the process, it would be desirable to have virtual applications, which we call Cumbiambas, to visualize the execution of a subset of activities introduced by the viewpoint, and to present only parts of the process execution to the stakeholders. An initial step in this vein was done by J. Jimenez, who developed a traceability model to support the tracking of changes introduced by a viewpoint in a target process model.

9.4.7 Knowledge-Intensive Processes

Knowledge-intensive processes and process-oriented knowledge management are relatively new research areas, focused on the modeling of processes that demand an intensive use of knowledge-management techniques. We believe that our viewpoint proposal could be used to define these types of concerns, which would require the integration not only of activities, resources and products, but also the use of ontologies and repositories of learning objects.

9.4.8 Formal Verification

A final line of research derived from this work is the verification of the process model obtained after the weaving of the strategies. Similar works can be found in the YAWL project, which uses verification methods based on Petri nets to check that a process model is correct with respect of a set of concurrency theories. We believe that the same principle can be used to offer a formal verification.

Bibliography

- [AHKB03] W. M. P. Van Der Aalst, A. H. M. Ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distrib. Parallel Databases*, 14(11):5–51, 2003.
- [Ali05] Rashid A. Ali, N. M. A state-based join point model for aop. In *Workshop on Views, Aspects and Roles VAR (held with ECOOP 2005)*, 2005.
- [AOS07] AOSD. Aosd glossary. Web Site, 2007. <http://aosd.net/wiki/index.php?title=Glossary>.
- [APQ06] APQC. American Productivity and Quality Center (apqc). Web Site, 2006. <http://www.apqc.org>.
- [BBM03] Frank Budinsky, Stephen A. Brodsky, and Ed Merks. *Eclipse Modeling Framework*. Pearson Education, 2003.
- [BBS03] Microsoft Corporation BEA. BEA Systems, International Business Machines Corporation. Business process execution language for web services version 1.1. Technical report, 2003.
- [BCR94] Victor R. Basili, Gianluigi Caldiera, and H. Dieter Rombach. The goal question metric approach. University Of Maryland, 1994.
- [BPM06] BPMi. Business Process Modeling Initiative (bpmi). Web Site, 2006. <http://www.bpmi.org/>.
- [CKS06] Mary Beth Chrissis, Mike Konrad, and Sandy Shrum. *CMMI Guidelines for Process Integration and Product Improvement*,

- 2nd ed. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2006.
- [Coa02] WPMC. Workflow Management Coalition. Workflow process definition interface – xml process definition language version 1.0. Technical report, Workflow Management Coalition, 2002.
 - [Cor06] Marino O. Correal, D. Application of aspect oriented process viewpoints in e-learning systems. In *Learning Objects of the Future: Integrating Knowledge and Services*, 2006.
 - [CSN05] Kai Chen, Janos Sztipanovits, and Sandeep Neema. Toward a semantic anchoring infrastructure for domain-specific modeling languages. In *EMSOFT '05: Proceedings of the 5th ACM international conference on Embedded software*, pages 35–43, New York, NY, USA, 2005. ACM Press.
 - [Dav93] Thomas H. Davenport. *Process innovation: reengineering work through information technology*. Harvard Business School Press, Boston, MA, USA, 1993.
 - [DR06] H. Wu J. Gray. D. Rebernak, M. Mernik. Domain-specific aspect languages for modularizing crosscutting concerns in grammar. In *ACM International Conference on Generative Programming and Component Engineering (GPCE2006), October 2006*, 2006.
 - [DSBA05] Pascal Durr, Tom Staijen, Lodewijk Bergmans, and Mehmet Aksit. Reasoning about semantic conflicts between aspects. In *EIWAS 2005: 2nd European Interactive Workshop on Aspects in Software*, 2005.
 - [Ecl06] Eclipse. The Eclipse Modeling Framework (emf). Web Site, 2007 06. <http://www.eclipse.org/modeling/emf/>.
 - [Ecl05] Eclipse. The eclipse modeling framework (emf) overview. Technical report, Eclipse, 2005.
 - [Ecl06] Eclipse. The Eclipse Modeling Framework Overview. Web Site, 2006. <http://dev.eclipse.org/viewcvs/indextools.cgi/org.eclipse.emf/doc/org.eclipse.emf.doc/references/overview/EMF.html>.

- [Ecl07a] Eclipse. The ATL Project. Web Site, 2007. <http://www.eclipse.org/m2m/at1>.
- [Ecl07b] Eclipse. The Eclipse Modeling Framework Project. Web Site, 2007. <http://www.eclipse.org/modeling/emf>.
- [Eff06] Kadura C. Effinge, S. Openarchitectureware 4.1 xpanse language reference. Technical report, OpenArchitectureWare.org, 2006.
- [Eff07a] Sven Effinge. Openarchitectureware 4.1 extend language reference. Technical report, OpenArchitectureWare.org, 2007.
- [Eff07b] Sven Effinge. Xtext reference documentation. Technical report, OpenArchitectureWare.org, 2007.
- [Enh07a] Enhydra. Enhydra shark. Web Site, July 2007. <http://www.enhydra.org/workflow/shark/index.html>.
- [Enh07b] Enhydra. Jawe. Web Site, 06 2007. <http://www.enhydra.org/workflow/jawe/index.html>.
- [ESB04] M. J. Emerson, J. Sztipanovits, and T. Bapty. A mof-based metamodeling environment. *Journal of Universal Computer Science*, 10(10):1357–1382, 2004.
- [Fav04] Jean M. Favre. Towards a basic theory to model model driven engineering. *Workshop on Software Model Engineering*, October 2004.
- [Fon04] Silaghi R. Fondement, F. Defining model driven engineering processes. In *In Proceedings of the 3rd Workshop in Software Model Engineering. WiSME*, 2004.
- [GBNT01] Jeff Gray, Ted Bapty, Sandeep Neema, and James Tuck. Handling crosscutting constraints in domain-specific modeling. *Communications of the ACM*, 44(10):87–93, 2001.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
- [Gra05] Thomas R. Graziadei. *Aspect Oriented Model Weaver*. PhD thesis, Fachhochschule Vorarlberg GmbH, 2005.

- [Gro06] OMG. Object Management Group. Business process modeling notation (bpmn) specification. Final Adopted Specification dct/06-02-01, Object Management Group, February 2006.
- [HC93] Michael Hammer and James Champy. *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness, New York (NY), 1993.
- [HO93] William Harrison and Harold Ossher. Subject-oriented programming: a critique of pure objects. In *OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications*, pages 411–428, New York, NY, USA, 1993. ACM Press.
- [Hut06] DeLesley Hutchins. Partial evaluation + reflection = domain specific aspect languages. In *ACM International Conference on Generative Programming and Component Engineering (GPCE2006), October 2006*, 2006.
- [IBM03] IBM. Business process execution language for web services. Version 1.1, OASIS, May 2003.
- [IBM07a] IBM. Eclipse Development Platform. Web Site, 2007. <http://www.eclipse.org/>.
- [IBM07b] IBM. Rational Rose. Web Site, 07 2007. <http://www-306.ibm.com/software/awdtools/developer/rose/>.
- [IEE00] IEEE. Ieee standards description: 1471-2000 ieee recommended practice for architectural description of software-intensive systems-description. Technical report, IEEE, 2000.
- [IEE02] IEEE. Learning technology standards committee. ieee 1484.12.1-2002 draft standard for learning object metadata. Technical report, IEEE, 2002.
- [IG03] IMS-GlobalLearningConsortium. Ims learning design information model version 1.0. Technical report, IMS Global Learning Consortium, 2003.
- [IMS02] IMS. Ims reusable definition of competency or educational objective - information model. version 1.0. Technical report, IMS, 2002.

- [Ins07] Software Engineering Institute. Capability Maturity Model Integration (cmmi). Web Site, July 2007. <http://www.sei.cmu.edu/cmmi/>.
- [Iqb06] Elrad T. Iqbal, A. Modeling timing constraints of real-time systems as crosscutting concerns. In *AOM 2006*, 2006.
- [JGN03] Sandeep Neema Douglas C. Schmidt Aniruddha Gokhale Jeff Gray, Ted Bapty and Balachandran Natarajan. An approach for supporting aspect-oriented domain modeling. In *Generative Programming and Component Engineering (GPCE 2003)*, Springer-Verlag LNCS 2830, Erfurt, Germany, September 22-25, 2003, pp. 151-168., 2003.
- [Jim07] J. Jimenez. Traceability over business process model transformations. Technical report, University of Los Andes, 2007.
- [KK03] Per Kroll and Philippe Kruchten. *The Rational Unified Process Made Easy: A Practitioner's Guide to Rational Unified Process*. Addison-Wesley, 2003.
- [KLM⁺97] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. Aspect oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.
- [KM04] R. Koper and J. Manderveld. Educational modelling language: modelling reusable, interoperable, rich and personalised units of learning. *British Journal of Educational Technology*, 35(5):537–551, 2004.
- [KS] Christian Koppen and Maximilian Störzer. PCDiff: Attacking the fragile pointcut problem.
- [KV06] Audris Kalnins and Valdis Vitolins. Use of uml and model transformations for workflow process definitions. *TECHNIKA*, 2006:3, 2006.
- [Led01] Akos Ledecz. The generic modeling environment. In *WISP'2001, Budapest, Hungary*. IEEE, May 2001.

- [LGZ06] Yuehua Lin, Jeff Gray, and Jing Zhang. The embedded constraint language: A transformation language for visual models. Submitted for publication, 2006.
- [Lie96] Karl J. Lieberherr. *Adaptive Object-Oriented Software: The Demeter Method with Propagation Patterns*. PWS Publishing Company, Boston, 1996. ISBN 0-534-94602-X.
- [LK97] Cristina Videira Lopes and Gregor Kiczales. D: A language framework for distributed programming. Technical Report SPL97-010, P9710047, Palo Alto, CA, USA, February 1997.
- [MIT06] MIT. Open Process Handbook Initiative (ophi). Web Site, 2006. <http://ccs.mit.edu/ophi/index.htm>.
- [MMA07] Christof Momm, Robert Malec, and Sebastian Abeck. Towards a model-driven development of monitored processes,. In *8 Internationale Tagung Wirtschaftsinformatik (WI2007)*, Karlsruhe, February 2007., 2007.
- [NoM06] NoMagic. Magicdraw. Web Site, 06 2006. <http://www.magicdraw.com/>.
- [OAW07] OAW. The Openarchitectureware Framework. Web Site, 2007. <http://www.openarchitectureware.org/>.
- [oLA07] Software Construction Group University of Los Andes. Cumbia home site. <http://agamenon.uniandes.edu.co/cumbia/index.php>, 06 2007.
- [OMG03] OMG. Mda guide version 1.0.1. Technical report, OMG, 2003.
- [OMG05a] OMG. Initial submission for mof model to text transformation language rfp. Technical report, OMG, 2005.
- [OMG05b] OMG. Software process engineering metamodel specification version 1.1. Technical report, OMG, 2005.
- [OMG05c] OMG. Software process engineering metamodel (spem) version 1.1. Misc, 2005.
- [OMG06] OMG. Object constraint language version 2.0. Technical report, OMG, 2006.

- [Ora07a] Oracle. Jdeveloper. Web Site, 06 2007. <http://www.oracle.com/technology/products/jdev/index.html>.
- [Ora07b] Oracle. Oracle BPEL Process Manager. Web Site, 06 2007. <http://www.oracle.com/technology/products/ias/bpel/index.html>.
- [Org07] AspectC Organization. Aspect c++. Web Site, 06 2007. <http://www.aspectc.org/>.
- [OUN07] OUNL. Coppercore. Web Site, 2007. <http://coppercore.sourceforge.net/>.
- [Pau04] G. Paulusberger. Bpel - editor. Technical report, Sun Technology and Research Excellence Center (SunTREC), 2004.
- [Pie06] Samuel Pierre. *E-Learning Networked Environments and Architectures: A Knowledge Processing Perspective (Advanced Information and Knowledge Processing)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.
- [Pro06] AspectJ Project. Aspectj language documentation, February 2006.
- [PWR05] M. Dumas A.H.M. ter Hofstede P. Wohed, W.M.P. van der Aalst and N. Russell. Pattern-based analysis of bpmn - an extensive evaluation of the control-flow, the data and the resource perspectives. Technical Report BPM Center Report BPM-05-26, BPM Center, 2005.
- [RtHEvdA05] Nick Russell, Arthur H. M. ter Hofstede, David Edmond, and Wil M. P. van der Aalst. Workflow data patterns: Identification, representation and tool support. In *ER*, pages 353–368, 2005.
- [RvdAtHE05] Nick Russell, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and David Edmond. Workflow resource patterns: Identification, representation and tool support. In *CAiSE*, pages 216–232, 2005.
- [SA06a] Retschitzegger W. Wimmer M. Schauerhuber A., Schwinger W. A survey on aspect-oriented modeling approaches. Technical report, January 2006.

- [SA06b] Kapsammer Elisabeth Retschitzegger Werner Wimmer Manuel Schauerhuber Andrea, Schwinger Wieland. Towards a common reference architecture for aspect-oriented modeling. In *8th International Workshop on Aspect-Oriented Modeling, in conjunction with AOSD'06, Bonn, Germany, March 21.*, 2006.
- [Sei03] Ed Seidewitz. What models mean. *IEEE Software*, 20(5):26–32, 2003.
- [SK97] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *Computer*, 30(4):110–111, 1997.
- [SKB⁺95] J. Sztipanovits, G. Karsai, C. Biegl, T. Bapty, A. Ledeczi, and A. Misra. Multigraph: an architecture for model-integrated computing. *iceccs*, 00:361, 1995.
- [SKF96] Janos Sztipanovits, Gabor Karsai, and Hubertus Franke. Model-integrated program synthesis environment. *ecbs*, 00:348, 1996.
- [SKVS95] Ian Sommerville, Gerald Kotonya, Steve Viller, and Peter Sawyer. Process viewpoints. In *European Workshop on Software Process Technology*, pages 2–8, 1995.
- [Sof07] SoftCom. C-saw. Web Site, 2007. <http://www.gray-area.org/Research/C-SAW/>.
- [Tel07] Teluq. Téléunivertité. Web Site, July 2007. <http://www.teluq.quebec.ca/>.
- [TOHJ99] Peri L. Tarr, Harold Ossher, William H. Harrison, and Stanley M. Sutton Jr. N degrees of separation: Multi-dimensional separation of concerns. In *International Conference on Software Engineering*, pages 107–119, 1999.
- [Uni07a] Uniandes. Los andes university. Web Site, July 2007. <http://ingles.uniandes.edu.co/>.
- [Uni07b] Los Andes University. Qualdev group. Web Site, July 2007. <http://chie.uniandes.edu.co/~changeset/wiki/doku.php>.

- [Uni07c] Los Andes University. Software construction research group (csw). Web Site, July 2007. <http://agamenon.uniandes.edu.co/~csw/index.php>.
- [UQA07] UQAM. Univertié du québec a montréal. Web Site, July 2007. <http://www.uqam.ca/>.
- [vdAH02] Wil M. P. van der Aalst and A. H. M. Ter Hofstede. YAWL: Yet another workflow language. QUT Technical report FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2002.
- [vdBC05] Klaas van den Berg and Jose Maria Conejero. A conceptual formalization of crosscutting in aosd. In *DSOA'2005 Iberian Workshop on Aspect Oriented Software Development*, Technical Report TR-24/05, pages 46–52. University of Extremadura, 2005. ISBN 84-7723-670-4.
- [vdBCH06] Klaas van den Berg, José María Conejero, and Juan Hernández. Identification of crosscutting in software design. In *8th International Workshop on Aspect-Oriented Modeling*, Bonn, Germany, 2006. March 21, 2006.
- [Vil06] Jorge Villalobos. Cumbia-xpm: Model specification draft version 0.1. Technical report, Universidad de Los Andes, 2006.
- [WfM95] WfMC. The workflow reference model. Technical report, Workflow Management Coalition, 1995.
- [WFM07] WFMOpen. Wfmopen. Web Site, 2007. <http://wfmpopen.sourceforge.net/>.
- [WGRM05] H. Wu, Jeff Gray, S. Roychoudhury, and M. Mernik. Weaving a debugging aspect into domain-specific language grammars, 2005.
- [Wik07] Wikipedia. Model-driven engineering — wikipedia, the free encyclopedia, 2007. [Online; accessed 30-January-2007].
- [WPI07] WPI. Workflow Pattern Initiative. Web Site, 2007. <http://www.workflowpatterns.com/>.

- [Zha06] Cottenier T. van den Berg A. Gray J. Zhang, J. Aspect interference and composition in the motorola aspect-oriented modeling weaver. In *Workshop on Aspect-Oriented Modeling at the 9th International Conference on Model Driven Engineering Languages and Systems, Milan, Italy*, 2006.

Appendix A

The AspectViewpoint Grammar

```
Viewpoint:
  "viewpoint" name = ID
  "affiliation" affiliation = ID
  "targetprocess" targetprocess = ID
  "{"
    (pointcuts+=Pointcut)+
    (strategies+=Strategy)*
  "}"
;

Dependency:
  from=ID "dependsOn" to=ID ";"
;

Pointcut:
  "pointcut" name = ID "(" parameter=Parameter ")" ":"
  functions+=PointcutFunction
  (
    "&&"
    functions+=PointcutFunction
  )*
  ";"
;

Abstract BpmnType :
  TaskType | SubProcessType |
  ArtifactType | GatewayType |
  EventType | SequenceType
;

PointcutFunction:
  selection = SelectionFunction
;

Abstract SelectionFunction:
  TypeFunction | NameFunction |
```

```

        StatusFunction | MarkerFunction |
        TargetFunction
    ;

Strategy:
    "strategy" name=ID
    "(" parameter=Parameter ")"
    ","
    execution = ExecutionType
    pointctuRef=ID "(" argument = StrategyArgument)"
    ( "follows" "(" ID ")" ) *
    "{"
        (declarations+=Declaration)*
        (newstatements+=NewStatements)*
        (patternstatements+=PatternStatement)*
    "}"
;

NewStatements:
    (variable = ID "=")*
    context=ID ( "." "parent" "(" " )" ) +
    "." "newFlowObject"
    "(" elementname=STRING ","
        elementtype=STRING ","
        elementclass=STRING
    ")"
    ","
;

PatternStatement:
    context=ID "." pattern=Pattern ";"
;

Declaration:
    type=BpmnType
    variables+=Variable
    ( "," variables+=Variable ) *
    ","
;

Variable:
    name=ID
;

Pattern:
    patterntype = PatternType
    "("
        elements+=Branch ( "," elements+=Branch ) *
    ")"
;

Abstract Branch:
    "[" object=ID "," condition =STRING "]"
;

GateBranch:
    method = "[" object=ID "," condition =STRING "]"

```

```

;

SequenceBranch:
    method = "addSequenceFlow"
    "(" object=ID ("," condition =STRING)* ")"
;

Abstract ExecutionType:
    ExecutionTypeAfter |
    ExecutionTypeBefore |
    ExecutionTypeAround
;

Abstract Activity:
    NewActivity | ContextActivity
;

ContextActivity:
    contextobject = ID "."
    method = "Successor" "(" ")"
;

NewActivity:
    type = "newFlowObject"
    "(" name=STRING "," object=STRING "," type=STRING ")"
;

TypeFunction: name="type" "("argument=ID)";
NameFunction: name="name" "("argument=STRING)";
StatusFunction: name="status" "("argument=ID)";
MarkerFunction: name="marker" "("argument=ID)";
TargetFunction: name="target" "("argument=ID)";

TaskType: name = "Task";
SubProcessType: name = "SubProcess";
ArtifactType: name = "Artifact";
GatewayType: name = "Gateway";
EventType: name = "Event";
SequenceType: name = "SequenceFlow";

ExecutionTypeAfter: name = "after" ;
ExecutionTypeAround: name = "around";
ExecutionTypeBefore: name = "before";

StrategyArgument: name=ID;
Parameter : type=BpmnType name=ID;

Enum PatternType :
    Sequence = "Sequence" |
    ArbitraryCycles = "ArbitraryCycles" |
    ParallelSplit="ParallelSplit" |
    Synchronization = "Synchronization" |
    ExclusiveChoice = "ExclusiveChoice" |
    Merge = "Merge" |
    MultiChoice = "MultiChoice" |
    DeferredChoice = "DeferredChoice"
;

```


Appendix B

Viewpoint Transformation

```
// This rules transform AVL models into PWE models

import pwe;
import aspectviewpoint;
extension org::aspectviewpoint::AVLExtensions;
extension org::openarchitectureware::uml2ecore::io;

create pwe::TargetProcess this transform (Viewpoint vp):
setName(vp.targetprocess) ->
setConcern(vp.createAspect())
;

create pwe::Aspect createAspect (Viewpoint vp):
setName(vp.affiliation) ->
pointcut.addAll (vp.allPointcuts().createPoint()) ->
rules.addAll (vp.allStrategies().createRule())
;

create pwe::AdaptationRule createRule (Strategy st):
setName (st.name+ "_" + st.pointctuRef) ->
setPosition (st.execution.name) ->
setPointcutRef (st.pointctuRef) ->
setContextType (st.parameter.type.name) ->
setContextObj (st.parameter.name) ->
adaptations.addAll (st.allPatternStatements().createCompositeAdaptation()) ->
adaptations.addAll (st.allNewStatements().createStructuralAdaptation())
;

create pwe::StructuralAdaptation createStructuralAdaptation (NewStatements ns):
setName ("NewStatement") ->
setResult (ns.variable) ->
setContext (ns.context)->
setAdaptation ("newFlowObject") ->
setAdaptedElement (ns.createElement ())
;
```

```

create pwe::Element createElement (NewStatements ns):
    setName (ns.elementname) ->
    setType (ns.elementtype)->
    setClass (ns.elementclass)
;

create pwe::CompositeAdaptation createCompositeAdaptation (PatternStatement ps):
    setName (ps.pattern.patternType()) ->
    childAdaptation.addAll (ps.pattern.allElements().createStructuralAdaptation(ps.context))
;

create pwe::StructuralAdaptation createStructuralAdaptation(Branch elem , String caller):
    setName ("PatternStatement")->
    setContext(caller) ->
    setObject(elem.object) ->
    setAdaptation ("SequenceFlow") ->
    setInfo (elem.condition)
;

create pwe::Pointcut createPoint (aspectviewpoint::Pointcut vp):
    setName(vp.name) ->
    setContextType (vp.parameter.type.name) ->
    method.addAll (vp.allPointcutFunctions().createSelectionMethod())
;

create pwe::Pointcut createPointcutRef (Strategy st):
    setName(st.pointctuRef)
;

create pwe::SelectionMethod createSelectionMethod (SelectionFunction sf):
    setName(sf.name) ->
    setExpression(sf.argument)
;

create pwe::Adaptation createAdaptation (Pattern se):
    setName (se.patternType())
;

```