

# Application of policy-based techniques to process-oriented IT service management

---

## Dissertation

an der

**Fakultät für Mathematik, Informatik und Statistik der  
Ludwig-Maximilians-Universität München**

vorgelegt von

**Vitalian Danciu**

Tag der Einreichung: 6. Juli 2007  
Tag der mündlichen Prüfung: 27. Juli 2007

1. Berichterstatter: **Prof. Dr. Heinz-Gerd Hegering**  
Ludwig-Maximilians-Universität München
2. Berichterstatter: **Prof. Dr. Bernhard Neumair**  
Universität Göttingen



# Thanks

A thesis—like any piece of written work—is seldom the result of a single person’s isolated efforts. Rather, it is influenced by the culture in the surroundings where it is written and the people who offer comments and advice on early (and not-so-early) versions of the work.

I wrote this dissertation while working as a research and teaching assistant at the chair of Prof. Dr. Heinz-Gerd Hegering, who supervised my work. I am indebted to Prof. Hegering for his committed guidance and kind advice during all phases of my work, as much as for maintaining at his chair an extraordinary work and learning environment.

I express a heartfelt thanks to Prof. Dr. Bernhard Neumair who provided important suggestions and who cheerfully put up with tight time frames for reviewing earlier versions of the text.

During my time in the MNM Team, I have enjoyed the community and help of my team colleagues, past and present. Undoubtedly, this thesis has profited greatly from discussions and debates within the Team.

I am grateful to my family and friends for their support during the preparation of this thesis and especially for their patience during the intensive time of writing.



## Zusammenfassung

Rechenzentren und Telekommunikationsanbieter sowie die IT-Abteilungen der meisten größeren Organisationen setzen in immer höherem Maße organisatorische Mittel ein, um den Betrieb und die Wartung ihrer Infrastrukturen zu koordinieren. Eines der verstärkt eingesetzten Instrumente ist das prozessorientierte IT-Dienstmanagement, bei dem IT-Managementprozesse die Abläufe in der IT-Organisation steuern. Diese können mittels einer der formalen Prozessbeschreibungssprachen festgehalten werden, wie sie für die Dokumentation von Geschäftsprozessen bereits entwickelt wurden. Ein derart formalisierter Prozess kann mit Hilfe von Werkzeugen unterstützt werden, wie sie für die Umsetzung von Geschäftsprozessen zur Verfügung stehen.

Der Spezialfall eines IT-Managementprozesses weist allerdings Eigenschaften auf, die ihn von allgemeineren Geschäftsprozessen unterscheiden. Seine Prozessaktivitäten werden einerseits von technisch geschultem Personal durchgeführt, andererseits bieten die inhärent auf den IT-Betrieb ausgerichteten Aktivitäten ein hohes Automatisierungspotential. Eine Automatisierung ist nicht nur aus Kostengründen wünschenswert; auch Konsistenz und Reproduzierbarkeit der Abläufe wären dadurch in höherem Maße gewährleistet. Dieses Potential kann allerdings nur ausgeschöpft werden, wenn die technischen Verfahren im IT-Betrieb mit dem Ablauf der Prozesse eng verbunden werden. Eine halb- oder vollautomatische Umsetzung von Prozessaktivitäten wird prinzipiell möglich, wenn die technischen Teilabläufe in der Prozessbeschreibung ausdrücklich berücksichtigt werden und somit die Voraussetzung für eine in den regulären Ablauf des Prozesses eingebundene Ausführung von Managementaktionen auf der zu betreibenden Infrastruktur geschaffen wird.

Die vorliegende Arbeit entwickelt einen Ansatz zur flexiblen Realisierung von IT Service Management Prozessen mittels operationaler Management Policy. Der Ansatz berücksichtigt die Weiterverwendung bereits vorhandener Managementinfrastruktur und verfolgt die Automatisierung der Prozessausführung. Dabei wird die hohe Frequenz an Änderungen (change) berücksichtigt, der heutige IT Organisationen unterliegen.

Die Konsistenz und Kosteneffizienz der Prozessrealisierung wird durch einen musterbasierten Übersetzungsmechanismus erreicht, der

detailliert ausgearbeitete Spezifikationen von Prozessen, die in einer formalen Sprache vorliegen, in Management Policy Regeln überführt. Da das Übersetzungsverfahren eine Fragmentierung des Prozesses zur Folge hat, wird dabei auch die Erhaltung des Informationsflusses im Prozess berücksichtigt.

Die Übersetzungs- und Ausführungskonzepte für Managementprozesse werden in einer funktionalen Architektur zusammengefasst, deren Bausteine den einzelnen Bausteinen im Verfahren entsprechen. Die Umsetzung der Architektur wird exemplarisch anhand eines Managementsystems illustriert, dessen Komponenten teils in Zuge der vorliegenden Arbeit entstanden, teils gängige Elemente einer Managementinfrastruktur darstellen.

## Abstract

In the perpetual quest to reduce operational cost, IT centres as well as telecommunication providers increasingly shift their IT management focus away from the technical plane. The most important goal today is the alignment of IT operations to the organisational or business needs. The instrument of choice to achieve this objective is the introduction of IT Service Management Processes, intended to govern the procedures of the IT organisation. The introduction of processes draws on the rich assortment of practices and tools already developed for business processes. In many cases IT management processes are treated as an additional kind of business processes. Although this view may be effective to some extent, it fails to acknowledge the special characteristics of processes applicable to IT management. In contrast to many a business process that is merely supported by IT, the processes executed to manage IT itself exhibit a number of opportune details: they are executed by personnel with an affinity to IT, whose object of management, the IT infrastructure, offers a high automation potential.

This work presents an approach to the flexible realisation of IT Service Management Processes by means of management policy. It takes into account the reuse of existing management infrastructure, targets the automation of process execution and aims to support the high rate of change in present IT organisations. To allow consistent and cost effective process realisation a pattern-based translation mechanism has been devised that transforms detailed, formal process specifications into management policy rules in an automated manner. As contiguous process specifications are fragmented during the translation procedure, special attention has been given to the preservation of the information flow within the process.

The projection of the concepts of translation and execution of management processes onto a tool set is concentrated into an architecture encompassing the functional building blocks necessary for the task. The practical realisation of that architecture is illustrated by an exemplary management system that includes components specific to this work, as well as common management infrastructure elements.





# Contents

<b>Contents</b>	<b>ix</b>
<b>I. Foundation</b>	<b>1</b>
<b>1. Introduction</b>	<b>5</b>
1.1. Problem statement . . . . .	10
1.2. Approach Outline . . . . .	13
1.3. Subproblems and results . . . . .	19
1.4. Structure of this work . . . . .	21
<b>2. Scenarios and requirements analysis</b>	<b>25</b>
2.1. Inter-domain application service management . . . . .	26
2.1.1. Management processes and tools . . . . .	26
2.1.2. Roles, relationships, and interfaces . . . . .	28
2.1.3. Challenges . . . . .	31
2.2. Grid management . . . . .	33
2.2.1. Management arrangements . . . . .	37
2.2.2. Management challenges . . . . .	38
2.2.3. Summary . . . . .	41
2.3. Practical example . . . . .	42
2.3.1. Setting . . . . .	42
2.3.2. Example process partition . . . . .	43
2.3.3. Automation of activities . . . . .	45
2.3.4. Tools . . . . .	46
2.3.5. Challenges . . . . .	47
2.4. Requirements . . . . .	47
2.4.1. Requirements catalogue . . . . .	49
2.4.2. Weighting of requirements . . . . .	58
2.4.3. Discussion . . . . .	59
<b>3. Related work</b>	<b>61</b>
3.1. Reference process frameworks . . . . .	62

## CONTENTS

3.1.1.	IT Infrastructure Library . . . . .	63
3.1.2.	Extended Telecom Operations Map . . . . .	70
3.2.	Formalisms for process representation . . . . .	71
3.2.1.	UN/CEFACT and OASIS . . . . .	71
3.2.2.	OMG and BPMI . . . . .	72
3.2.3.	Workflow Management Coalition . . . . .	73
3.2.4.	IDS Scheer . . . . .	75
3.2.5.	Interrelations of process formalisms . . . . .	75
3.2.6.	Process maturity . . . . .	77
3.2.7.	Summary . . . . .	78
3.3.	Pattern in processes . . . . .	78
3.4.	Policy Fundamentals . . . . .	80
3.4.1.	Policy refinement . . . . .	81
3.4.2.	Policy conflicts . . . . .	82
3.4.3.	Architecture for policy-based management . . . . .	83
3.4.4.	Standardisation efforts . . . . .	84
3.5.	Policy Languages . . . . .	85
3.5.1.	Ponder . . . . .	85
3.5.2.	Rei . . . . .	86
3.5.3.	<i>PDL</i> . . . . .	87
3.5.4.	ProPoliS . . . . .	87
3.5.5.	XACML . . . . .	88
3.6.	Summary and appraisalment . . . . .	88

## **II. Elaboration 91**

### **4. Process translation 95**

4.1.	Analysis of process and policy formalisms . . . . .	97
4.1.1.	Requirements of IT Management processes . . . . .	98
4.1.2.	Basic elements . . . . .	98
4.1.3.	Assessment of process formalisms . . . . .	106
4.1.4.	Analysis conclusions . . . . .	113
4.1.5.	Requirements for policy formalisms . . . . .	115
4.1.6.	Assessment of policy formalisms . . . . .	120
4.1.7.	Summary of analysis results . . . . .	127
4.2.	Meta-models of process representation . . . . .	128
4.2.1.	Simple meta-model for process definitions . . . . .	129
4.2.2.	Target meta-model . . . . .	129
4.3.	Substitution rules . . . . .	131
4.3.1.	Context-free substitution . . . . .	132
4.3.2.	Context-sensitive substitution . . . . .	135

4.3.3.	Order of application . . . . .	137
4.4.	Methodology for translation . . . . .	137
4.4.1.	Outline . . . . .	137
4.4.2.	Description of steps . . . . .	138
4.4.3.	Discussion . . . . .	142
4.5.	Fundamental patterns . . . . .	142
4.5.1.	Basic pattern . . . . .	143
4.5.2.	Condition patterns . . . . .	144
4.5.3.	Synchronisation pattern . . . . .	149
4.5.4.	Discussion . . . . .	152
4.6.	Detection and translation . . . . .	153
4.6.1.	Fragment discrimination . . . . .	153
4.6.2.	Algorithms . . . . .	155
4.7.	The generating system . . . . .	161
4.7.1.	Elements and transformations . . . . .	162
4.7.2.	Demonstration of totality . . . . .	166
4.8.	Extending the pattern catalogue . . . . .	167
4.8.1.	Pattern substitution . . . . .	168
4.8.2.	Pattern extension mechanism . . . . .	169
4.9.	Translation example . . . . .	170
4.9.1.	Application of the substitution rules . . . . .	172
4.9.2.	Identifying patterns . . . . .	173
4.9.3.	Translation result . . . . .	175
4.9.4.	Optimisation . . . . .	178
4.10.	Summary . . . . .	178
<b>5.</b>	<b>Process data flow</b>	<b>181</b>
5.1.	Preservation of the information flow . . . . .	182
5.1.1.	Data/information items in processes . . . . .	183
5.1.2.	Attaching information specification to patterns	189
5.2.	Requirements on information transport . . . . .	190
5.2.1.	Dimensions of process data flow . . . . .	190
5.2.2.	Policy-based process realisation . . . . .	194
5.3.	Realisation of process data flow . . . . .	195
5.4.	Summary . . . . .	196
<b>6.</b>	<b>Architecture</b>	<b>199</b>
6.1.	Management process life-cycle . . . . .	200
6.1.1.	Initial workflow . . . . .	200
6.1.2.	Implementation phase . . . . .	202
6.1.3.	Change and iterative refinement . . . . .	203
6.1.4.	Decommission and retirement . . . . .	206

## CONTENTS

6.2. Functional components . . . . .	206
6.2.1. Process management station . . . . .	207
6.2.2. Management policy architecture . . . . .	209
6.2.3. Process-to-policy translator . . . . .	210
6.2.4. Facilities for information transport . . . . .	211
6.2.5. Tools . . . . .	213
6.2.6. Abstraction layers . . . . .	214
6.3. Interoperation . . . . .	215
6.3.1. Fundamental interactions . . . . .	215
6.4. Summary and discussion . . . . .	218
<b>III. Proof of Concept</b>	<b>221</b>
<b>7. Exemplary design</b>	<b>225</b>
7.1. Components overview . . . . .	227
7.2. SLPR – A minimal process language . . . . .	231
7.2.1. Language overview . . . . .	232
7.2.2. Grammar . . . . .	233
7.2.3. SLPR Example . . . . .	235
7.3. The Process-aware Policy System . . . . .	238
7.3.1. Language . . . . .	238
7.3.2. Components . . . . .	240
7.4. A facility for information aggregation . . . . .	243
7.5. Summary . . . . .	247
<b>8. Evaluation</b>	<b>249</b>
8.1. Fulfilment of requirements . . . . .	249
8.2. Issues and hazards . . . . .	254
8.3. Applicability . . . . .	257
<b>IV. Conclusions and Further Work</b>	<b>259</b>
<b>9. Future prospects</b>	<b>263</b>
9.1. Issues for further study . . . . .	263
9.1.1. Metrics for process detail . . . . .	263
9.1.2. Security and privacy considerations . . . . .	264
9.1.3. Independent policy . . . . .	264
9.2. Applications . . . . .	265
9.2.1. Generalisation . . . . .	265
9.2.2. Bottom-up assessment of tool requirements . . . . .	266

9.2.3. Self-management . . . . .	266
<b>10. Summary and conclusions</b>	<b>269</b>
<b>List of Figures</b>	<b>273</b>
<b>List of Tables</b>	<b>275</b>
<b>Bibliography</b>	<b>281</b>
<b>ProPoliS Schema</b>	<b>291</b>
<b>SISL Schema</b>	<b>301</b>
<b>Index</b>	<b>305</b>

# *CONTENTS*

# Part I

## Foundation





# Contents – Part I

<b>1. Introduction</b>	<b>5</b>
1.1. Problem statement . . . . .	10
1.2. Approach Outline . . . . .	13
1.3. Subproblems and results . . . . .	19
1.4. Structure of this work . . . . .	21
<b>2. Scenarios and requirements analysis</b>	<b>25</b>
2.1. Inter-domain application service management . . . . .	26
2.2. Grid management . . . . .	33
2.3. Practical example . . . . .	42
2.4. Requirements . . . . .	47
<b>3. Related work</b>	<b>61</b>
3.1. Reference process frameworks . . . . .	62
3.2. Formalisms for process representation . . . . .	71
3.3. Pattern in processes . . . . .	78
3.4. Policy Fundamentals . . . . .	80
3.5. Policy Languages . . . . .	85
3.6. Summary and appraisalment . . . . .	88

*CONTENTS – PART I*

# Chapter 1

## Introduction

**S**INCE IT services are becoming commodities in recent years, they are no longer provided within single organisations, or to single customers: they are bought and sold in an open market subject to the same criteria as any other product: quality, price, interoperability, support and so on. Hence, as with any other product, the 'manufacturers' of IT services strive to streamline 'production' to improve their product while at the same time reducing costs.

As organisations grow, restructure and merge, IT functions must cope with structural change transparently. Service delivery must therefore be adapted in such manner, that changes to the structure or business activities of the organisation are reflected in the management of its supporting IT functions. In order to concentrate on their core activities, organisations outsource IT functions to service providers. In such cases, structural change in an outsourcing organisation, or alteration of its focus concerns not only the organisation itself, but also the providers delivering IT services. Thus, an orchestrated effort on behalf of the changing organisation and the providers is necessary in order to effectively adapt to any change.

At the same time, infrastructure and software technology continues to evolve at a fast pace to meet the ever growing requirements of customers. In consequence, IT operators upgrade their infrastructure and software base frequently in order to keep in touch with the state of the art. While evolution in the application domain is driven by customer demand, it is in turn the driver of the adaption of management infrastructure and tools, creating over time a heterogeneous environment of management facilities. Interoperability between tools is desired in order to achieve a consistent management view of the production systems. Great effort must be expended to realise such interoperability, since coexisting new and legacy components must be accommodated at the same time.

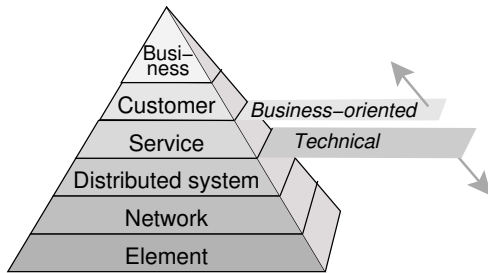


Figure 1.1.: Management pyramid

Different management disciplines, depicted in the Management Pyramid in Figure 1.1 (see e.g. [HAN 99, ITU M.3010]) require different views on the IT infrastructure and the management facilities. The views differ in the level of abstraction and the amount of detail contained. Obviously, all views refer to the same activities, executed on the same infrastructure. To keep them consistent, translations between the views of different disciplines are necessary.

Costly  
projection of  
management  
goals

In particular, management goals defined on higher levels of abstraction must be propagated toward the lower, more concrete levels. Management paradigms often include their own mechanisms for the projection of these goals onto operational management. Inherently, this projection is semantics-bound, since it enriches the management specification given on a more abstract layer with detail information necessary to the more concrete layer. Consequently, this refinement of management must be executed by persons with domain knowledge, thus incurring a large cost. In many cases, such refinement does not happen in an explicit, coordinated manner. Instead, the operational management specification is approximated from experience. Most often it lacks documentation and remains the knowledge of the small group of people in charge of it.

The implementation and deployment of a management system that satisfies the requirements of the organisation can be seen as just another projection step. Yet, it introduces the risk of errors and the problem of validation of the projection. The use of off-shelf management products facilitates short deployment time but introduces the problems of interoperability of the different products. Customised solutions, on the other hand, are expensive in terms of time as well as financially.

Change in requirements is often reflected by ad-hoc changes to the implementation. The addition of new management tools requires the implementation of interoperability between the newly added component and the installed base of tools. To deal with change in a consistent manner, projection of management goals onto operational management specification must be repeated for every change. After that, the modified specification must be realised in the management implementation. With each iteration, the implementation must be validated against the modified specification.

Management implementation targets an ever-changing specification.

In conclusion, the IT management efforts of today struggle to cope with frequent structural change in a heterogeneous, distributed environment. High paced technological and business model evolution, as well as the introduction of new services and new arising customer demands impose great challenges that are increasingly difficult to respond to.

Key issues: diversity; distribution; business, technological and structural dynamics.

In addition to the efforts expended towards the technical disciplines of element, network and systems management, an increase in the attention given to *organisational IT management* has been noted in recent years. Probably the most important aspect is the advent of process-oriented *IT Service Management (ITSM)*. Several best practices frameworks describe IT management reference processes that are introduced into today's IT centres at a quick pace. Their purpose is to make IT operations and planning repeatable, accountable—and of course lower their cost.

Increasingly, it is understood that IT operations need to be aligned to the needs of the “core” business of an organisation. In the business domain, processes have been employed for some time in order to specify and control the business operations of an organisation. The introduction of defined processes (e.g. business processes for its core operations) in an organisation implies the modelling of process activities, the identification of relevant roles and the act of determining process artefacts (as e.g. input and output of activities). The execution of thus modelled processes can be supported by means of workflow management systems that track the execution of process instances and information systems, e.g. database systems, that store information relevant to the workflows. This allows some control over the execution of due tasks by the persons holding the roles specified in a process. Figure 1.2 shows the hierarchy of management concepts after inclusion of process-oriented management. As indicated in the diagram, processes are intended to govern the management proce-

Process orientation: an emerging aspect in IT management

dures performed by administrators and supported by a tool set, in order to fulfil business requirements.

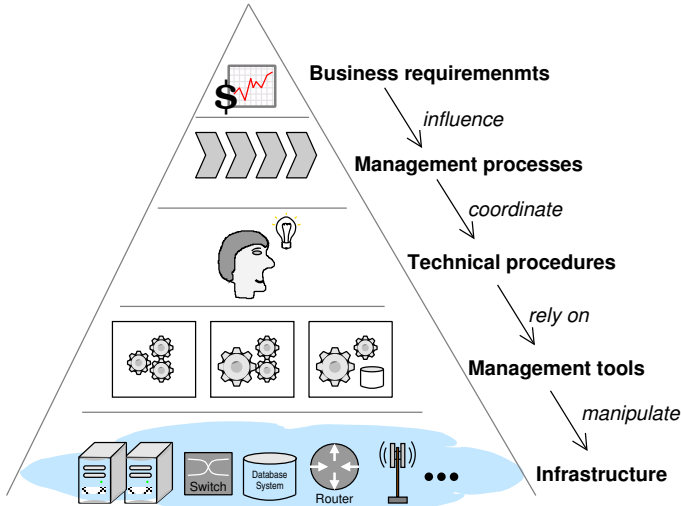


Figure 1.2.: IT management hierarchy

Best practices collections

The larger-scale projection of the process-oriented method onto IT management is being promoted and catalysed by the availability of reference process frameworks. As sources of “best practices”, they provide guidance towards specifying IT management processes. They propose process structure, including the division of IT operations into processes and activities; they define roles associated with these processes and their activities; and they describe generic interfaces between processes, as well as artefacts of their activities.

Process customisation

Being targeted at a broad audience with quite varying IT management requirements, the best practices frameworks are inherently kept generic and adaptable. They need to be tailored to the realities of the IT organisation applying them, as shown in Figure 1.3. In particular, an IT organisation that customises reference processes needs to decide which processes to implement, how to populate the roles specified in the reference, how to provide tool support for process execution and so on. This effort produces a set of detailed IT

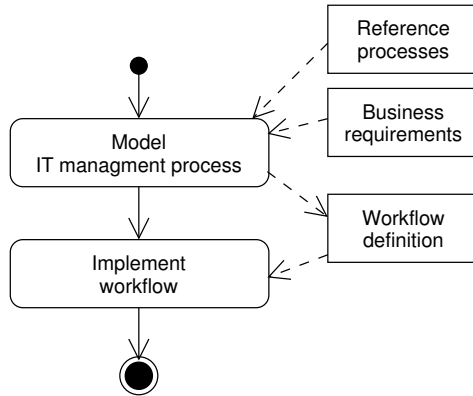


Figure 1.3.: Creation of an IT management process  
 Bird's eye view: Creation of an IT management process

management process definitions that the IT organisation plans to deploy.

Being part of a fairly new concept in the IT domain, such processes tend to be treated and implemented like any other business process. In order to achieve the goals that motivate the introduction of management processes in the first place, the differentiation between the characteristics of IT management processes and “other” processes is compelling. Two obvious, generic features of IT management processes can be identified. First, they focus on planning and operation of their management targets: IT infrastructures and services. Second, they are executed by personnel that is knowledgeable in the field of IT.

High potential  
for automation

These features suggest a high potential for automation in process-oriented IT management, if only the operational procedures executed within process activities can be integrated, seamlessly if possible, into the management workflows. To fully exploit the benefits of process-oriented management, its integration with technical management procedures is a necessity; otherwise, the decisions and actions defined within process activities will have to be manually mapped onto technical management actions again and again. The implied labour cost of such mappings will be supplemented by susceptibility to (human) error and delays.

Integration  
with technical  
IT  
management

## 1.1. Problem statement

The integration of technical and process-level management faces the challenges posed to technical management disciplines as well: they are introduced as a consequence of linking the high-level management workflows to technical management tasks. It is therefore necessary to combine the process-level management with techniques that address the technical issues present in operational management and provide support for the execution of the processes themselves at the same time.

Today, the introduction and implementation of process-oriented management techniques still maintains a certain degree of novelty. The dedication to process-oriented management exhibited by many organisations has already resulted in development of tools and tool sets for IT management process support. As these tools evolve and become entrenched in the market it is probable that the heterogeneity found in current systems and network management tools will become an issue with tools for process support. In addition to different API and data format definitions the reliance on different reference frameworks, different versions of those frameworks as well as divergent interpretations and levels of compliance will pose problems for the implementation of IT management processes. In essence, this work addresses the two challenges sketched above:

- ◇ to integrate process-oriented management with the existing procedures of technical IT management; and
- ◇ to address the heterogeneity of tools emerging in the domain of process-oriented management.

To address the coupling of technical and process-oriented management against the background of a diverse process support landscape, several aspects must be considered. Figure 1.4 illustrates the dimension space that envelops the management challenges.

**IT management process** Different processes have different affinity to technical IT management procedures. Current reference process frameworks specify operational, lower level processes that can be found to have greater potential for automation than the higher-level, tactical and strategic processes [Bren 06]. The type of process—whether it was derived from a reference framework, or defined without their use—has, in consequence, an impact on the degree of attainable automation support.



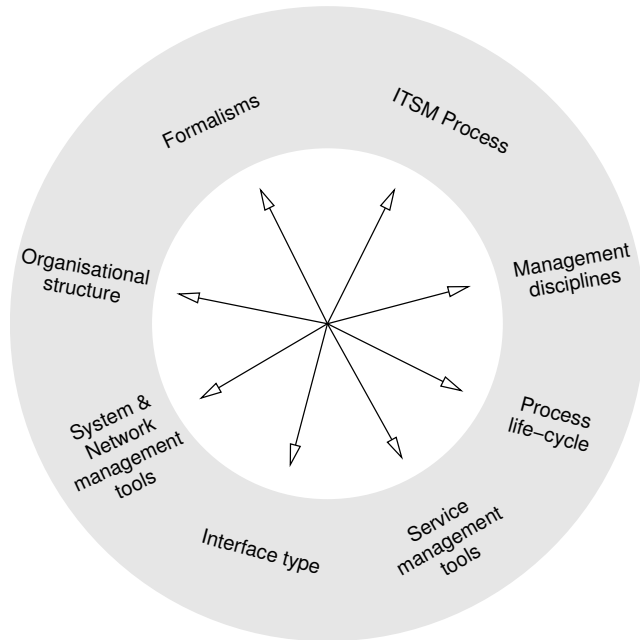


Figure 1.4.: Dimension space of management challenges

**Management discipline** Current frameworks focus on reference processes for “service management”. They leverage the concept of *service* in order to direct management activity towards the provisioning and operation of entities chargeable to customers. In fact, the same frameworks provide reference for activities in the disciplines of application and networks/systems management as well. Process-orientation has been applied in business management for some time, though this discipline is out of the scope of this work. In summary, the approach must consider several of the management disciplines shown in Figure 1.1.

**Process life-cycle** The existence of any IT management process can be described along a life-cycle. Beginning with its design, a process definition (or *workflow*) traverses several states, including being deployed, executed, and retired, as shown in Figure 1.5. Process execution and automation support must

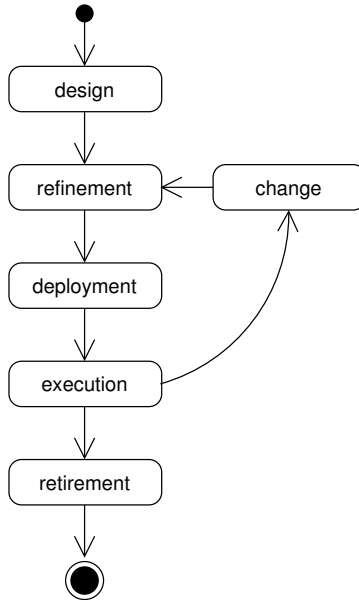


Figure 1.5.: The life-cycle of an IT management process definition

take into account the state of a process along this life-cycle. In particular, change administered to a process must be reflected.

**Service management tools** In most cases, process execution will be supported by software tools. The nature of these tools, as determined by their feature set, vendor, and capabilities impact process automation—and in some cases limit process design.

**Interface types** To support the operations of an entire organisation, many tools are typically deployed, that not only differ in scope and feature set, but also in the type of their interfaces, e.g., they may use different communication middleware and different data formats.

**System & Network management tools** Process activities may include automatable, technical management procedures that are supported by specific management tools. To interface process execution with tools for technical management, the capabilities and interfaces of such tools must be taken into account.

**Organisational structure** IT centres serve organisations that are increasingly flexible in terms of geographical distribution and intra-organisational divisions. Organisational change stresses even well-prepared IT processes by forcing re-validation of the assumptions used for their definition. Hence, the structure and dynamics in an organisation generates requirements on the handling of IT management processes.

**Formalisms** Processes may be documented in one of the existing formal workflow languages in order to allow execution and automation support. These formalisms differ in scope and expressive power. Hence, the choice of formalism for process representation may support or limit the process's potential for automated execution.

From the discussion of these dimensions it appears obvious that management systems should be constructed such, that, beside being flexible they take into account the management processes, as well as automatable routine procedures.

## 1.2. Approach Outline

One management strategy that appears flexible enough to face these challenges is found in the specification of management policy. In *Policy-based Management (PbM)* small, self-contained rules can be specified in order to influence the state and behaviour of a system. As a stand-alone approach, policy-based management has not been very successful, although conceptually it can greatly ease management endeavours at multiple levels of abstraction. The flexibility of management policy rules is supplemented by the fairly simple execution model. Grouping and abstraction mechanisms such as role and domain definitions can be easily supported, and the principle of management by means of rules is applicable to virtually any deployment environment.

The basic approach of this work consists in the representation of IT management processes by means of management policy rules. As suggested in Figure 1.6, the approach provides a path to policy-based process execution (i.e. automation), while bypassing the encumbrances (see Section 3.4) that have weighed down pure policy-based management.

Process  
execution by  
means of  
management  
policy

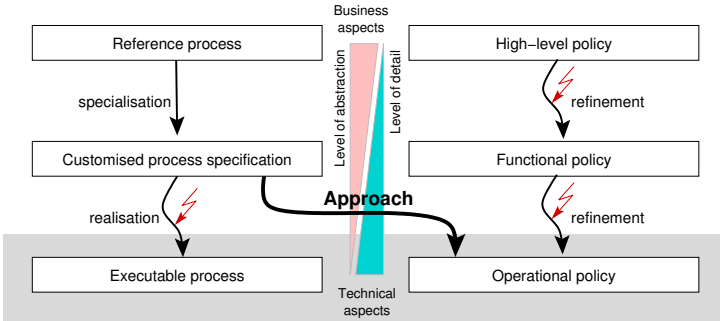


Figure 1.6.: Approach rationale

Given a certain level of detail in process specifications, technical management procedures can be integrated into the process execution. As a side-effect, the approach allows the coexistence of process-oriented management and policy-based management in order to combine the benefits delivered by each of them. The policy-based representation of management processes is attained by generating policy rules from the process specification. Rule generation is achieved by means of translation patterns developed in this thesis.

The remainder of this section sketches the core idea of and the idea behind this work. In particular, it outlines the two topics central to this thesis and highlights the issues to be dealt with. After discussing how the trend towards process-orientation delivers management specification for an increasing number of organisations, we focus on how implementation of processes can be realized by employing policy-based management.

### Leveraging the success of process orientation

Organisations increasingly model their management processes.

In recent time, the benefits of process oriented management have become apparent to more and more organisations. These, be they corporations, government agencies or universities, strive to improve service level and at the same time cut costs. Experience shows that these goals can be achieved simultaneously by modelling, documenting and deploying IT management processes. Cost savings are experienced when training employees since the process documentation can be used as a reference. Also, cost savings result from less service

down-time, since well-defined processes ensure timely, orchestrated response to incidents in addition to more organised efforts in service planning and deployment.

In addition, certifications are available for organisations truly adhering to process reference standards. Process frameworks are available and have a high acceptance. Prominent examples include the *IT Infrastructure Library (ITIL)*, a general process framework and the *Enhanced Telecom Operations Map (eTOM)*, which is targeted primarily at telecom providers. Both standards (see Section 3.1) constitute collections of best practices. To be applicable to various organisations, the process definitions are generic—in the case of the ITIL they are even textual, abstaining from formal means.

Therefore, organisations customise the process definitions found in the frameworks to closely match the scope of their fields of activity and organisational structure. The standards provide an environment for this adaptation and customisation by defining different abstraction levels for processes; e.g. eTOM specifies strategic, tactical and operational processes to differentiate between different levels of detail. Operational processes are on the most concrete level of the hierarchy. They specify the management processes of an organisation in detail and constitute a representation of IT management adapted to the needs of the organisation. If implemented correctly, customised operational processes deliver “management the way the organisation wants it”.

Generic standards → process customisation

However, process implementation is not a trivial task – especially in the context of large organisations. Most often, a management tool suite is adapted to fit the process requirements. Alternatively, tools are evaluated and somehow integrated in a suite that satisfies the process requirements. These approaches obviously constitute compromises between a faithful implementation of the processes defined and an adequate level of automation, as well as fast deployment. In addition, changes in strategy or redefinition of business goals often result in change to management processes. Therefore, after remodelling the relevant process parts, the changes have to be reflected in the implementation of the tool set. This introduces a high implementation and maintenance overhead for the deployed management processes.

Process implementation and change are problematic.

In conclusion, from an integrated management point of view, the specification of operational processes constitutes merely a resource. Substantial additional work is necessary until practical deployment becomes effective and efficient.

## Idea

Since management processes are actually defined and customised, they can serve as a source of detailed management specification. From operational management process definitions, operational policies can be derived by translating the representation of processes to policy sets. By automating this translation, the effort expended towards implementation and change in management can be minimised.

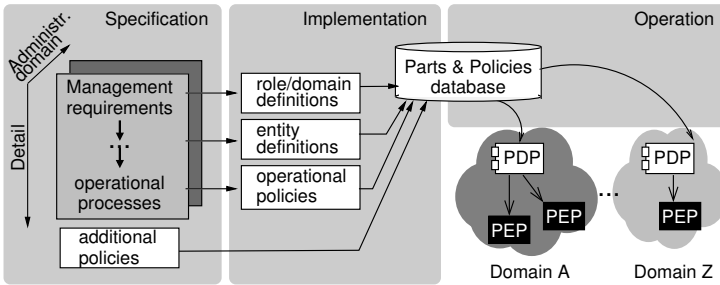


Figure 1.7.: Phases of process-to-policy translation

An overview of the approach is depicted in Figure 1.7. From the customisation of management processes (leftmost box in the figure), entity, role and domain definitions relevant to the processes are derived. The most detailed form of the processes is translated into operational policy sets stored in a policy repository. The policies generated in this way can be enforced using a PbM architecture (rightmost box in the figure) consisting canonically of the aforementioned repository, policy decision points (PDP) that decide which management actions are to be executed, and policy enforcement points (PEP). These are in principle agents designed to execute management actions on the resources targeted by the policies. Distribution of this basic architecture can be employed to account for requirements from organisational structure.

## Overview of process-to-policy translation

The conceptual and practical realisation of the approach does raise a number of issues. As a prerequisite, formal representations of processes as well as of policies need to be evaluated, in order to find a common semantic mapping for translation. In particular, the expressiveness of a process language and that of a policy language must allow mapping of process expressions onto one or more policy expressions. Existing formalisms for processes and policies are presented in Sections 3.2 and 3.5. Analysis of these formalisms in Section 4.1.2 yields common, basic formal elements that are instrumental to automated translation. Once the necessary expressiveness of languages has been ascertained, a source language for process description as well as a target policy language are selected. Exemplarily, these are used to illustrate the subsequently described concepts.

Evaluation of languages

A pattern based approach is used for deriving policies from pro-

Pattern based translation

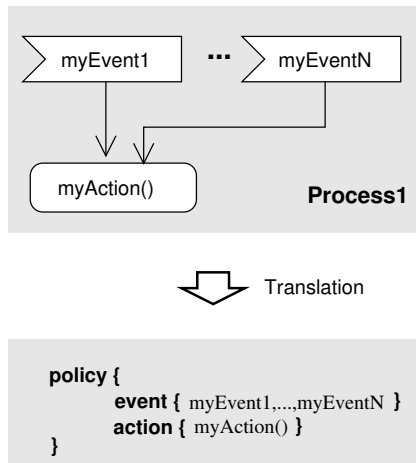


Figure 1.8.: Simple example of pattern translation

cesses. A catalogue of process patterns pertaining to control flow and transitions is presented in Sections 4.5. Given a formal process specification, these patterns are matched within that specification. When a pattern matches a process fragment, policies can be generated from that fragment, as described in Section 4.6.2.

Every match instance yields a number of policies that are generated from the the process partition. To illustrate the principle, Figure 1.8 shows a very simple pattern as an UML activity diagram, and the policy generated from it in a generic policy notation. Several events expected within the process (`myEvent1 . . . myEventN`) trigger an activity within the process (`myAction()`). Provided a policy formalism that supports multiple triggering events for a policy (please refer to Section 3.5 for a survey of policy languages), the single policy shown in the lower box in the diagram could be generated from the pattern instance.

Event-based  
data flow

In general, process activities will expect input and generate output, so that the output of one process activity may be used as input for another. Additionally, activities are performed by, or on account of, roles and affect entities (e.g. infrastructure resources) targeted by the activity. Normally, policies are executed in parallel and independently of other policies' execution. Hence, a policy set representing a process partition (by having been derived from that process partition's formal specification) needs to be provided in a way to receive that process fragment's input and deliver the output accordingly. An elegant solution to achieve this behaviour is achieved by exploiting message-based transport of necessary data. Since most policy implementations rely on events to trigger policies, input data can be transmitted along with the triggering event. In the same manner, output data can be released by having a policy generate an event in addition to executing the specified action. These *rich events* differ from system traps and other events readily generated by the infrastructure, in that they transport aggregated information. This requires a facility that can handle messages carrying payload of different types and sizes. A prerequisite for such a facility are concepts for correlation and aggregation of information items into the rich events needed for process execution.

## Concurrent approaches

At the time of this writing, the community around process-oriented management is quite active, in particular with respect to the design of architectures and tools. As process frameworks attempt to be generic, a certain amount of effort is necessary to adapt OSS tools to local organisational needs (e.g. interfaces to existing tools, management functions, process flow control).



### 1.3. Subproblems and results

The functionality required to implement the procedures within management process activities depends on the specific needs of an organisation and varies accordingly. Given a fast pace of change, architecture and framework designs must exhibit a high degree of flexibility. Newer design ideas such as *Service Oriented Architectures (SOA)* address this issue by incorporating change in their design philosophy. Still, modification of OSS tools' behaviour in response to change in the management processes or administrative procedures incur cost and sources of faults.

Organisations' specifics determine tool functionality

One approach addressing the latter pursues the derivation of tool requirements from best practices collections, specifically the ITIL (see [Bren 07]). Thus, it promises to determine OSS component types for the reference processes regarded. Hence, it focuses on support of processes derived from a specific process framework. Automation of management procedures within such processes is possible to the extent that such procedures are described in the process framework.

Tools can be designed according to best practice collections

An approach targeting process-oriented accounting realised with management policy is found in [Radi 02d, Radi 03]. The flexibility of management policy for the realisation of management processes was leveraged in this work, while accepting the cost of manually deriving operational policy expressions.

### 1.3. Subproblems and results

A number of issues must be resolved in order to map process control and data flow to policy based execution of processes. Prerequisite issues concern process and policy representation, as well as concepts for translation of process-equivalent policies. This section gives a short description to each subproblem together with the conclusions and results that ensue from their treatment in this work.

**Determination of requirements on process specification.** Process specifications are created by persons with domain knowledge. While processes can be described at different levels of abstraction, the most interesting case is that of low-level, operational process specification. Such management processes can be automated if their specification contains sufficient details about the process. The required level of detail is specified in a list of criteria that can be compared to the

Criteria for process specifications

expressiveness of existing process description formalisms or used to extends such formalisms.

Analysis of formalisms

**Determination of required expressive power.** In order to be translated between, management policy and process formalisms must have equivalent expressive power, or have common element subsets with equivalent expressive power. To determine these elements and ascertain semantic equivalence of the elements without limiting the approach to single languages:

- Common process formalisms as well as policy languages/formalisms are analysed and assessed.
- The expressive power of the language elements in both groups is determined.
- Common semantic elements of two groups (process and policy formalisms) are identified.
- The largest possible common subset of semantic elements of both formalism classes allowing applicability of the approach with the largest number of languages is determined.

This element subset lays the foundation for the translation schemes presented in this thesis.

Mapping process partitions to policy rules

**Realisation of process control flow using management policy** While processes are per se procedural, policies constitute rules that are executed in parallel, and with very little execution context. A mapping of process control flow onto a set of policies without loss of information requires

- examination of basic procedural constructs that are executable with little or no execution context,
- specification of process patterns that can be detected in process definitions,
- translation sets for process patterns, allowing automated generation of management policy rules from process parts.

**Realisation of process data flow** As policy execution is easily distributable and parallelisable as it typically does not require to keep

or set state between execution of single rules. When seeking to implement an IT management process by means of management policy rules, however, some of the process's activities will require that process state be kept. In addition, parallel execution must be suppressed in some cases. For this reason, an event-based approach for process data flow is developed in this work that

Event/message-  
based data  
flow

- transports complex data structures, as opposed to simple named events
- couples processes in different administrative domains.

### 1.4. Structure of this work

The greater part of this work is divided into three main parts: Foundation, Elaboration and Proof of Concept. Figure 1.9 shows the conceptual workflow mapped onto the structure of the thesis.

After introducing the issues at hand in this chapter, the FOUNDATION part presents scenarios (Chapter 2) that illustrate the problems by example. Analysis of the scenarios yields a catalogue of requirements (Section 2.4) that are used as a benchmark of discussed work throughout the thesis. This work contacts two different areas of management research, both of which are represented by a vast amount of background work. While some requirements are found addressed in the related work presented in Chapter 3, others are addressed by original work presented in the second part of the thesis. An a-priori, informal summary of the contribution has been given in Section 1.2.

The second part, ELABORATION, contains the in-depth treatment of the core contributions of the thesis. Chapter 4 deals with control flow. It analyses process and policy formalisms, to determine a common ground necessary for the automated derivation of policies from formal process specifications. After outlining a method for translation, a pattern catalogue is presented that constitutes the fundament for the pattern-based process-to-policy translation mechanism. In contrast, Chapter 5 concerns itself with the control flow in policy-based execution of management processes. Based on the concepts developed in these chapters an architecture for their implementation is described in Chapter 6. It includes classic elements of policy based management architectures, as well as the building

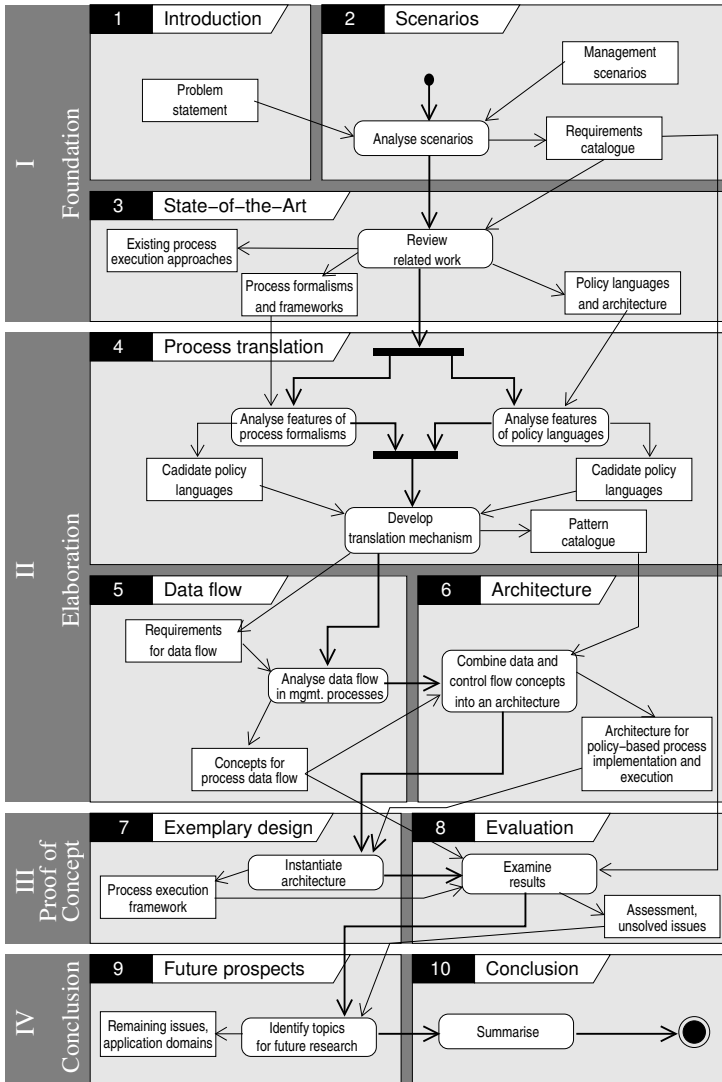


Figure 1.9.: Overview of this work

blocks necessary for the assessment of policy parts as well as the generation of policies.

To validate the approach, the PROOF OF CONCEPT part discusses the prototypical realisation of the concepts developed in the second Part. Chapter 7 instantiates the architecture proposed in Chapter 6, which yields a design for an exemplary organisation of a management system. In this context, the software components developed in this work are discussed. Finally, Chapter 8 examines the applicability of the concepts developed in this thesis against the background of the requirements catalogue given in Section 2.4.

The final part of this work points out topics of future work that are related to this thesis in Chapter 9 before concluding with a summary in Chapter 10.



# Chapter 2

## Scenarios and requirements analysis

**A**N APPROACH in the domain of IT management should be aligned to needs and desiderata in the practitioners' domain. Therefore, the applicability of the approach presented in this thesis is asserted by relating the developed concepts to realistic management scenarios. The scenarios discussed in this chapter are either derived from real-life projects, or constitute common management settings found in large organisations. The scenarios are described and analysed to identify the management challenges presented in Sections 2.1.3 and 2.2.2. The process automation scheme developed in this thesis aims to support IT organisations in coping with such management challenges. To illustrate their manifestation in an operational process, Section 2.3 discusses an exemplary process and highlights the projection of management issues (that were extracted from the scenarios) onto its execution.

Management challenges

The challenges identified in the scenarios result in requirements on the process-to-policy translation procedure developed in this work, as well as on the overall management architecture employed to realise the policy-based execution of processes. The requirements are discussed, weighted and categorised in Section 2.4. They serve as a guideline and a benchmark. In consequence, they are referred to throughout the thesis, and they are revisited in Chapter 8, where the degree of their fulfilment is assessed.

Requirements on the approach

The scenarios are intended to plausibly illustrate the issues arising from the integration of process-oriented management with technical management. They have deliberately been selected to originate from two different settings—commercial ASP and Grid computing—so that management challenges in a broader scope could be gathered for the requirements analysis. The purpose of this selection is to foreclose an all too narrow range of applicability in the translation

Broad scope

and policy-based process execution schemes developed in the second part of this thesis.

## 2.1. Inter-domain application service management

This first scenario describes the situation of a provider of application services, from an IT management point of view. An outline of the provider's operations is given, before focusing on her management infrastructure, roles and interfaces, within and outside the provider's administrative domain. The assessment of these key characteristics allows us to determine the management issues faced by the a provider; they are summarised in Section 2.1.3.

This type of scenario is common, as more and more organisations rely on an *Application Service Provider (ASP)* to host and operate their business applications. This outsourcing strategy allows the organisation in the customer role to focus on its core operations. The burden of operating and managing application services is transferred to the provider, and the cost for application usage is predictable for the customer. An ASP will typically operate different applications for multiple customers.

Consider the situation depicted in Figure 2.1. An ASP hosts a number of business applications for her customers (a single customer is shown exemplarily in the figure). To be able to provide service, the ASP organisation operates and maintains an *IT infrastructure*. A network and a server farm, as well as system and application software are part of this infrastructure.

A customer may access the service by means of a *client application*. In our scenario, this is a common web browser; several versions of popular browser applications have been tested to work with the web-services the customer subscribes to. Customers are allowed to perform simple, predefined management actions (e.g. to allocate resources) themselves, using a *management tool*.

### 2.1.1. Management processes and tools

As customary, the contract between the ASP and her customers includes telephone and email support for the customer organisation's employees. The support is to be claimed by means of a *Single Point*

Customer-side components

Outsourcing of Incident Management and Service Desk



## 2.1. Inter-domain application service management

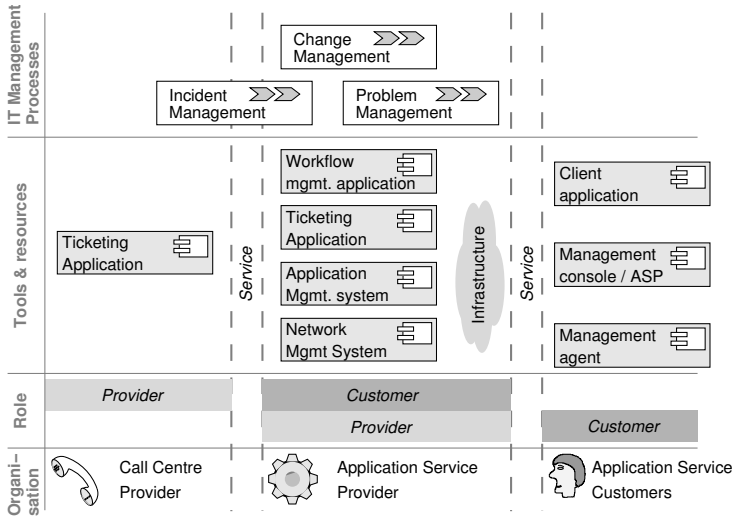


Figure 2.1.: ASP scenario

*Of Contact (SPOC)* provided by the ASP's *IT Service Desk*. However, the ASP lacks the resources to operate a service desk by herself: the number of customers varies, and the number of incidents reported (that require intervention) is seldom uniformly distributed over time. To control cost, the ASP decides to outsource her incident management to an offshore call centre provider that offers application-specific telephone-based support.

The ASP has established Change and Problem Management processes. These processes are implemented, maintained and executed within the ASP's IT organisation. They are concerned with the prevention and solution of problems, and with the controlled selection and execution of changes to the infrastructure (see Section 3.1.1).

The problem management process of the ASP is aimed at reducing the number of incidents, and, hopefully, reducing the fees from the outsourced service desk. Frequently encountered incident types are diagnosed, and a permanent solution is devised, if possible. To track the problems, the problem management staff of the ASP's uses a simple ticketing system.

Problem Management

The ASPs' Change Management process is designed to ensure safe

Change Management

change procedures. In particular, it is important that changes do not incur unplanned service outages. The change management process is supported by a workflow management application that ensures that the change management activities are carried out properly.

To manage applications running on behalf of customers, the ASP's technical staff employs a customised application management utility. In addition, the state and load of network links and elements is monitored and set by means of a network management system. A *management agent* installed on the customer's systems facilitates monitoring of services with regard to availability and quality.

### 2.1.2. Roles, relationships, and interfaces

Roles according  
to the MNM  
Service Model

The relationships between the ASP, her customer and the outsourcer determine, in part, the requirements on IT management. The participants in Figure 2.1 form a value chain that provides an application service to the ASP's customer. Their roles can be classified roughly according to the MNM Service Model [GHKR 01], yielding a two-level nesting of services. The bars in the *Role* part of the figure indicate the roles assigned to a participant. This scenario focuses on the management needs of the ASP, however the remaining two participants define these management needs in part.

**Customer** The customer organisation subscribes to services provided by the ASP. The application service provisioning, as well as its implementation is hidden from the customer; instead she is presented with a *Service Access Point (SAP)* accessible through the client application, and with the management console mentioned in the beginning of this section. Characteristics and quality of the service are negotiated exclusively by ASP and customer<sup>1</sup>. As part of the service, the customer is offered support from the ASP's service desk under the operational stipulation that the service desk shall be the single point of contact for the application service users.

**Call centre outsourcer** The call centre outsourcer processes calls from the ASP's customer. Even though it interacts with members

---

<sup>1</sup>Note that, while the application service customer may provide her own services to her own customer (relying on the ASP), this provisioning is not visible to the other two participants (ASP and CC provider), and therefore irrelevant to the scenario.

## 2.1. Inter-domain application service management

of the APS's customer's organisation, it does so on account of the ASP and charges the latter for the service provided. Incidents and service requests that the service desk cannot handle need to be transferred to the ASP. The outsourcer provides the data in the export format of its ticketing tool, to be imported in the tool-set of the ASP. As the call centre is the SPOC to the ASP's customers, it coordinates the handling of incidents and provides information to the callers. Hence, records of escalated and subsequently resolved incidents must be transferred back to the call centre for final processing (e.g. notification of users).

**Application Service Provider** The ASP is in both service provider and customer roles. It provides service to the customer (provider role), while relying on the service provided by the call centre outsourcer (customer role).

The ASP has two roles with corresponding views

The application services offered to the customer are subject to a certain feature set and quality guarantees. These details are negotiated with the customer and codified into an SLA. The customer is offered a defined point of access to the service itself using a suitable client software, as well as limited management functions with regard to the service, by means of the management console. The ASP includes provisions for having management agents installed in the customer's domain, to facilitate client-side measurement of performance. The ASP assumes responsibility for the management these clients (installation, updates, patches, de-installation etc), but the customer insists on authority regarding such changes. The support function (service desk) outsourced to the call centre provider is included in the contract.

The ASP acts as a customer towards the call centre provider. From the ASP's point of view, the agreements regarding the outsourcing of the service desk function is an *Underpinning Contract (UC)* for the application services it provides. Incidents reported by users of the application service should be handled by the outsourcer where possible; they should be forwarded to the ASP's problem management personnel in all other cases.

In consequence, the ASP organisation needs to establish several *technical management interfaces*...

Required technical management interfaces

- Updates and patches to performance management agents should be applied in an automated manner.

- The outsourcer should be offered a means to report unsolved, frequent and critical incidents to the ASP.
- Incident reports should be echoed to the ASP's technical personnel in order to allow for active improvement of the infrastructure.
- The outsourced service desk will need information pertaining to user accounts, status of application servers, sessions etc. The ASP must provide up-to-date information to the service desk outsourcer in a manner compatible with the outsourcer's ticketing tool.
- Problems reported by the service desk may need to be supported by information gathered from the infrastructure. Hence, the ASP's ticketing tool must integrate with the application and network management suites.
- Changes to the infrastructure are dependent on up-to-date information gathered from the infrastructure. To provide the change workflow with this information, the application management and network management applications need to pass data to the workflow management system.
- Changes to customer-domain agents will be performed in the ASP's change management process. Changes that are not pre-authorised require the customer's consent, given via the management console. To automate this process, the management console system needs to interact with the ASP's workflow application.

... as well as a number of *organisational management interfaces*:

- Changes regarding the management agents in the customer's domain must be authorised by the customer.
- Online reporting and overview of cost with regard to the outsourced service desk function (e.g. number of incidents, time-to-solve) constitute a basis for management decisions with the ASP.
- Possible issues between the ASP's customer and the outsourced service desk should be escalated to the ASP, bypassing the service desk itself (e.g. if phone support is less than satisfactory).

Required  
organisational  
interfaces

### 2.1.3. Challenges

The distribution of roles and the direction of the value chain determine some of the management challenges the ASP's IT organisation must face. The ASP needs to direct her management effort in a manner that takes into account several issues that are inherent to the scenario.

The introduction of IT management processes promises transparent, reproducible management operations and planning. Cost and time savings are expected to justify the introduction of new IT management processes, as well as the operation of existing ones. Process implementation and execution must therefore be cost efficient. The volume of investments necessary for the introduction of process-oriented IT management must be kept low by reusing the existing assets, in particular collaboration tools and IT management utilities. A simple, consistent solution is sought, that contributes towards financial savings with respect to staff training.

Financial  
challenges

Documented processes make visible the interfaces and information flows between different management tasks. The existing tool collection assembled to support these tasks isn't typically 'process-aware'. Common difficulties involve different interfaces (syntactically as well as semantically), which differ between different vendors, as well as between different tools.

Implementation  
challenges

The multitude of management tools operated by the ASP is in part a function of the application managed on behalf of her customers. As with the IT management tools, such application will offer various (if any) management interfaces. The policy-based process execution scheme developed in this work allows easier integration, and thus reuse, of existing tool sets.

An additional management challenge in the case of our ASP is the outsourced service desk function. To provide transparent support to her customers, the ASP needs to encapsulate it, while ensuring proper and quick incident management. This requires not only coupling of processes in the two domains (the ASP's and the call centre provider's), but also a certain degree of integration between their tool sets. Again, the issue of heterogeneity regarding interfaces and data formats poses difficulties.

Another limiting aspect of the inter-domain management is the administrative autonomy of the participating organisations. Simply

Inter-  
organisational  
management  
challenges

speaking, none of the companies will allow another to (co-)manage its infrastructure, or its services. Hence, loosely coupled alternatives must be sought in order to allow the necessary degree of management process integration. Best practices collections like ITIL recommend to insist on the use of the same tools (e.g. a compatible or identical ticketing tool) in outsourcing scenarios. This requirement is, however, not always possible to meet without incurring additional fees. Thus, the ASP needs to couple the outsourced incident management functions with her in-house change and problem management processes.

The agreement with the customer to install and operate management agents in her domain implies a limited co-management concession, but also entails responsibility for the proper and secure operation of such agents. Changes to these agents need to be authorised by the customer; hence, the ASP's change management process must be coupled with the the customer's processes.

The ability to decompose process specifications, as developed in this work, allows the distribution of process fragments to cooperation partners. The approach presupposes that care has been taken to specify the process in sufficient detail, and to provide sufficient information about the exchange of data between process partitions—which, in this case, are distributed across multiple domains. Thus, the mechanism developed in this work provides an incentive to prepare the specifications of management processes carefully, in order to attain the benefits of automation within and across administrative domains.

Alignment of  
the scenario to  
a process  
example

The inter-domain aspect of the scenario is illustrated by the example presented later in this chapter (Section 2.3). The example describes a workflow partition that is used when urgent patches are to be applied to a managed system.

Applied to our ASP scenario, a security issue discovered in the software running at the customer's site is reported to the service desk operated by the call centre provider. The incident must be escalated to the ASP, who in turn needs to provide a suitable solution, and implement it. The actual change is executed on systems in the customer's administrative domain and requires coordination between ASP and customer.

Similar incidents do occur at unpredictable time intervals, and assuming they are granted sufficient attention, the ASP's management

concept should provide for their timely resolution.

The automation mechanism proposed in this work addresses this issue by allowing for a flexible specification of such management procedures to be integrated within the management processes themselves. This carries the benefits of both having specified the automation procedures as an integral part of the process, and having an adequate means to realise them.

## 2.2. Grid management

The Grid is an emerging paradigm for distributed computing, storage and community interaction. In an academic scenario, resources located in different organisations' data centres are made available to world-wide communities of researchers in a—more or less—standardised manner. As Foster and Kesselmann put it, the Grid aims at

“coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organisations” [FKT 01]

In the Grid environment, such a community of users (e.g. researchers) acts as a *Virtual Organisation (VO)* that is defined by its purpose, i.e. its field of research, instead of by the *Real Organisation (RO)* that employs its members. In short, real organisations are offering services running on real infrastructure to members of virtual organisations. The services may be offered cooperatively by several ROs; hence, everything short of the SAP is being virtualised in the Grid.

Below all these layers of virtualisation, the real, physical infrastructure still needs to be managed. Although Grid applications are quickly becoming more important, most sites participating in a Grid must fulfil other, sometimes far more important, functions at the same time. Thus, in addition to the requirements common in the technical management disciplines, Grid management induces new challenges originating in greater part in the virtualisation of resources and organisations.

In the same way that Grid services are provided cooperatively by several domains, management of these services—and in consequence, of the underlying infrastructures—necessitates a *cooperation aspect*. While this manifests in agreements upon specific tools (and versions thereof) when providing usage functionality, the inter-domain

Management in  
Grids requires  
cooperation

management functions require coordinated (high-level) management policy and coordinated procedures.

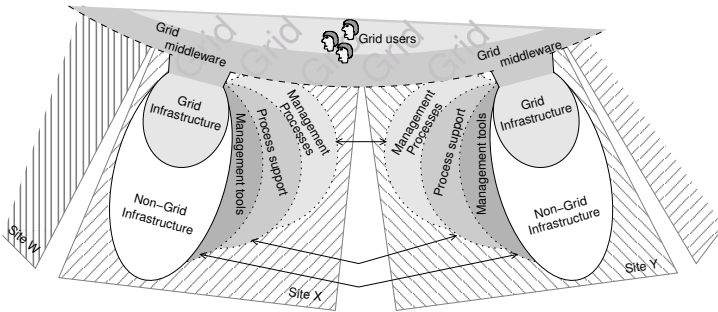


Figure 2.2.: Grid management scenario overview

Services are provided locally and in the Grid

As sketched in Figure 2.2, the Grid itself is constructed by multiple sites, each operating an IT infrastructure. A part of the latter is part of the Grid, while the rest provides services locally, e.g. internet services for a university campus. A site will manage its infrastructure as a whole: while there may be special provisions for the management of the Grid portion, identical management procedures will be applied to the whole infrastructure. For example, management of the core network of an organisation will be governed by a set of procedures and policies applicable to all links and network elements—there will be no division into “Grid-elements” and “non-Grid-elements”. That being said, Grid applications will in most cases generate additional requirements to the management of the infrastructure.

Grid middleware virtualises usage

Even though the infrastructure is far from homogenous across sites, certain aspects of a grid tend to be agreed upon to allow effective operation. An example is the use of grid toolkits (also known as *grid middleware*) such as Globus Toolkit [GT 07], gLite [GLI 07] or Unicore [UNI 07] that hide some of the complexity from *users* of the grid. Note that the grid toolkits do not, however, provide an integrated management of grid resources. From a technology point of view, the preferred communication middleware is Web Services based, and authentication of users is typically performed by means of a *Public Key Infrastructure (PKI)*.

Tools and procedures for management differ between sites

Each site uses its own tool set for systems and network management. They are selected according to requirements originating from



“on-site-only”, as well as Grid management. Naturally, such tools constitute an investment, and operators are trained to use the selection available at their site.

If process-oriented IT management has been implemented within a site/organisation, process support tools are employed. In this case, IT management processes govern the management activities at that site. These processes, in turn, reflect the management needs of a site: they are adapted to the site’s specific features and policies, including financial aspects, personnel aspects, security policy, and site-specific management objectives (e.g. goals regarding QoS). As with technical management tools, the process support applications employed correspond to the (specified) management workflows that need to be supported.

**Example: a Grid computing service**

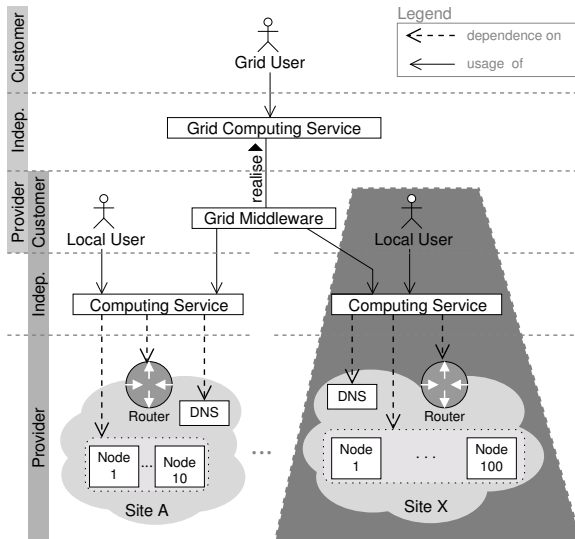


Figure 2.3.: A Grid computing service

To render the management issues of Grid sites more palpable, we consider the following outline of a single service, and the background

for its provisioning.

A computing service is a common Grid application

One well-known application for Grids is the execution of computationally intensive jobs. Ideally, a Grid user will simply submit a job to a broker function of the Grid. The job specifies a program to be run and data to be processed. Application examples include finite element calculation, weather forecasting computations, simulations etc. The Grid middleware will ensure that the job is executed: it will identify a site that agrees to run the program, transfer the input data there if necessary, schedule the job, and secure the results. Thus, the Grid provides a *computing service* to its users.

The instance of the computing service sketched in Figure 2.3 depicts two different sites that participate in offering this service in the Grid. (When a site-specific view is applicable, the scenario is presented from the viewpoint of a site defined as local and placed on a dark trapeze in the figure). Both sites operate computing clusters that offer cycles to their local users, as well as Grid users. To effectively offer the computing service, a number of computing nodes, a network routing function and a naming function must be provided by each operator. The sites agree to participate in providing a Grid computing service, however they retain their administrative sovereignty: *they provide together, but they manage alone*.

Virtual communities each have different requirements

Several VOs require computing cycles from the Grid in which the computing centre participates. Their members come from different research communities, hence the computing jobs submitted to the Grid by members of different VOs may have utterly different properties. In general, Virtual Organisations using the grid will use different resources and services in different ways, according to the purpose of the VO. The requirements arising from this use apply to different degrees in every participating Real Organisations (i.e. providers). As VOs are disbanded and new VOs arise, management requirements change as well.

To maintain a service level that is satisfactory to all customer VOs, the computing service must be provided—and managed—in a corresponding fashion at *all* participating sites. Moreover, changes must be propagated quickly between domains in order to ensure uniform reactions to changes in management requirements.

Effective management is critical for expensive infrastructure

A large, expensive supercomputing infrastructure, operated in addition to the computing nodes, has a short lifespan (compared to acquisition costs), thus rendering any downtime or idle time very

costly, as in the example of DEISA [DEIS 05]. Another issue is the extreme heterogeneity found in general in supercomputing grids: supercomputers are often customised to an institution's needs. The corresponding management facilities are therefore equally heterogeneous.

Management knowledge is found only with experts in the field, which are scarce and expensive. On the other hand, many management tasks are performed by research personnel holding time limited contracts for compelling reasons originating outside the scope of IT management. This leads to a high degree of fluctuation in the occupation of administrator positions. As a consequence of the frequent transitions between managers, management knowledge acquired by the parting administrator is lost and must be acquired anew.

Management  
knowledge  
must be  
conserved

### 2.2.1. Management arrangements

A collection of management facilities and a team of administrators is in place to ensure provisioning of Grid, as well as other, non-Grid services. The management setup attempts to take into account the issues described above.

**IT management processes** To conserve management knowledge, and in order to facilitate repeatable, controlled management measures, IT management processes are defined. A workflow system is employed to provide support for the management workflows at our example Grid site.

The process specifications are site-specific, as they are adapted to the local infrastructure. Thus, they are not transferable between Grid sites, but need customised execution support instead. In addition, processes are in a way "private" to a site; they are not published or made generally available. Interaction between sites therefore requires defined interfaces and data formats.

Management processes should not be constrained to the dedicated Grid infrastructure, but view the infrastructure as a whole. E.g. a Change Management process will be applied to any change required at the site. However, in the case of changes to Grid components, a certain amount of coordination will be necessary in addition.

**Integrated management tools** Cost savings and minimisation of service downtime are paramount in any management setting. Common, off-shelf integrated management systems are employed to support network and systems management. They are complemented by scripts and other tools created and maintained by the administrators of the computing centre.

**Inter-domain collaboration** Collaboration between domains is necessary in any Grid environment, in order to keep the Grid services in good condition. Communication between the administrators of our site, and IT management personnel at other sites is conducted by simple internet email. While agreements are in place regarding the format of some routine messaging, the information still needs to be managed “manually” by the administrators.

**Interaction with Grid and local users** Operation of a service is inseparable from a support function that faces the users of the service. In our example, support must be given to users of local services, as well as users of Grid services. In general, ticketing systems are utilised to collect and track user requests and incident reports. While the use of a set of agreed-upon tools could be enforced within any one Grid site, different sites will in most cases employ different selections of tools. Unfortunately, this obviates interoperability with regard to ticket management within a Grid. In addition, different user VOs may present different requirements with regard to user support.

For some user support functions, SPOCs are already intended, e.g. a central authority for the distribution of digital certificates and keys. However, solutions for many other use cases remain to be defined.

### **2.2.2. Management challenges**

Grids, and sites that participate in Grids are entities that today pose perhaps the greatest management challenges. We will focus on the challenges in the context of management processes. A selection of these is discussed in the following, divided into management challenges pertaining to the local site only, and challenges with regard to several Grid sites.

### Local management

- The IT management processes being introduced require software support. To facilitate interaction between processes, a coupling between process support tools must be realised. This applies to existing tools, as well as to packages introduced as a consequence of the increased emphasis on process-oriented management.
- Existing network and systems management tools must be made interoperable with the process support tools in order to avoid the need for error prone manual intervention, and in order to quicken process execution.
- The coordination of tools must be flexible, in order to facilitate the introduction of additional packages.

The proposition to realise management processes by means of operational management policy offers the opportunity to flexibly interconnect process-driven management activities (e.g. as modelled according to a reference process collection) with the perennial operational tasks. Thus, instead of deploying new, “process-aware” management tools, the proven (and possibly customised) local tool set can be kept in continued use.

**Global management** In the context of the Grid, the scope of management tasks will span multiple sites. The following will illustrate the management issues that, in principle, arise from the collaborative aspect of service provisioning in this multi-site environment.

### Fault and security management

- Problems encountered with Grid applications may be local, or they may be the result of malfunction between domains. In the latter case, problem management needs to be performed collaboratively.
- Grids provide SPOCs for registration/authorisation of users and for user reports. However, effective incident management requires distribution of such reports (e.g. incident records) to relevant sites. To achieve this, interoperation of tools operated centrally with the sites actually solving the incident must be established.

- Reciprocal status monitoring may be implemented in Grid middleware, in order to provide usage functionality. Even though Grids constitute very loosely coupled distributed systems, collaboration in management (e.g. in fail-over cases) will require similar functionality in management interfaces.

Thus, the realisation of management processes governing fault management (e.g. *incident management* or *problem management* processes) is faced with challenges in organisational, as well as technical diversity. The distributable, policy-based realisation of processes, that is developed in this work should offer the perspective to support for automation across participating domains, and to cope with infrastructure and tool set changes—as long as adequate agreements (regarding IT management procedures) between organisations participating in a Grid can be ensured.

Collaborative  
security  
management

Grid services may be targets or collaterals of attacks. To fend off such threats, collaboration between participating sites is paramount. Collaborative security measures, such as a Grid-wide *Intrusion Detection System (IDS)* could be employed to recognise (possibly likewise cooperative, distributed) attacks that threaten the operation of Grid services.

To allow effective collaboration, common security management procedures must be introduced among the operational agreements between Grid sites. The automated translation of a formalised process specification should aid a site in keeping its security-relevant (sub-)processes compliant to the local needs, as well as the requirements of the Grid. This could be achieved by integrating the execution of security measures (auditing, monitoring etc) into the process specifications maintained by the (local) IT organisation. In this context, automated generation of operational management policy can shorten the time and lower the cost of implementing changes to security procedures.

**Configuration and change management** Several standardised software components ensure the functioning of a Grid. Among these are the Grid middleware itself, as well as auxiliary packages commonly expected on standard systems.

- Collaboration in change management is required to maintain matching (or at least compatible) versions of the middleware.

The informal collaboration employed today may be replaced by a more efficient form.

- Global (Grid-wide) notification of local changes is necessary to facilitate tracking of problems across grid sites, in cases when changes at one site induce side-effects.

**Performance and accounting management** Though Grids are used primarily in the academic domain today, accounting of resource use is an interesting topic. Not only would it allow charging based on actual use, but it would also render Grid technology more attractive for corporate operations.

- Since several sites contribute to the execution of a Grid job, collaborative accounting and charging become a necessity. Applied to management software, accounting must be performed across domains, and charging agreements must be constant in a Grid.
- Availability management is a prerequisite for the specification of SLAs for Grid services. Monitoring of service availability in the Grid requires interoperation of monitoring facilities at several, or all, sites.

The adoption of policy-based management techniques to process-oriented accounting, charging and billing is not a new idea. It has been presented in [Radi 02d] and discussed in detail in [Radi 03]. The approach presented in this thesis advances that work by introducing automated generation of policy rules, directly from the process specifications in the area accounting.

### 2.2.3. Summary

While grid toolkits enable the *use* of the pooled resources—effectively creating the grid—a great number of management issues remain. When introducing process-oriented management, our example Grid operator must take into account coupling of management facilities within her organisation, as well as outside of it. Change to processes and tools at her site, as well as at other participating sites, may have an impact on local management arrangements. It is therefore necessary to provide an efficient way to cope with frequent changes. Automation of management procedures will not cross site boundaries,

since every site maintains administrative sovereignty. Coupling of inter-domain management must be designed with this issue in mind.

Both scenarios presented in this chapter hold in common a high-level view on the process-oriented management efforts of the respective IT organisations. Indeed, process specifications constitute a form of high-level management themselves. On the other hand, the translation scheme presented in this work targets process specifications documented in detail. For this reason, the following Section 2.3 introduces a fragment of a management specification that is executed in a simplified environment applicable to both scenarios.

## 2.3. Practical example

The scenarios given in the preceding sections describe management settings as found in typical IT organisations today. This section aims to complement the high-level scenarios by showing an example process in more detail. It is intended to illustrate the challenges arising when the process specification described in Section 2.3.2 is employed in order to govern an aspect of infrastructure management—in this case, the patching of software components—while using commonly encountered types of management tools (described in Section 2.3.4).

Thus, the technical example examined in this section aims to narrow the gap between the management scenarios treated in the foregoing sections (2.1 and 2.2) and the actual process translation scheme addressed in Chapter 4. Then again, due to its structure, the example can be viewed as a third scenario, that focuses on the technical detail in dealing with process-oriented management, instead of paying attention to scale, inter-domain issues and operational complexity.

### 2.3.1. Setting

Consider an IT organisation like the ones presented in the scenarios. It has committed to process-oriented management, and it has defined a number of vital IT management processes in a detailed manner, based on the ITIL. Among these are Incident, Change and Configuration management. The organisation uses tools to support its processes and strives to automate process execution. It also employs a suite of systems and network management applications.

Processes and  
management  
tools



A common case in any organisation operating IT infrastructure is the discovery of security flaws in the deployed application software. In recent time, when important flaws are discovered in a software, its producer or distributor is expected to offer her customers a solution. Such solutions include *patches* that correct the flawed software behaviour, *work-arounds* that offer a way to operate the software without exposing to a certain risk, or new, corrective releases of the software. Typically, periodic notifications of new threats, as well as the solutions available are issued by email. Examples include the bulletins sent by most Linux distributors, as well as the notifications issued by various Computer Emergency Response Teams (CERT). In urgent cases, supplementary notification are issued in order to shorten the time-span during which customers are exposed.

Our example organisation takes security issues seriously and aims to integrate the evaluation of security bulletins, the acquisition of corrective means (e.g. patches), as well as their deployment, into its operational IT management processes. Naturally, this task should be accomplished using the facilities already present in the organisation, and elicit the smallest possible amount of additional effort on the part of the management crew.

### 2.3.2. Example process partition

As a first step, the handling of patches is included in the process specification. A possible result is sketched in Figure 2.4.

1. Once the discovery of a security flaw is advertised, this information is made available to the incident management process. This process is responsible for handling situations that disrupt nominal operation of services (see Section 3.1.1.1).
2. Within the incident management process, an incident record is filed.
3. The affected *Configuration Items (CI)*, i.e. systems, are identified by means of the organisation's *Configuration Management Database (CMDB)*.
4. Based on this information, the incident can be classified according to its severity and the expected impact on the identified CIs. Classification associates a priority, noted in the incident record, with the security loop-hole in question.

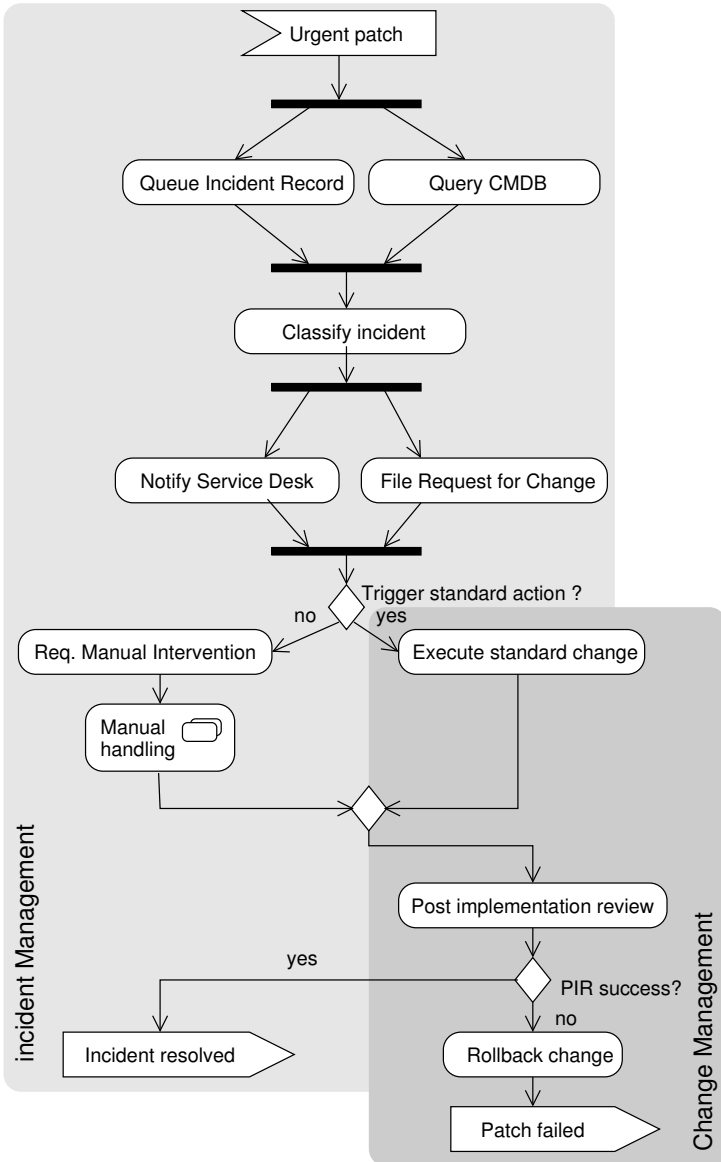


Figure 2.4.: Example: The handling of an urgent security patch

5. The organisation's Service Desk should be notified of the situation in order to be able to respond to customer's queries, if necessary.
6. Any change should be handled by the *change management process*; therefore, a *Request for Change (RFC)*. For recurring, low-impact changes, it is common to define a class of changes as *standard change*, which are pre-authorised require less effort from the change management process.
7. The change is scheduled in a *Forward Schedule of Change (FSC)* according to its priority.
8. At the appropriate point in time, given by the FSC, the patch is actually installed on the group of machines requiring it.
9. A *Post Implementation Review (PIR)* ensures the continuity of service from the machines that have been patched.
10. If the PIR is deemed successful, the incident is marked as solved, and a message is generated to propagate the notification of success.
11. In the contrary case, if the PIR identifies unacceptable flaws, the change is rolled back, and the relevant parties are notified of the failed change.

The deploying of security patches on the organisation's machines spans two different processes. It involves arguably two different administrative domains: that of the software distributor advertising the patch, and that of our example organisation.

#### 2.3.3. Automation of activities

In our setting, several of the activities can be performed automatically, without human supervision. Examples include creating an incident record and placing the record in a queue, notifying the Service Desk, classification according to formal criteria, and the installation of the patch.

Some of the activities require human interaction. The obvious case is the *Manual Intervention* action. If a patch is deemed as critical, human supervision is necessary. Another example is the evaluation of the installed patch within the *PIR* activity. Some aspects of the change can be evaluated in an automated manner, e.g. the success

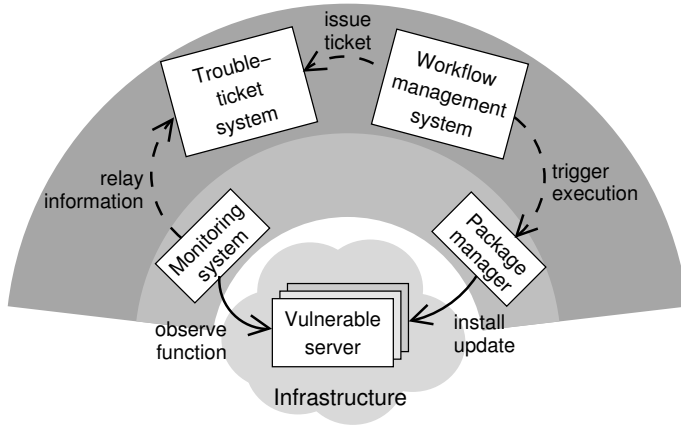


Figure 2.5.: Management tools involved in the example setting.

or failure of the installation itself, the ability of the updated software to run, or the ability of an updated component to interact normally with other components. Other aspects concerning the behaviour of a component after change may be more subtle, and intervention of a human administrator may be necessary to confirm their functioning correctly.

### 2.3.4. Tools

The activities in our example can be supported by appropriate tools. The process activities pertaining to incident management can be supported by a ticketing tool, the deployment and rollback of a patch can be facilitated by a packet management utility, and some of the activities (e.g. the CMDB query) may be supported by in-house tools designed with our example organisation's setup in mind (e.g. interface and schema of its CMDB).

The diagram in Figure 2.5 shows the tool setup in our example. The outer sector contains tools for process support, while the next-inner sector contains the tools for technical management. The managed infrastructure resides at the core of the diagram. The arrows indicate interaction between two components. Solid arrows denote interaction which is easily achievable, as it conforms to the purpose

of a component. The dashed arrows suggest interactions that are heavily dependent on the interfaces and data exchange formats of the interacting components.

### 2.3.5. Challenges

Several issues must be dealt with in order to effectively support execution of our example workflow partition. Interoperability issues arising from different interfaces and data formats in the tools must be solved. In our example, this applies e.g. to interoperation between the monitoring system and the trouble-ticket system (effectively, all dashed arrows in Figure 2.5). The solutions need to be revised every time a product is exchanged or new ones are added. Changes to the process, as well as changes in the tool-set must be handled graciously and in a cost effective manner. The collection of tools for process support and technical management can itself be viewed as a distributed system. Effective process execution, including the delegation of management action to management tools (e.g. the packet manager, in our case) must be feasible even if the components/tools are widely distributed, and some of them reside in other administrative domains.

Heterogeneity  
of interfaces

Dealing with  
change

Distributed  
process  
execution

The above challenges, as well as those ensuing from the two scenarios described in Sections 2.1 and 2.2, impose requirements on the management approach presented in this thesis. Thus, to effectively address these challenges, the policy-based process realisation scheme proposed in this work must satisfy the set of requirements pertaining to the scenarios described in this chapter. This set of requirements is introduced and discussed in the following section.

## 2.4. Requirements

The management challenges identified in the scenarios are a source of requirements on the approach presented in this work. In the context of particularly relevant management challenges, the adequacy of a flexible mechanism for the realisation of processes has already been noted.

In this section, we derive requirements on the solution developed in this thesis based on the scenarios in the previous sections. The

resulting requirements catalogue presented in Section 2.4.1 is employed as a reference for the development of the process-to-policy translation scheme developed in this work (Chapters 4 and 5), and it serves as a framework of criteria for integrating the approach in a management architecture (Chapter 6). The requirements catalogue is revisited in Chapter 8, where it is instrumental to benchmarking the results of this work.

The requirements presented in this section apply in part to the conceptual work presented in the following chapters (e.g. to the method for generating management policy rules from formal process specifications), but also to the actual execution mechanism for (translated) process specifications.

Before describing the requirements in detail, we highlight management challenges that are common to both the ASP and Grid scenarios, in order to derive the requirements categories that apply to this work. To begin with, the pervasive demand of cost reduction and cost control apply, making automation and integration of management procedures a priority. This concerns:

- integration of process support tools among themselves,
- coupling of process support tools with (technical) management tools, and
- integration of (technical) management tools among themselves.

To avoid negating the benefits obtained by a higher degree of integration, the volume of investment (cost) necessary for realising it must evidently be kept low.

Another important challenge is the ability to cope with change in:

- process specification,
- the utilised tool-set, and in
- the infrastructure itself.

The above pre-requisites are flanked by the increasing trend towards collaborative/cross-domain service provisioning that becomes evident from the scenarios. Therefore, the approach should provide means to cope with management across administrative domains.

In summary, the conceptual components of this thesis must take into account the challenges presented in this chapter by means of satisfying the requirements catalogue presented in the following.

### 2.4.1. Requirements catalogue

Being based on scenario management challenges, the requirements enumerated in this section reflect the management necessities sketched above, with regard to the approach presented in this work. They concern the proposition of policy-based execution of management processes, as well as the translation mechanism that allows the generation of operational management policy rules from process specifications.

The requirements are ordered by a number of categories, to reflect the groups of management challenges found in the scenarios. They do, incidentally, include requirements categories of a broader scope. In particular these concern aspects of realisation technology, as well as general expedient requirements applicable to most settings in process-oriented management.

The larger part of the requirements discussed in this section reflect, as might be expected, the demands formulated in the scenarios. They concern the applicability of the approach presented in this work to different services, infrastructures, management facilities and processes, as well as the scalability of the approach in those areas. Further, the requirements catalogue takes into account dynamic aspects, in particular the issue of perpetual change that appeared pervasive in the scenario descriptions—and which today should be expected in virtually every IT operations and management environment. The inter-domain issues noted in the scenarios are likewise addressed by a category of requirements.

#### General requirements

The general requirements given in the following are assumed to apply independently of scenario or setting. They reflect invariant criteria on any realisation of an IT management process.

**1** COMPLIANCE TO PROCESS SPECIFICATION. Process implementation must be correct in that it corresponds to process specification. This is a crucial requirement. If the implementation is not correct, the process will produce unexpected results that in most cases will be harmful to the organisation.

2 DETERMINISTIC RESULTS. The results of process execution should only change when the process itself is changed. In particular, changes in the persons involved (as opposed to the roles defined for the process) should not affect execution.

3 ROBUSTNESS. The breakdown of the processes governing IT operations and planning may have dire consequences. Hence, the requirement of robustness is placed on their realisation. Even though this (robustness) is a popular criterion in any requirements list, it normally refers to robustness of software packages. Here, in contrast, it refers to the execution of process instances.

### **Applicability and scalability**

Constraining a process implementation to apply only to specific types of services, specific infrastructure layers or specific entities would render that implementation incomplete and in need of additional tools. Striving towards a complete management solution, the approach should be applicable to all types of infrastructure components and layers. To support arbitrary processes, the approach must support different entities as well as common grouping mechanisms for entities.

4 APPLICABILITY TO ANY SERVICE. The services provided by an IT organisation vary with regard to the nature and purpose of the IT organisation, as well as over time. For example, the services provided in the two scenarios described in this chapter will be different: in one, valued added services may be offered to the customers of an ASP, in the other, service is provided for the users of a Grid.

To retain its value as a practical management instrument, the approach needs to be applicable independently of the concrete services offered. In particular, the translation mechanism, as well as the policy architecture employed to effectuate process specifications should exhibit no dependencies on the type of service(s) being managed (technologically, e.g. web-based services, as well as regarding the purpose of the service), nor should it be dependent on the operative context in which the service is provided (e.g. a Grid service or a service typical of a telecommunications provider).



5] APPLICABILITY TO ANY INFRASTRUCTURE. The two scenarios obviously describe different infrastructures. In addition, infrastructure changes over time. Heterogeneity of the infrastructure is also a common issue. Thus, the realisation of process specifications must be able to cope with any kind and size of infrastructure. In concrete terms, neither the process translation mechanism, nor the the scheme for process execution/automation may be based on assumptions regarding the infrastructure. Specifically, it would be counterproductive to make assumptions regarding the existence of certain structures (e.g. centralisation of storage facilities) or hardware/software components (e.g. reliance on a family of operating systems). This requirement does not, however, encompass the *management* infrastructure, which is not employed for actual service delivery.

6] APPLICABILITY TO ANY TOOL SET. While common, off-the-shelf management tools are available, the variety of tools employed by one organisation is improbably equal to another's. Therefore, the approach to process realisation should be applicable independently of the tool set encountered in a given IT organisation.

It should be noted, however, that an adequate management tool set is presupposed to be in operation (how would the IT organisation have managed their installation before, if those tools were not available?). Thus, this requirement is not intended to claim that the methods described in this work will completely replace the existing tools of the IT organisation—only to coordinate their work according to a process specification.

7] APPLICABILITY TO ANY PROCESS SPECIFICATION. The nature of an IT organisation's process specifications are impossible to predict, although best practices frameworks can be expected to be consulted to aid their inception. Therefore, an approach targeting their practical realisation must be able to deal with any process specification. It is essential that the specification be formal and syntactically correct.

8] APPLICABILITY TO ALL PARTS OF A PROCESS SPECIFICATION. A process implementation that covers only parts of the defined processes is of less value since it must be complemented with additional

components. This would require costly tool integration and verification of the integrated implementation, even if the parts are verified to comply with process definitions.

**9** **APPLICABILITY TO ANY ORGANISATIONAL STRUCTURE.** Today's organisational structures are increasingly flexible. In addition, the assignment of roles between the IT organisation, its suppliers and its customers may change (e.g. the local IT organisation may buy service from one of the customer-role entities it provides to). Therefore, this work shall make no assumptions regarding the characteristics of the IT organisation. In particular, it should provide mechanisms to cope with process specifications that refer to, and may pertain to, multiple organisation parts. Likewise, the policy-based techniques intended for process execution should be effective regardless of the particular structure of the IT organisation employing them.

**10** **SCALABILITY TO ANY NUMBER OF PROCESS SPECIFICATIONS.** Certainly, the introduction of process-oriented IT management yields the highest benefit to large organisations, where "process knowledge" cannot be concentrated by one person, or by a small group of persons. As the number of process specifications may increase with the size of the organisation, the approach presented in this work is designed to be invariant with respect to the number and size of the processes employed in the IT organisation.

**11** **SCALABILITY TO ANY NUMBER OF PROCESS INSTANCES.** The number of active process instances is independent of the number of process specifications or workflows that are actually defined. To be effective, a realisation of IT management processes should not be encumbered by the number of process instances being executed.

**12** **SCALABILITY TO ANY NUMBER OF SERVICES.** The number of services being provided in the IT organisation varies between organisations, as well as over time (as services may be introduced or retracted). The realisation and execution of management processes must not be encumbered by the number of services being provided, or by the addition or retirement of services. Even with management process specifications that (technically) depend on the number of the number of services provided (examples include tasks in the context

of accounting and charging), the process execution mechanism itself should function independently of the (possibly fluctuating) number of services.

**13** SCALABILITY TO ANY SIZE OF INFRASTRUCTURE. Process orientation is embraced primarily by large organisations managing very large systems. To secure investment and to ensure a long lifespan, process implementation needs to be scalable to large systems even if their growth cannot be predicted at the time of implementation. Ideally, both the translation procedure, as well as the policy-based execution of the management process, as well as the integrated operational management tasks within the process should be independent of infrastructure scale. Note that this requirements applies only to the conceptual approach presented in this work—obviously, management and process support tools, as well as the management architecture that realises process execution need to be scalable to satisfy the needs imposed by the size of the local infrastructure.

### Change

As new technologies become available and customer demands grow, IT management has to cope with frequent changes to infrastructure and services. The following requirements pertain to this aspect.

**14** - CHANGES IN SERVICES AND CONTRACTS. Process implementation must gracefully cope with service change or augmentation. Services offered to customers or used in-house are often enhanced or otherwise changed in response to customer demand or for purposes of optimisation. The process implementation must be adapted quickly to the new requirements, independently of the size of changes in management processes.

Changes to customer contracts often imply different and/or additional requirements to the management of the system. The required changes to process implementation must be performed quickly and correctly (as per contract with the customer).

**15** CHANGES IN INFRASTRUCTURE. Process implementation must gracefully cope with changes to the infrastructure. Growth, reduction or diversification of the infrastructure is common. Adaptation

of processes to reflect these changes need to be implemented quickly and at low cost.

**16** CHANGES IN TOOL SET. Tools may be upgraded, replaced or decommissioned. The realisation of process specifications must be able to adapt or be adapted to such changes in the tool set.

**17** CHANGES IN PROCESS SPECIFICATION. There is no point in implementing something different from process specification. If changes need to be made they should begin at process level and not be executed directly on the implementation. Process implementation refactoring must be quick and cheap.

**18** GRACEFUL RETIREMENT OF PROCESS SPECIFICATION. When no longer necessary, the specification of a process (or a part thereof) may be retired, thus rendering its implementation obsolete. This implementation should be removable from the management system without a negative impact on the overall operations.

### **Inter-domain management**

More and more services are provided jointly by several service providers. To submit all parts of a service to management, management processes need to span across organisational and administrative domains. The following requirements pertain to coupling of processes in different domains.

**19** INTER-DOMAIN PROCESS EXECUTION. Any human interaction required in process coupling generates cost and slows down process execution. Hence, it must be avoided wherever possible. The approach should provide a means to realise process coupling automation. Each domain owner needs to be aware of her responsibilities, the data that needs to be exchanged with processes active in other domains than the local one and the formats and protocols used to communicate with other administrative domains.

[20] RESILIENCE TO CHANGES OF PROCESS SPECIFICATION IN REMOTE DOMAIN. Any organisation should be able to apply changes to their process implementation with justifiable impact on partner organisations. Certainly, adaption of the local processes will be necessary. Nevertheless, it is important that impromptu changes to processes in cooperation partners' domains can be dealt with in a controlled manner.

[21] SCALABILITY TO ANY NUMBER OF INTER-DOMAIN DEPENDENCIES. The number of inter-domain relationships may vary over time (e.g. in Grid environments). Therefore, the approach should not make assumptions regarding the number of inter-dependencies between domains.

[22] SOVEREIGNTY OF DOMAIN OWNER. Domain owners must maintain sovereignty regarding their systems, networks and applications. It is unlikely that providers will agree to let their resources be managed by external subjects. Therefore, it is important to respect their sovereignty when considering process coupling.

### **Economy and reuse**

The cost of management increases steadily due to labour and re-engineering cost. In large scale scenarios, as those described in this chapter, the cost of management can assumed to be high on an absolute scale. The requirements given in the following are aligned around the concepts of economy in management realisation efforts, and reuse of existent solutions within the approach proposed in this thesis.

[23] COST-EFFICIENT PROCESS REALISATION. One of the reasons for employing process orientation is the attempt to save money. Thus, it is important that the realisation of processes is cost efficient, otherwise it might negate some of the benefits acquired by process orientation. As cost in management pertains primarily to labour, the realisation and execution of process specifications must not be labour-intensive.

Routine decisions that need to be made in the course of IT management should be supported by the process implementation. A

mechanism should be provided to formalise a decision and apply it in process instances.

**24** PROCESS EXECUTION WITHOUT EXPERT INVOLVEMENT. Process execution should require little or no expert involvement. For execution of a process to be efficient, routine execution steps should be automated. Supervision should not require domain knowledge regarding the processes.

**25** REUSE OF EXISTING/DEPLOYED PROCESS SUPPORT TOOLS. The approach should enable and support the reuse of the tools available for process support, wherever these functionally satisfy the requirements of the process specification.

**26** REUSE OF EXISTING/DEPLOYED MANAGEMENT TOOLS. The approach should support the continued use of existing management tools, in order to assure the continuity of management procedures at a technical level, and to conserve investments.

**27** AUTOMATION OF INTER-TOOL PROCEDURES. The approach should provide means to automate the interactions between tools, wherever possible, in order to reduce the amount of manual interventions. Where active support cannot be rendered, the approach should at least not limit the endeavour to automate inter-tool procedures.

**28** AUTOMATION OF MANAGEMENT ACTIONS ON THE INFRASTRUCTURE. Management processes may have impact on, or query, infrastructure components. Management actions and queries present in the process specification should be performed in an automated manner.

**29** ACCOMMODATION OF MANUAL PROCEDURES. Manual steps within IT management processes cannot be eliminated completely. The necessary manual steps should be computer supported, where possible. More important, they should be integrated into the overall process. This should be supported by the approach.

**30** IMPLEMENTATION/DEPLOYMENT WITHOUT MANUAL STEPS. Handling detailed process specifications is a labour-intensive task that requires precision and, in many cases, domain-specific knowledge. Therefore, such handling should be constrained to the development and maintenance of the process. The realisation and deployment treated in this approach should be automated.

### Technology independence

The technology employed in different settings varies according to local preferences and needs. In addition, the technological base available to service provisioning and management evolves constantly over time. To maintain its current and future applicability in different domains, the approach presented in this work must be highly independent of concrete technologies employed for its own realisation, the realisation of management tools, as well as the realisation of services.

**31** INDEPENDENCE OF PROCESS FORMALISM. A process formalism can be viewed as a technology for representing processes. The approach should be independent of the peculiarities of specific formalisms. This does not, however, imply that any formalism at all should be usable. Indeed, certain capabilities must be assumed in order to properly represent IT management processes in detail (please refer to Section 4.1 for an elaboration on these capabilities).

**32** INDEPENDENCE OF REFERENCE PROCESS (BEST PRACTICES) FRAMEWORKS. Process design in the context of IT management, most often denoted IT Service Management, frequently relies on reference process definitions such as those provided by the ITIL or the eTOM. The reliance on best practices collections of this kind is not mandatory, however. Therefore, the approach developed in this work should refrain from tying itself too closely to any reference framework or best practices collection.

**33** INDEPENDENCE OF MANAGEMENT INFORMATION REPOSITORY TYPE. IT Management is inherently dependent on accurate information about its management targets. Hence, the collection of management information available to an IT organisation defines (or

rather: limits) its management capabilities. In consequence, a number of approaches have elected to centre around the repositories for management information. While this method may have merit for certain scenario types, it does make assumptions regarding the structure and technology involved in capturing and storing management information. As suggested by the different nature of the scenarios described in this chapter, the approach presented in this work must maintain a high degree of independence.

34 INDEPENDENCE OF MIDDLEWARE AND PROTOCOLS. Communication middleware and management protocols constitute powerful instruments to realise abstraction from concrete infrastructure elements, systems and networks. In consequence, they are used extensively—if not pervasively—in all but the trivial management scenarios. The approach will maintain independence of *specific* middleware and *specific* protocols. It does not, however, purport to renounce their use: the general function these offer will be relied upon and endorsed in several parts of the techniques presented in this work.

### 2.4.2. Weighting of requirements

The requirements listed above carry different importance with regard to the management scenarios that they were derived from. A true, quantitative weighting cannot be attempted, as it would require a very high level of detail in the scenario descriptions. Unfortunately, describing the scenarios in minute detail would commit those descriptions to a narrow scope, thereby compromising their generality. In consequence, the summary given in Table 2.1 includes a coarse, qualitative weighting of the single requirements, using three levels of significance.

Requirements marked with three stars (\*\*\*) constitute a *conditio sine qua non* on this work. Their fulfilment constitutes a critical item for the approach presented herein. Those marked with two stars (\*\*) are important requirements on the approach; they constitute the bulk of the requirements list, and thus they carry a “normal” significance. Finally, the requirements marked with one star (\*) can be said to be the most forgiving ones. They will have only mild negative impact, if left unsatisfied. Nevertheless, they originate in the management scenarios presented, thus their fulfilment is desirable.



### 2.4.3. Discussion

Ultimately, the goal of any concept or facility in the domain of IT management process handling is to render possible the *real execution* of processes. The approach presented in this work addresses this goal from a technical perspective, but in a holistic manner. The issues interposed between the intention to introduce effective and efficient process-oriented management, and the actual realisation of that intention span many different dimensions. They manifest in every phase of the management of processes themselves: design/-modelling, refinement, implementation etc; they are reintroduced through the constant change affecting every aspect of a managed IT organisation; they regard the infrastructure, the tools for managing it, the tools for supporting process; they are even connected to the attitude of the personnel performing the actual management tasks. This challenge is reflected in the majority of the requirements stated in this chapter.

Given the multitude of aspects (some of them named above, others shown in Figure 1.4), the approach developed in this work follows the strategy of being flexible, rather than comprehensive.

**Observations regarding groups with common weight** Requirements with the same weight seem to pertain to common abstract goals. We could argue that the requirements with the highest weight (three stars) ensure that immutable constraints of operations are not violated (e.g., it is not likely that a domain owner will yield control over her own domain when providing a service in a cooperation). Similarly, requirements with middle weight all seem to pertain to aspects of the approach developed in this work (e.g. the requirements regarding applicability or economy/reuse). In contrast, requirements with the lowest weight target mostly technical issues, pertaining to a concrete realisation of the concepts described in this work.

The requirements formulated in this section explicitly demand, or imply a variety of capabilities in existing concepts and technologies pertaining among others to process-oriented as well as policy-based management. In the next chapter, some of the sought characteristics are identified and brought into relation with the approach pursued in this work. For the remainder, solutions are developed in the second Part of this thesis.

<i>Nr.</i>	<i>Requirement title</i>	<i>Weight</i>
1	Compliance to process specification.	***
2	Deterministic results.	***
3	Robustness.	**
4	Applicability to any service.	**
5	Applicability to any infrastructure.	**
6	Applicability to any tool set.	**
7	Applicability to any process specification.	***
8	Applicability to all parts of a process specification.	*
9	Applicability to any organisational structure.	*
10	Scalability to any number of process specifications.	**
11	Scalability to any number of process instances.	**
12	Scalability to any number of services.	**
13	Scalability to any size of infrastructure.	**
14	Changes in services and contracts.	**
15	Changes in infrastructure.	***
16	Changes in tool set.	***
17	Changes in process specification.	***
18	Graceful retirement of process specification.	**
19	Inter-domain process execution.	**
20	Resilience to changes of process specification in remote domain.	*
21	Scalability to any number of inter-domain dependencies.	*
22	Sovereignty of domain owner.	***
23	Cost-efficient process realisation.	***
24	Process execution without expert involvement.	**
25	Reuse of existing/deployed process support tools.	**
26	Reuse of existing/deployed management tools.	**
27	Automation of inter-tool procedures.	*
28	Automation of management actions on the infrastructure.	**
29	Accommodation of manual procedures.	***
30	Implementation/deployment without manual steps.	**
31	Independence of process formalism.	**
32	Independence of reference process (best practices) frameworks.	**
33	Independence of management information repository type.	**
34	Independence of middleware and protocols.	**

Table 2.1.: Weighted summary of requirements on the approach

# Chapter 3

## Related work

THIS chapter surveys related work in the fields of process oriented management and policy based management as well as work on patterns and code generation concepts. The management approach presented in this thesis pursues policy-based automation of the execution of IT management processes. Related work in each of the areas of policy-based and process-oriented management is abundant, even though these two topics constitute virtually unrelated research domains.

Reference process collections are the premier drivers behind the rapid advancement of process-oriented IT service management. The reference process definitions they provide are intended to be used as a starting point for the detailed, customised operational processes treated in this work. The purpose of a reference process definition indicates, to some extent, the potential for automation available to customised process specifications. For this reason, two prominent ITSM frameworks are discussed in Section 3.1.

Reference  
process  
frameworks

To formalise process specifications, a number of languages/formalisms are available. This work relies on a suitable process formalism to be used as an input language for the translation to management policy rules. For this reason, an overview of process languages is given in Section 3.2. Based on this overview, a detailed analysis of process languages, to determine effectively suitable languages, is found in Section 4.1.

Process  
languages

The translation mechanism proposed in this work is pattern-based. Prominent existing work in the domain of process pattern is discussed in Section 3.3. To allow automation, the translation procedure needs to be performed on suitably detailed process specifications. Though a conclusive metric for detail in processes is still missing (*see* Section 9.1.1), the level maturity of processes can be

Process  
patterns

used as a guideline instead. Maturity models are therefore reviewed in Section 3.2.6.

Policy-based management

The proposition, to employ policy-based techniques in order to automate management processes, prompts a discussion of existing concepts and technologies in the domain of policy-based management in Section 3.4, including the difficulties that have hitherto encumbered this management paradigm (e.g. in Sections 3.4.1 and 3.4.2).

Policy languages

As the translation procedure described in this work yields management policy rules, a number of policy languages are reviewed in short in Section 3.5. To select suitable target languages for the translation procedure, a subset of these languages is later examined in detail in Section 4.1.

### 3.1. Reference process frameworks

Processes are introduced in a rapidly growing number of organisations. By documenting tasks and procedures, the high-level goals of an organisation can be put into relation to the actions performed at an operational level. In addition, knowledge with respect to certain tasks is no longer “owned” by individuals; instead it is made available to the organisation as a whole. Thus, training of new or relocated employees can be accelerated. Certain certifications (e.g. ISO 9000, BS 15000/ISO 20000) require documentation of processes and training of employees to execute those processes.

Purpose and benefits

Apart from these obvious benefits of process orientation, organisations can profit from best practice knowledge put forth in process frameworks. In principle, such frameworks are built either bottom up, i.e. from experience, or top down, i.e. by analysis of general requirements and consideration of expert knowledge.

Relevance to this work

The approach proposed in this thesis relies heavily on the continued, high speed introduction and use of IT management processes. Process frameworks are the single most efficient promoters of process-oriented management, as they provide a plausible starting point to the introduction of documented, formalisable processes into the area of IT management. In the following, two process frameworks are discussed: the ITIL, an IT-centric collection of reference processes, and the eTOM, which provides a holistic view on the operations of telecommunications organisations.

#### 3.1.1. IT Infrastructure Library

The IT Infrastructure Library (ITIL) is a collection of best practices published by the UK Office of Government Commerce (OGC). It provides definitions of reference processes that are meant to be customised to user needs. The ITIL is quickly becoming a de-facto standard, as adoption in the industry continues at high speed.

In its second version, the ITIL is organised into several books, each focusing on a specific aspect of process-oriented service management. The best known are the *Service Support* [ITIL 00] and *Service Delivery* [ITIL 01] parts that are discussed in more detail below. The ITIL also contains material to specific aspects of service management, such as software [ITIL 03] and application [ITIL 02b] management, security management, as well as management of the infrastructure [ITIL 02a]. The third version of the ITIL is presently in the making, and it appears that changes to the current, well-known structure are to be expected.

The ITIL reference processes provide different degrees of opportunity for automation and integration with more technical management procedures. In the following, the processes described in the Service Support and Service Delivery books are described in short to highlight their potentially automatable aspects. Obviously, an assessment at the abstraction level of a generic IT service management framework can only serve as a coarse indication of the applicability of the management approach proposed in this work to single process activities. A definitive assessment can only be conducted on customised, organisation-specific process specifications.

##### 3.1.1.1. Service Support processes

ITIL Service Supports treats five operational reference processes. Each process description includes the basic workflow, activity details, process artefacts (e.g. documents, messages), roles and interfaces to other processes. It is thought of as being more technical a process collection than Service Delivery is. This notion is upheld by the larger number of integration and automation opportunities.

ITIL Incident Management describes a process for processing incidents regarding a service and handling customer requests. A vital function is the Service Desk. It serves as a SPOC towards cus-

tomers and coordinates the processing of incidents. The incident management process manages Incident Records (e.g. in the form of trouble tickets). It requires access to the *Configuration Management Database* as well as to various sources of information provided by other processes (e.g. Known Error DB).

Incident  
management  
process  
automation

The incident management process exhibits a high potential for automation, and that automation promises to deliver substantial reduction of cost. Frequently, incident management must handle a large volume of incidents and *service requests* by means of its service desk function. In addition, the personnel pool charged with incident management tasks tends to exhibit a high degree of fluctuation, especially where *virtual service desk* scheme is employed, or a *follow-the-sun*, around-the-clock incident management has been instituted. In addition, important metrics for process execution quality (the *Key Performance Indicator (KPI)* set of the process) is the speed of incident resolution, the number of incidents resolved per time frame, as well as communication with other ITIL processes.

For these reasons, the incident management processes is one of the ITIL processes that are earmarked for automation support early in their introduction, by most of the organisations choosing to deploy ITIL-based management processes. The process carries the same importance in the context of this thesis: it can be viewed as one of the most automation-affine reference processes in the ITIL collection.

Problem  
Management

The Problem Management process organises the identification and resolution of longer-term (when compared to incidents) issues. Reactive problem management focuses on the resolution of issues with great impact on the services, e.g. the correction of problems that generate a large amount of incidents. Proactive problem management seeks solutions to (e.g. structural) problems that may have an impact in the future. Important outputs of the problem management process are *workarounds* to incidents experienced by customers and a collection of *known errors*.

Problem Management is tightly linked to Incident Management. The *reactive* form of Problem Management concerns itself with issues that manifest as incidents. Therefore, an automated, fast coupling between the two processes is desirable. Both processes use common databases, e.g. the Known Error DB and the list of workaround solutions that are made available by the Problem Management process. The management of problems can be performed *pro-actively*, by anticipating issues before they provoke actual incidents. Analysis of

### 3.1. Reference process frameworks

technical management data and statistical methods are employed in this form of Problem Management, giving opportunity for the integration of the corresponding tools into the execution of the process.

The releases of hardware and software employed in an organisation can be managed within a Release Management process. Its activities include the implementation and maintenance of a *Definitive Software Library (DSL)* that centralises the storage and management of deployed (as well as previously deployed) software versions, licences. A *Definitive Hardware Store (DHS)* provides a centralised management of tested and approved hardware, including components and spare parts. The release management process defines a versioning scheme for releasing, taking into account full releases, delta releases (e.g. updates) and package releases. It is responsible for managing releases across intra-organisational boundaries (e.g. in different branch offices), taking into account different needs for software localisation (language-wise) and legal constraints. Deployment of releases must be reversible, which is accounted for in *rollback plans*.

**Release  
Management**

Depending on the level of tools support for the management of software packages and hardware assets, a variety of potential automation and integration hotspots may be leveraged in the context of the Release Management process. Examples include integration of tools for package management and remote, distributed installation into the process, as well as provision for automated rollback procedures.

The Change Management process is instrumental in enforcing controlled change within an IT organisation. All changes must be approved within this process, taking into account their business impact (i.e. the impact on service provisioning). Their implementation is documented and tested in a *Post Implementation Review*. The change management process is triggered by a *Request for Change* originating, for example, in the incident management or problem management processes. It makes heavy use of the CMDB in determining the impact of a change. Planned changes are organised by means of a *Forward Schedule of Change (FSC)*, while urgent *emergency changes* are handled along a quicker path; examples include security-related software updates. It is important to note that the change management process controls and authorises change; it does not perform the actual changes to systems and components.

**Change  
Management**

The Change Management process is perhaps the most complex operational ITIL process, and at the same time a very important one. Beside the interfaces offered to other processes, e.g. for submit-

ting Requests for Change, automation support for the Impact Assessment and Classification activities could yield an effective performance gain. In the same manner, standard changes (i.e. pre-authorised minor changes) could be automated, provided the availability of corresponding tools.

**Configuration Management**

Most management processes and administrative tasks in the IT organisation rely on accurate information about the IT infrastructure. The Configuration Management process is responsible for creating and maintaining the information about every *Configuration Item*. In particular, the CIs themselves must be assigned attributes, and the values of these attributes must be kept up-to-date. The CMDB is the common place where information about CIs is stored. Any change must be reflected in the CMDB in order to keep the information current.

The Configuration Management process is traditionally the most difficult to implement process. This is motivated by the necessity of a current and accurate CMDB. Beside the initial modelling and initialisation issues, any change performed on e.g. services, infrastructure elements, contracts (SLAs) must be recorded in the CMDB. On the other hand, most other processes will query the CMDB in order to retrieve data relevant to their own operation. An integration of discovery and auditing tools within the corresponding activities of a Configuration Management process could help keep the database current and consistent with the actually deployed infrastructure, while automation of responses to external queries can ease the workload of configuration management personnel.

### 3.1.1.2. Service Delivery processes

The ITIL *tactical processes* in the Service Delivery book are considered to be less technical than the operational Service Support processes. They are much less concerned with day-to-day operations. Instead, they focus on planning tasks that benefit the IT organisation in time.

**Service Level Management**

Services are provided to customers at a certain quality level. The service comprises the purely technical part (e.g. a hosted application accessible over a pre-defined interface), as well as non-technical aspects such as the daily time interval during which telephone support is provided by the service desk. It is considered good practice to ne-



### 3.1. Reference process frameworks

gotiate these aspects in detail and to document them in an SLA. The *Service Level Management (SLM)* process handles the contact with the customer's account managers, negotiates SLAs and supervises compliance of the IT organisation to the service levels specified.

The policy-based automation proposed in this work could support the SLM process on the one hand in the task of gathering information, e.g. generated reports and measurements of specific service level values, which often originate in other processes. On the other hand, it offers the opportunity to integrate procedures supported by customer service management tools into the SLM process of a provider.

The availability of a service is often an important criterion for its quality. In most cases, the availability is linked to characteristics of the infrastructure and to the management processes governing it. The availability management process monitors a service, measures its availability (which can be defined differently, per service) and indicates the need for changes.

**Availability  
Management**

The automatable aspects within the Availability Management process centre around monitoring and the evaluation of monitoring data. Monitoring tools, as well as utilities used regularly in order to assess the availability of services can conceivably be actuated from within the process.

An important goal of the Capacity Management process is the dimensioning of the infrastructure in order to hold ready enough capacity for effective service provisioning, while reducing spending on surplus capacity. Capacity management needs to take into account the life-cycle of IT components in order to plan purchase of new and replacement components. According to the ITIL, effective capacity management allows an IT organisation to profit from discounts for bulk purchases and keep its operations within budget.

**Capacity  
Management**

Depending on the size of the IT organisation and on the frequency of change in necessary capacity (e.g. with fluctuating level of use in services), some automation potential can be identified in this process. Leaving aside the scheduling of recurring purchases, the implementation of the Capacity Management workflow can profit from automated requisition of data originating in the Availability and Service Level Management processes. Due to its nature, however, this process is a poor candidate for both automation and the integration with technical IT management: though it attempts to formalise the

adaptation of capacity so that it can be performed by documented procedures, in truth it encapsulates the domain knowledge acquired by experienced IT managers.

**IT Service  
Continuity  
Management**

For many organisations, IT services are a critical component of business operations. Hence, disruption of the IT services for a period of time may have ruinous consequences for the organisation. The objective of the continuity management process is to provide appropriate strategies and means to deal with IT service disruption, e.g. in the aftermath of a natural catastrophe. The measures provided to ensure service continuity must adapt to changes in the services and in the infrastructure. They need to be supervised (e.g. fallback components need to be tested periodically) and maintained.

The greater part of this process is not suited for either automation, nor the integration of technical management tools. However, two areas that constitute exceptions to this rule may benefit greatly from a tighter coupling with technical management: the reaction to changes in the infrastructure (with the aid of information acquired from the Change Management process) and the automatic actuation of fail-over actions when service continuity must be provided in the face of massive failure. While the first ensures that the provisions for service continuity are kept current, the second will save time in critical situations.

**Financial  
Management  
for IT Services**

An important link between business operations and ITIL processes is created by the Financial Management for IT Services reference process. It can be viewed as an accounting process facing the customer, as well as a controlling process facing IT operations.

The best candidates for automation in this process are the charging and reporting activities. Integration of accounting/charging tools into the process is a necessity if service usage is to be accounted at any usable granularity. The automated generation and dispatch of reports can, on the other hand, ease the workload of the financial management personnel. A policy-based approach to accounting management has been presented in [Radi 03].

**Security  
Management**

Increasingly, security aspects become an important part of IT operations. This development is due to the increased reliance on IT services for mission-critical business activities. ITIL's second version provides a collection of best practices regarding security. It is separate from the reference processes given in Service Delivery and Service Support.

### 3.1. Reference process frameworks

Certainly, quick reactions to security incidents are desirable (and in many cases very important). However, security management is a “cross section” process, that concerns every aspect of IT management—whether process-oriented or not. In consequence, automation and integration of tools and procedures in this domain is a special concern that is placed outside the scope of this work.

**The ITIL in current research work** As the ITIL continues to gain popularity, a variety of research projects not affiliated with the OGC are being conducted. The ITIL offer guidelines and reference process definitions but does not provide guidance with regard to the tools that can be used to implement processes, the integration of management processes with an existing management base, or the facilities required to design, model and maintain process specifications at an adequate (for a given IT organisation) level of detail.

The work presented in this thesis carries the assumptions that detail, formal process specifications are available. Thus, while the efforts to formalise and reason about ITIL processes are not in the focus of this work, they are auxiliary to its implementation in any real-life setting.

ITIL processes are described in prose at a quite high level of detail. The reader of ITIL standards literature is deliberately left to fill in the omitted details, and to create models applicable to “her” IT organisation based on the reference models provided. To accelerate this procedure, efforts to provide a formal framework for ITIL processes can be observed presently. In many cases, they are part of tool development projects (e.g. with IDS Scheer’s ARIS toolkit). The availability of a framework of ITIL process definitions sketched at the level of process activities within a modelling tool can be used as a starting point for the creation of customised process specifications at an appropriate level of detail.

Formalisation  
of ITIL  
Processes

Another active area of research in the context of the ITIL is the search for modelling guidelines for the CMDB [BSSG 06]. The structure (e.g. schema) and implementation of a CMDB is highly dependent on the IT organisation where it will be used, as well as dependent on the data repositories already being maintained. In addition, it is of critical importance to many processes specified by the ITIL—in particular those that show the greatest potential for automated execution. Thus, any automation approach, including the one

The CMDB  
issue

discussed in this work, must take into account CMDB structure. As of yet, no conclusive solution has been devised to the CMDB problem: its design still remains an expert domain, and automation attempt must take into account the variable interfaces and information items that may be provided by any given CMDB implementation.

Tool support  
for ITIL  
processes

Presently, a number of applications are being offered to the public, touted to be “ITIL-compliant”, or claiming to adequately support the ITIL processes. The nature of their compliance to the ITIL is however unclear. As the ITIL lacks recommendations regarding tools for process support, an important topic of current research is the determination of requirements on such tools, including capabilities, interfaces and architectural aspects (see e.g. [Bren 06]).

### 3.1.2. Extended Telecom Operations Map

Developed by the Telemanagement Forum (TM Forum), the *Extended Telecom Operations Map (eTOM)* [GB921] is a top-down approach to defining management processes for the telecommunications industry. eTOM specifies four levels of processes, ranging from Level 1 (being the most abstract) down to the operational Level 4. Unlike in ITIL (3.1.1), eTOM processes are defined formally, and eTOM is not constrained to processes for IT management. Instead, it attempts to provide a holistic view of a telecommunication provider’s operations. Moreover, TM Forum has specified a *Service Information/Data (SID)* model [GB922, GB922-0, ?] and an OSS framework, the *New Generation Operations Systems and Software (NGOSS)* harmonised with the eTOM process definitions.

While this would make the eTOM an ideal candidate for deriving customised processes, the concrete process levels (especially Level 4) are still sketchy and incomplete. For the purposes of the approach proposed in this work, the availability of a formal process reference framework, coupled with the definition of interfaces for software tools would greatly ease the creation of detailed process specifications, as well as facilitate the integration of technical management procedures into the process specifications. Considerations regarding the potential for automation and integration of single reference processes can be performed in analogy to the exemplary assessment of the ITIL processes in Section 3.1.1.

## 3.2. Formalisms for process representation

Automated translation of process specifications requires a machine readable format for the processes. Several languages suitable for process definition or description have been devised, most often with business processes in mind. In addition, several formalisms not necessarily related to processes are noted in this section. To capture the purpose of the specification examined, the section is structured by the standardisation body having released the specification to reflect the interdependencies of standardisation bodies and documents sketched in Figure 3.2 [Danc 06].

The examined formalisms have been ordered by the organisation where they originate. A compact overview shown in Figure 3.2 shows a significant number of relationships between standards, and suggest a convergence of the languages in the future. A more in-depth analysis of a selection of process languages in Section 4.1 precedes the selection of suitable source languages for process-to-policy translation.

### 3.2.1. UN/CEFACT and OASIS

The *Organisation for the Advancement of Structured Information Standards (OASIS)* produces standardisation documents related to process specification in collaboration with the *United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT)* under the label of *Electronic Business using eXtensible Markup Language (ebXML)*.

**Business Process Specification Schema** Among the documents jointly released by UN/CEFACT and OASIS is the Business Process Specification Schema [BPS 01] that describes itself to be

*“a standard framework by which business systems may be configured to support execution of business collaborations consisting of business transactions.”* (from the executive summary of [BPS 01])

The specification is focused on document flow between collaborating parties, and complements the efforts of UN/CEFACT in the domain of standardisation for business/trade documents.

**Business Process Execution Language** The *Business Process Execution Language for Web Services* (*WS-BPEL*, *BPEL4WS*) is an XML-based language standardised by OASIS to specify executable business processes for use with *Web Service* (*WS*) architectures. BPEL is enjoying popularity (and adoption) due to the increasing number of services created as generic web services or existing (non web service) applications that are given a WS interface.

BPEL is considered to be an *orchestration language* since it is used to specify a workflow using multiple web services whose interfaces must be defined using the *Web Service Definition Language* (*WSDL*) [CCMW 01]. While BPEL is constrained to the domain of web services, it does allow for process specification at a sufficiently technical level to allow execution of a process (hence “Execution Language”). At the time of this writing, BPEL is available in Version 1.1 [ACD<sup>+</sup> 03], with Version 2 being actively developed.

### 3.2.2. Object Management Group and Business Process Management Initiative

The *Business Process Management Initiative* (*BPMI*), formerly an independent organisation (BPMI.org) has joined the business modelling efforts of the *Object Management Group* (*OMG*) as of June 2005, creating the *Business Modeling & Integration Domain Task Force* (*BEIDTF*).

**Business Process Modeling Language** The *Business Process Modeling Language* (*BPML*) [Arki 02] is an XML-based language published by the BPMI aiming to

“provide[s] an abstract model for expressing business processes and supporting entities.”

Development for BPML seems to have stopped, but the underlying concepts have been incorporated in the *Business Process Modeling Notation* (*BPMN*).

**Business Process Modeling Notation** Released by the BPMI, the BPMN is a graphical language for process representation. It contains graphical elements somewhat resembling *Unified Modeling Language* (*UML*) elements although not explicitly based on UML. What is

worse, BPMN uses different semantics for a number of graphical objects also present in the UML. On the other hand, the notation is quite powerful while at the same time using icon metaphors to make the diagrams more accessible to human readers. A concise source for BPMN notation samples can be found in [Whit 04].

**UML-based process modelling** Due to the popularity of the Unified Modeling Language several promising approaches exist to model business and/or management processes with UML. Most of these make use of the extension mechanisms provided in UML versions 1.x. The latest UML version 2.0 [UML2i, UML2o, UML2s] incorporates *profiles* that provide support for business process modelling. The diagram type predestined for process representation is the *Activity Diagram*.

**The XML Model Interchange** UML models can be represented in textual form by means of the *XML Model Interchange (XMI)*[OMG 02-01-01, UML2d], which is an XML-based representation of UML. An increasing number of recent tools feature at least some support for the XMI. Thus, processes modelled in UML can be exported in a formal textual format.

#### 3.2.3. Workflow Management Coalition

The *Workflow Management Coalition (WfMC)* is an industry standardisation body established in 1993 to establish workflow standards with respect to common terminology, interoperability and connectivity.

**Workflow Reference Model** WfMC's Workflow Reference Model [Holl 95] gives a definition of workflow/process terms and concepts and presents an abstract architecture as well as a process definition meta-model and an overview of functional (API) requirements for handling processes. A white paper [Holl 04] written ten years after the 1995 release of the Workflow Reference Model summarises the development in the field of workflow (certainly with an emphasis on WfMC publications) and reviews the impact of the Model. A number of interesting subjects are addressed, notably a Business Process Management (BPM) Component model is sketched (see Figure 3.1)

and a classification of standards according to phases ( process definition, process execution etc) in process development.

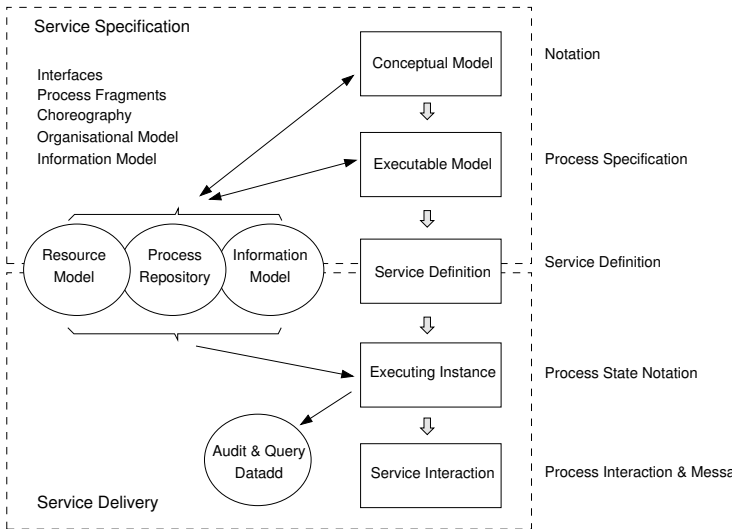


Figure 3.1.: The BPM component model  
The BPM component model [Holl 04]

**XML Process Definition Language** The WfMC released Version 2 of the *XML Process Definition Language (XPDL)* [XPD 05] in late 2005. An XML-based language, XPDL provides a framework for machine readable, textual representations of processes that is compatible with BPMI’s Business Process Modeling Notation. While primarily aimed at workflow system vendors, the XPDL specification is developed in a top-down manner, including meta-models for processes (extended from the Workflow Reference Model) as well as “packages”, which are seen as containers for multiple process definitions together with their meta-data.



### 3.2.4. IDS Scheer

Unlike most other organisations issuing standards for process languages, IDS Scheer is a company with strong ties in academia. The ARIS framework implementation marketed by IDS Scheer spans several aspect domains of business management (to some degree applicable to IT management). Workflows in the ARIS framework are specified using event-driven process chains, which are described in the following.

**Event-driven process chains** The Event-driven Process Chains (EPC) within ARIS [Sche 99, ScTh 05] can be viewed as an established language for process representation. Petri nets by structure, EPCs are directed graphs containing activities, events as well as control flow elements and exploit the idea of event-driven process execution.

ARIS, really an architecture for process oriented management, includes a specification of a graphical representation of EPCs. A more concise description of language concepts and notation can be found in [BKR 03]. Some parts of ARIS, some of them unrelated to process modelling, rely on UML to represent technical information related to classes/objects.

### 3.2.5. Interrelations of process formalisms

Despite their differences in scope and target audience, the process formalisms discussed do relate to each other. The specifications created by a standardisation body often reflect the interest of associated (industry) stakeholders. Thus, interrelationships between formalisms, as well as the relations between the standardisation bodies, may allow conclusions regarding the direction of future standardisation efforts.

The documents reviewed include work of the WfMC, OASIS, UN/CEFACT, OMG, BPMI and IDS Scheer. Figure 3.2 depicts the release of documents originating from these different organisations over time. The horizontal swim-lanes hold documents released by the organisation shown on the left aligned along a time scale. The relations between organisations and documents are marked in the diagram and suggest a certain degree of convergence. On the level of organisations this has manifested in cooperations and mergers, while a higher

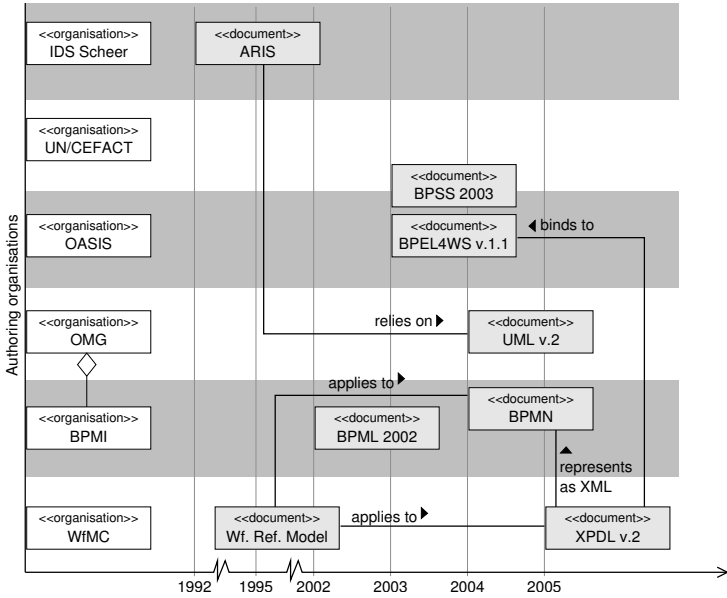


Figure 3.2.: Standardisation in workflow and process management

alignment of the standardisation documents can be seen in recent releases. While only the merger between BPMI and OMG constitutes a strong binding between two of the standardisation organisations, several other cooperations exist (not shown in the figure) that are manifested in common web presences of the cooperation partners.

The relations between documents are also manifold. WfMC's Workflow Reference Model has provided a common understanding of workflows and has been taken into account by the current versions of BPMN and XPD, as well as by former versions. XPD and BPMN are strongly related in that XPD aims to provide a textual, machine readable format for BPMN's graphical representation of processes. BPMN, on the other hand, provides an explicit binding to BPEL4WS by defining a mapping of elements and constructs into BPEL4WS. The ARIS model family relies on UML (since former versions of the UML) in parts, though the event-driven process chains (EPCs) discussed are based on petri net concepts.

The overall picture creates hope of a convergence of organisations as well as of standardisation efforts. Until such a convergence becomes reality, however, it is mandatory that the differences of the different formalisms – regarding expressiveness and language features – are taken into account. An analysis of several formalisms/languages presented in this chapter is found in Section 4.1.3.

#### 3.2.6. Process maturity

A prerequisite for formalised process definitions is the effort invested into their analysis and documentation. A five-stage “roadmap” for process improvement, including their documentation and formalisation, is the *Capability Maturity Model (CMM)* [Univ 95] developed at Carnegie Mellon University. It describes process maturity to start at an “initial” level (see Figure 3.3). At this stage, processes are said to be operative in some working way. The next-higher state, “repeatable” implies that the (undocumented) process can be executed and re-executed in a consistent manner. To achieve the “defined” level, a process needs to be documented. It is considered that most well-engineered processes remain at this level of maturity. Note: The CMM has since evolved into the *Capability Maturity Model Integration (CMMI)*.

Beside the Carnegie Mellon CMMI, several domain specific CMMs

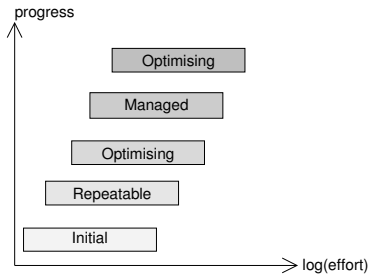


Figure 3.3.: Levels of the Capability Maturity Model

have been developed. In particular, some target IT Management, e.g. the *System Administration Maturity Model (SAMM)* and the *IT Service Capability Maturity Model (ITSCMM)*. In the future, such models may yield metrics for the assessment of process specifications, as suggested in Section 9.1.1.

### 3.2.7. Summary

This section gave an outline of formalisms and standards for process modelling and representation. All textual formalisms use XML for the definition of their grammar, while graphical languages are more diverse, aligning to or extending UML. It should be noted that the sources discussed either have a general scope, as with the UML, or focus on business aspects in particular. Formalisms explicitly supporting management process representation are missing.

## 3.3. Pattern in processes

Patterns have long since been introduced to the realm of software design, e.g. through the use of patterns for object oriented design. In a similar manner, process modelling has been studied to extract fundamental, reusable patterns and process formalisms have been analysed to determine their support of such pattern. An important source of research in this domain is the group around van der Aalst at Eindhoven University. Their research has resulted in a pattern catalogue [AHK+02]. The catalogue is applied to different formalisms to compare their expressiveness from a work-flow perspective.

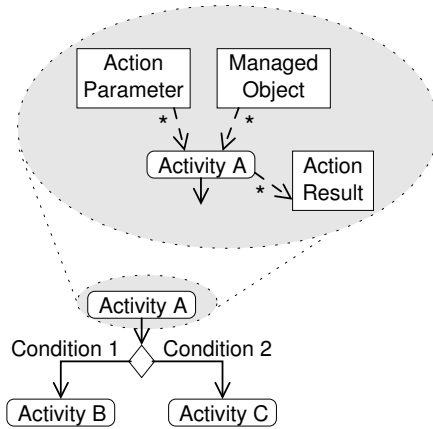


Figure 3.4.: Pattern example: Exclusive choice pattern.

The patterns employed in this approach are abstractions of recurring control flow in work-flows/processes. They concern themselves with the coordination of generic actions, excluding the technical details of actions from view. The pattern collection has been effectively employed for testing the support for a specific kind of process structure in formalisms and architectures/products.

IT management process modelling and implementation must take into consideration language aspects that support the mapping of management process specification onto technical IT management. The patterns developed at Eindhoven University can help by supplying the outer, structural framework for the information associated with an action. Figure 3.4 shows (in its lower part) the “Exclusive Choice” pattern as an UML activity diagram congruent to the one depicted in [Whit]. By design, it does not take into account the inner form of its elements (in this case, actions and conditions).

The approach presented in this work employs patterns in order to effect the translation between a detailed, operational process specification and a policy set. While the intent of the patterns given in Chapter 4 are different, certain structural similarities exist with regard to v.d. Aalst’s pattern catalogues. They, too, describe control flow, albeit focusing strictly on the activities of the process.

### 3.4. Policy Fundamentals

One of the advantages of policy-based management is that it relies on simple concepts, architectures and expressions. This allows for a flexible paradigm without requiring sophisticated architectures to support it. This section reviews important concepts in policy-based management.

Benefits of policy based management

In exchange for this effort PbM promises a flexible management environment, since policies can be altered or exchanged at runtime. Additionally, PbM is a concept that holds for all functional areas of management, is applicable over complete component or service life-cycles and can be employed in any management disciplines. Since policies are envisioned as being more or less isolated entities distributed enforcement is facilitated. At the same time, the set of policies specified for a domain preserve the management knowledge present in that domain, formulated in a single policy language.

Why is it difficult to deploy PbM?

As a research topic, policies continue to gain momentum. Yet, real-life deployment of PbM systems does not reflect this popularity, neither by number nor by size. While a number of management tools purport to be policy based, they employ the PbM paradigm in isolated areas and often cripple it by confining its use to their area of focus. The reasons for this situation can be quickly summarised. Full-blown deployment of PbM in an organisation requires enforcement of a paradigm shift as well as re-specification of management. To make things worse, policy refinement requires semantic mapping between high-level policy expressions and low-level, technical policy. While this promotes formal specification of management goals at different levels, it is costly and requires domain knowledge. Finally, to really leverage the benefits offered by PbM, an all-out solution must be pursued. This may induce the emotional issue of managers no longer 'owning' the knowledge regarding their systems, as well as dispensing with 'legacy' scripts' and similar, time proven management tools.

Policy types

Several types of policy have been described as parts of the paradigm of policy-based management. The single type that has been adopted widely is that of *access control* policy. A closely related type is that of *delegation* policy, which determines how (access) rights and obligations may be imparted to other parties by their original holders. Although it is seldom explicitly named, *selection* policy is an emerging type of policy employed to express, based on a set of available

choices, the mechanism of selecting one of them. Finally, *obligation* policy can be employed to express tasks that need to be performed by a subject entity. This type of policy has received some attention in the domain of privacy management. Also known under the name of *management policy*, it is in the focus of this work.

### 3.4.1. Policy refinement

PbM has a different view on management specification and implementation. One of its core ideas is to issue isolated, high-level rules (policies) that specify systems' behaviour. These high-level policies (also known as strategic policies) are then refined into operational, executable ones.

Policy can be specified at different levels of abstraction, ranging from high-level policy that applies to whole companies or divisions down to technical-level policy that applies to infrastructure elements. This raises the issue of mapping high-level policy down to the detailed, technical policy specification.

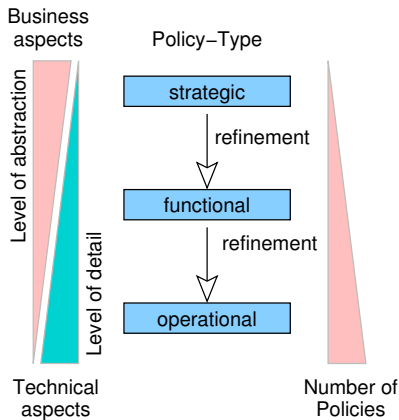


Figure 3.5.: Policy refinement hierarchy

Approaches originating in the policy research community pursue the traditional refinement of policies in a policy hierarchy (see [Wies 95, Koch 96], Figure 3.5) without taking into account management pro-

cesses. Unfortunately, large and complex systems require an accordingly great number of policies to be specified; hence, a lot of effort has to be expended towards the specification of policies.

A formal approach to policy refinement is found in [BLR 03]. It uses event calculus (see e.g. [CCM 94]) to formalise management goals and allow for some automation steps in policy refinement. While being an advanced approach towards policy generation, it requires event calculus based systems modelling as well as manual intervention.

As specialisation of policy implies introduction of additional domain knowledge in each specialisation step. Therefore, the refinement process may not be fully automatable at all.

In the scope of this work, only operational policy, i.e. policy at the most technical level in Figure 3.5 is of relevance. The domain knowledge that must be added manually is extracted from the operational process specifications that serve as the source of process-to-policy translation. This does not mean that refinement has not taken place: it is performed during the the refinement of process specifications (see Figure 1.3).

### 3.4.2. Policy conflicts

Policy conflicts arise when two or more policies contradict each other. The contradiction can occur on the conceptual level, as a conflict of management goals. When several managers or system administrators specify policies for an infrastructure, conflicts of goal can result from different views or opinions of these managers. This is not a problem characteristic of policy-based management; it occurs whenever inherently conflicting decisions are made by administrators. Detection of this type of conflict can yield valuable feedback to the high-level design of management processes (it indicates issues that require agreement among managers). The detection and handling of conflicts of goal is not in the scope of this work.

In some cases, security policy contradicts management policy [LuSI 99]. This is for example the case, when an obligation specified in a policy is contradicted by a security policy that denies the fulfilment of the obligation. In many cases, this subtype of conflict can be detected by analysing the policies alone. For policy-based execution of management processes, the set of necessary authorisations for the

Conflict of goals



actions to be taken could be extracted from the management policy realising the process, as a measure towards conflict prevention.

On a more technical level, policies can conflict in the actions they execute on the infrastructure. Consequences of such conflicts include inconsistencies in systems' state. Conflicts of action can be observed by analysing changes of state in management models prior to policy execution. Kempster proposes the specification of constraints on existing, object-oriented management models in order to detect and correct transitions of a system into invalid states [Kemp 04, KeDa 05]. Bandara et al. propose the modelling of the managed system as well as the policies/actions by means of event calculus to allow reasoning. Taking into account conflict of actions yields feedback to operational process design. In the scope of this work, it could help in the choice of translation patterns when generating policy rules.

Conflict of  
action

### 3.4.3. Architecture for policy-based management

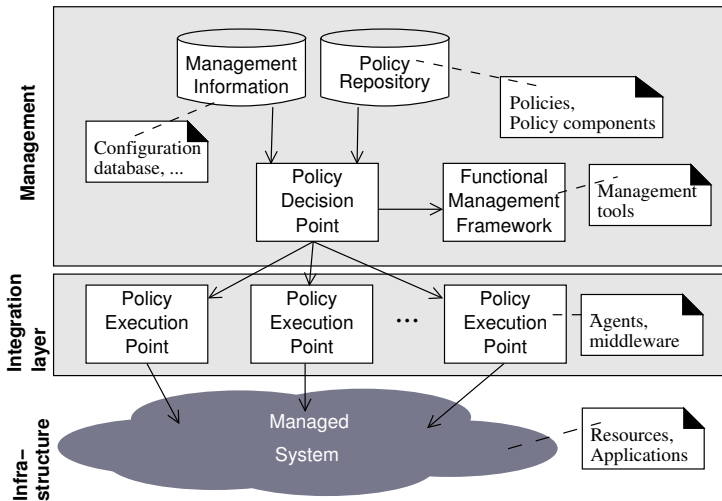


Figure 3.6.: Canonical policy management architecture

When discussing architectures for policy-based management the basic, functional architecture is used as a foundation. It is described

as consisting of three components fulfilling the three core tasks of a PbM architecture: a policy repository that acts as a central store for policies; a *Policy Decision Point (PDP)* that evaluates the policies; and a *Policy Execution Point (PEP)* that enforces a policy (action) on a managed object. Optionally, perhaps as a matter of implementation, a policy management console offers an interface to changing policy, introducing new policies into the system and performing other tasks of managing the policies themselves. It is important to note that this canonical architecture is a functional one; it describes the function of the components and tells little about their realisation. For instance, the PEP may be viewed as a component, where the effect of a policy rule happens. The actual enforcement strategy may include a request for decision by a managed object, or the invocation of an action on the MO by a (central) management entity.

### 3.4.4. Standardisation efforts

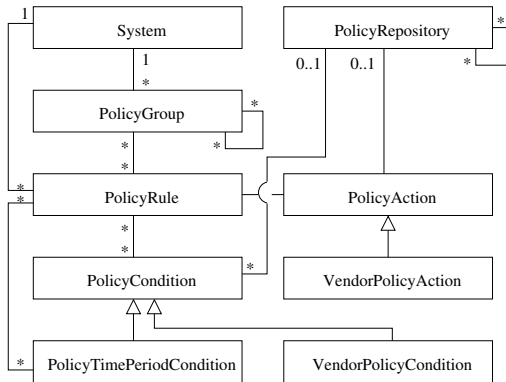


Figure 3.7.: Selected classes in the PCIM

**Policy Core Information Model** In an attempt to formalise the components of policies, the IETF released the *Policy Core Information Model (PCIM)* as a CIM-derived model specialised for policy [RFC 3060]. The PCIM describes an inheritance hierarchy of classes that correspond to policy components. Figure 3.7 gives an overview of the most important PCIM classes. It becomes apparent that a number of important policy components are missing, such as

entities and events. According to [Stra 04], the features of PCIM reflect the clash of interests of stakeholders in the working group - a globally acceptable subset of policy representation is standardised. In consequence, the standardised subset is too small to make a clear statement as to how policy should be described. The PCIM has been adopted as a base of the CIM Core Policy Model [DSP 0108b].

## 3.5. Policy Languages

The specification of policy by means of a formal language aims to influence the behaviour of infrastructural elements – as well as people – according to organisational goals. Several aspects can be differentiated between in this context:

- Behaviour, i.e. decisions to be taken according to a policy are found in different domains and functional areas. Simple examples include the busy domain of access control policy.
- Specification formalisms can differ in scope, syntax and semantics. Some of the policy languages described in this section have been designed for specific areas of use or technologies.
- Policy can be specified at different levels of abstraction. Only the lowest, operational level allows machine supported enforcement of policy.

A central aspect of the approach proposed in this thesis is the expression of low-level management processes by means of policy. The assessment of potential target languages for policy specification is an obvious prerequisite. The following sections introduce a number of candidates for the assessment found in Section 4.1.6.

### 3.5.1. Ponder

One of the well-known policy languages originating in the academic domain is the Ponder language created at Imperial College [DDLS 01]. Unlike most policy languages, Ponder attempts to support a wide range of policy types. For this reason, it defines a hierarchy of policy classes that is extensible to accommodate new policy types, and offers a corresponding extension mechanism in the language itself. In the scope of this work, only Ponder's management policy features are relevant. However, the language supports natively authorisation,

obligation and delegation policies [DDLS 00]. All of these can be specified in two modes. For example, positive and negative authorisation policy can be specified, signifying a permission in the former, and a prohibition in the latter case. The more detailed examination of the language in Section 4.1.6 is constrained to the positive obligation policy (`oblig+`), which is the policy type required by the approach of this work.

Ponder's implementation follows the canonical policy architecture and provides an LDAP-based policy repository, a policy compiler that transforms rules in Ponder into code executable by a PDP, and a management console that allows management of the policies themselves. Agent templates are available, which can be extended to act as PEPs for management policy.

### 3.5.2. Rei

An interesting policy specification language is Rei [KFJ 03], also originating in academia. It is targeted at security/privacy policy in the domain of ubiquitous/pervasive computing.

Rei uses logic constructs (predicates) in order to formulate policies. The Prolog-based language provides a set of predicates that can be used to express policy, and that can be used as a base for reasoning over policy. The predicates that are part of the language are organised in an ontology. A singular feature of Rei is a mechanism called 'speech act'. It allows a form of negotiation between policy system and managed system—in a sense, this is a means to specify simple protocols. The language offers obligation policy among other features. However, Rei lacks an explicit specification of triggers or events; it is unclear (or up to an implementation), how evaluation of a policy is launched. In addition, the predicate framework specified remains at a high level of abstraction (from operational management), and it lacks facilities to express e.g. obligations' actions in sufficient detail. A policy language instrumental to the approach presented in this thesis should support operational management policy; these shortcomings exclude Rei from the set of candidate languages.

### 3.5.3. *PDL*

The specification of the *Policy Definition Language* incorporates most important features of management policy and presents them in a formal manner in [LBN 99]. This language is interesting, since it supports—and focuses on—the policy type necessary for the approach presented in this thesis. The language can be used to formulate management policies at the operational level using simple event-condition-action semantics. Several important mechanisms (e.g. the event model) are developed to be quite powerful. On the other hand, several features that would allow deployment in an IT management environment are missing. Obviously, *PDL* was developed as a study of concepts, rather than as a tool for large-scale management. Experimental results with the language [KoLo 99] have been shown for the domain of network element management. Although the formalism is spartan with regard to its feature set, *PDL*'s language elements are defined formally. A more in-depth examination of the language is found in Section 4.1.6.

### 3.5.4. ProPoliS

The Process-aware Policy System (ProPoliS) incorporates an XML-based language for modular policy specification that focuses on obligation policy at an operational level. It has been designed in the scope of this work, with process-oriented management in mind. Therefore, it incorporates features that facilitate the management of the policies themselves.

ProPoliS is modular in that policy rules can be composed by combining pre-defined policy components. It carries management information in every stand-alone policy component, as well as every policy rule. In particular, a tagging mechanism allows the association of policy components and rules with management processes—a feature that is directly useful to the process-to-policy translation scheme addressed in this work. The language is among the set of suitable target formalisms for the translation developed in this work. A detailed analysis of the language is found in Section 4.1.6, while its implementation is described in Section 7.3.

### 3.5.5. XACML

The *eXtensible Access Control Modeling Language* (*XACML*) [XACML] can be employed to specify privacy and security policy. It is standardised by OASIS. XACML supports a form of obligation policy, however its specification is weak in that it delegates the details of obligation to the implementation, and it applies only to access control. The language has an XML-based syntax; a reference implementation is available.

## 3.6. Summary and appraisalment

A vast amount of valuable background work is available in the areas relevant to this thesis. Even though management approaches comparable to the one developed herein have not been described, scientific contributions as well as standardisation work with respect to process-orientation, as well as in the area of policy-based management provide a base for the mechanisms developed in the following chapters.

Process formalisms and patterns

The formalisms for the representation of processes yield a sufficient number of candidate languages for the more detailed analysis documented in Section 4.1. The existing work on patterns for process control flow provides a parallel for the pattern-based translation mechanism. However, the available pattern catalogues were developed for the assessment of the capabilities of process languages, and they do not take into account the specific requirements of IT management processes.

Process frameworks

From the review of reference process frameworks we can conclude that they constitute the single most important accelerating factor to the advancement of process-oriented management. However, they target broad audiences and are therefore forced to remain in the abstract with regard to process automation and tool support. On the positive side, it is possible to identify the process areas, activities and tasks that seem most suited for automated execution. In addition, the recommendations regarding management components (e.g. the CMDB as an information repository, stipulated by the ITIL) give valuable indications with respect to the classes of tools and components that can be expected in an environment that employs process-oriented management. Likewise, the specification of process

artifacts (e.g. incident records, again in the ITIL) constitute determining factors in the development of the process data flow concepts in Chapter 5.

In analogy to the survey of process formalisms, the available policy languages yield a number of candidates suitable for a detailed assessment conducted in Section 4.1.6. Most significant contributions in the domain of policy and policy-based management seem to originate in academia—standardisation work has hitherto not produced major results in the domain of management policy. This is also the case with respect to architectures and reference models. The well-known three-tier policy architecture (see Figure 3.6 can, nonetheless, be applied to the objectives of the current approach, as is reflected in Chapters 6 and 7.

Policy  
languages and  
architecture

In conclusion, the state-of-the art offers a substantial amount of relevant material to support the process translation mechanism addressed in the next chapter, as well as the realisation of data flow and the development of an architecture suitable for the policy-based implementation of management processes.





# Part II

## **Elaboration**



# Contents – Part II

<b>4. Process translation</b>	<b>95</b>
4.1. Analysis of process and policy formalisms . . . . .	97
4.2. Meta-models of process representation . . . . .	128
4.3. Substitution rules . . . . .	131
4.4. Methodology for translation . . . . .	137
4.5. Fundamental patterns . . . . .	142
4.6. Detection and translation . . . . .	153
4.7. The generating system . . . . .	161
4.8. Extending the pattern catalogue . . . . .	167
4.9. Translation example . . . . .	170
4.10. Summary . . . . .	178
<b>5. Process data flow</b>	<b>181</b>
5.1. Preservation of the information flow . . . . .	182
5.2. Requirements on information transport . . . . .	190
5.3. Realisation of process data flow . . . . .	195
5.4. Summary . . . . .	196
<b>6. Architecture</b>	<b>199</b>
6.1. Management process life-cycle . . . . .	200
6.2. Functional components . . . . .	206
6.3. Interoperation . . . . .	215
6.4. Summary and discussion . . . . .	218

*CONTENTS – PART II*

# Chapter 4

## Process translation

THE approach proposed in this thesis is founded on the idea of automated translation between formal process specifications and operational management policy rules. This chapter presents the conceptual framework for accomplishing this translation.

An important prerequisite for the envisioned translation mechanism is the characterisation and selection of adequate source and target languages. Even though the principles of the pattern-based translation mechanism developed in this work are not dependent on any one specific language, certain criteria regarding the expressiveness of the formalisms involved in the translation must be ensured. The source and target languages are chosen according to two different strategies: while the candidate list of process languages must accommodate the probable choices of process designers, the constraints placed on target policy languages refer primarily to the features relevant for detailed representation of the information contained in process specifications. Candidates for suitable process formalisms are therefore sought among the available, preferably standardised process languages. Thus, the analysis of process formalisms conducted in Section 4.1 primarily takes into account candidates that originate with well-known standardisation bodies, or that have achieved an industry-standard status in time. In contrast, the policy languages examined in Section 4.1.6 originate in academia.

Source and target languages

Both process and policy languages have been evaluated using the same method. They are tested against criteria representing the needs of the translation procedure. The criteria applied to process languages have been formulated with the representation of operational IT management processes in mind, in particular the integration of management tools into process-oriented management; they are summarised in Section 4.1.1. In contrast, policy languages have been assessed with the aid of the criteria set formulated in Section 4.1.5,

Criteria-based evaluation

taking into account among other things that the information contained in the (source) process specifications must be retained after translation to one of the candidate languages. The outcome of the language assessment is positive both in the case of process and in the case of policy languages: several suitable candidates for source and target languages can be selected from, as shown in Sections 4.1.4 and 4.1.6.

Substitution rules	The pattern-based translation scheme described in this chapter can be reduced to transformations of the input process specification by means of substitution rules. In turn, these substitution rules correspond to the transition between two meta-models, described in Section 4.2. The source meta-model that constitutes a common base for the process language candidates assessed in this work. The target meta-model describes a more constrained form for process specifications. The main difference between the two meta-models consists in the cardinalities of associations between process elements (i.e. the nodes of the graph representing the process).The substitution rules, discussed in Section 4.3, convert a process specification from its original form into a form complying to the target meta-model.
Meta-models	
Process patterns	This conversion invariably partitions the process specification, creating process <i>fragments</i> that are matched to a set of patterns, as compiled in the pattern catalogue in Section 4.5. Each pattern is associated with one or more parametrised policy rules. In the event of a match between a process fragment and a pattern, the policy rules corresponding to that pattern are instantiated using the information present in the process fragment's nodes (e.g. the actions specified in the process fragment are used for actions in the resulting policies). The set of policy rules generated in this manner from a process specification constitutes the result of the translation procedure. Together with a suitable execution environment, this set of policy rules allows the automated execution of the original process specification. The translation procedure is illustrated by means of a comprehensive example in Section 4.9.
Translation	
Totality of translation	The patterns presented in Section 4.5 should suffice to translate any syntactically correct process specification provided in one of the suitable source languages. To ensure that a remainder free translation can be achieved in each and every case, the generating system of processes, with respect to translation to management policy, is presented in Section 4.7.

In addition, and in order to allow optimisation, guidelines for extending the existing pattern catalogue are described in Section 4.8.

Extension  
mechanism

This chapter focuses solely on the translation of the control flow component of a process specification. Accommodation of the equally important data flow aspects is addressed subsequently, in Chapter 5.

## 4.1. Analysis of process and policy formalisms

The strong convergence towards best practices process frameworks like the ITIL or the eTOM promise an increase of homogeneity in the way management will be performed in the future. However, it can be expected that organisations will employ different process definition languages, model their processes at different levels of detail and interpret the best practices collections in different ways. Depending on the control structures of each organisation in question, IT guidelines may apply globally (and thus be near homogenous) or they may be defined within its substructures and as a result be similar at best. Additionally, some actors may have reasons to refrain from employing policy-based techniques in the realisations of their processes. In conclusion, it is safe to assume that:

Organisations  
model their  
processes in  
different ways.

- no single formalism for process definition will become a lingua franca of process modelling in the near future,
- the acceptance of a policy-based approach will vary over the organisations' acting in a management scenario,
- no single formalism for policy representation will become ubiquitous,
- no totality in the use of either paradigm will be achieved.

The expressiveness of both source (process) and target (policy) formalisms is of crucial importance. The formalisms' capabilities set bounds on the feasibility and quality of the translation. To avoid constraining the approach to single formalisms, analysis of different source and target formalisms is treated in this chapter. From another viewpoint, the analysis yields the common subset of constructs needed to realise translation.

### 4.1.1. Requirements of IT Management processes

The process formalisms discussed were not designed with IT management processes in mind. They are intended for use in any business process scenario. Although management processes are a type of business processes themselves, they can be defined based on specific assumptions as to the execution environment and the involved personnel. At least in part, the processes introduced into IT management are tightly adherent to the infrastructure providing IT services. Manual subprocesses are executed by persons with knowledge about the process activities (e.g. activities pertaining to service support) that at the same time are (typically) knowledgeable about the function of the OSS tools employed to execute the activities. In consequence, the knowledge “distance” between the process activity (e.g. registering a new configuration item) and the software supporting it (database, application server etc) is relatively small. In contrast, knowledge about a business process (e.g. in automated manufacturing or content management) does not typically imply knowledge about the tools utilised to support it.

This difference can be exploited to achieve a higher degree of automation and integration through adaptation of OSS tools to execute process parts (semi-)automatically by directly interacting with the IT infrastructure. For this purpose, automated and hybrid subprocesses (as described in Section 1) require access to object databases, coupling with monitoring tools as well as integration into facilities for enactment of administrative measures.

In concrete terms, the process automation facility must have access to definitions of entities (persons, components/systems, accounts etc), roles and domains; it must be coupled to the monitoring facilities available, in order to be able to execute process parts in response to events; and, it must be provided a means to influence the managed objects by executing management operations.

To integrate such technical aspects in the process specification, a certain degree of support in the formalism employed for the process definition is required.

### 4.1.2. Assessment of basic language elements

A superficial survey of language elements is sufficient for identifying

Some language elements found in most languages



the ones present in most of the languages assessed. These elements create the basis for a more in-depth assessment and comparison of the languages in question.

### 4.1.2.1. Actions

Due tasks in work-flow or process definitions are often called *activities*. On an operational, technical management level, we refer to due tasks as management *operations* or *actions*. The following requirements refer to actions in the latter sense, i.e. more like function calls against an API than human-executed procedures.

Management goals are realised by executing management actions. Since both processes and obligation policies aim at formalising management goals, all formalisms contain some form of action clause. Actions may take parameters and return values. The return values of actions may be used as input (e.g. parameters) to other language elements.

**Unique name or identifier** To be able to map an action from a process specification to a feature of a process support tool, the process formalism employed must allow specification of unique action identifiers.

**Input or formal parameters** Many examples of (graphically) modelled processes show only the due actions; the input data for the action is neglected or implied from the context. Machine execution of an action requires explicit specification of input data to an action.

**Output or return values** The output of an action, be it a value, a document or a status code signifying success or failure, is often needed as input of another process part. As with action input, explicit support for modelling output data is required.

**Control flow** Though not an intrinsic of actions themselves, language constructs for control flow determine how actions are executed. Common constructs include those for parallel execution (forking and joining execution threads), and conditional branching. Actions may thus be specified as single actions, as sequences of actions or as a

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Action	identification	by name
		by reference/ID
	formal parameters	literal
		action return value
		system/runtime attribute
		other source
		named parameters
	return value	single
		multiple
		complex (object)
	references to objects	in parameters
in return values		
as action target		
Control flow	action groups	sequential
		parallel
		none
	branching	unconditional
		conditional
	error handling	notification
		handling
		keep return “address”

Table 4.1.: Comparison criteria for *action* and *control flow* elements

parallelisable set. In addition, process and workflow formalisms may specify conditional or unconditional branching statements that represent transitions into other parts of the process.

**Error handling** Actions may fail during execution and/or cause further faults in the system executing them. The minimum requirement for error handling is error detection, which suffices to ensure that a process is executed correctly, or not at all. However, this is a quite spartan mode of error handling. Extended requirements appropriate for realistic process support include:

- Notification in case of error. This could mean the notification of an operator if the system is incapable of handling the error condition. Such notification is a criterion applied when more sophisticated error handling (as described below) is missing.
- Error handlers. As with most programming languages, some process formalisms include the concept of error handlers, that are invoked when the normal control flow is interrupted by a fault. The characteristics of such error handlers are dependent on the way the actions themselves are specified. Hence, for the purposes of comparing process formalisms, the mere existence of a error handling concept remains the only criterion.
- Recovery from error handling. Upon successful error handling, it may make sense for the process to continue at the point of interruption, thus requiring a mechanism for storing and retrieving that position in the process specification. As with error handlers, the criterion in our scope is the mere existence of a recovery mechanism.

##### 4.1.2.2. Objects

IT management relies more and more on system and service models based on object oriented modelling frameworks. Examples of such frameworks include DMTF's Common Information Model (CIM, [CIM 05]), Telemanagement Forum's Shared Information/Data Model or models based directly on the UML.

Management actions may hold references to objects in different roles. They may operate on a *target* object, be enforced in the responsibility of a *subject*, take objects as input or deliver them as output.

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Object	grouping	role expressions
		domain expressions
		other groups
	object role	subject
		target
		process data
system object		

Table 4.2.: Comparison criteria for *object* elements

Objects may refer to real-world entities (e.g. persons or systems in their MO representation), groups or abstractions of such entities (roles, domain expressions) or they may denote objects of the management or runtime system.

The modelled infrastructure and the provided services obviously have high relevance to the management processes defined. Therefore, a formalism used for IT management process definition should provide a way to reference these a priori defined object collections. At the very least, a concept of objects in the formalism could be employed to reference tailored “copies” of the (mostly object oriented) management information in the models.

Management objects may relate to several of the expression classes mentioned in this section. For instance, they may be referenced in actions, either as parameters or as targets of an action; Their attributes may be part of conditional expressions; and they may be referenced when creating messages or events. Therefore, the formalisms examined here are checked for the following characteristics:

- Existence of the concept of process-external objects.
- References to objects in action input or output.
- References to objects in conditional expressions.
- Direct references to externally defined objects.

#### 4.1.2.3. Events

When considering a policy-based approach, events are an important language element, as policy system implementation often evaluate

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Event	simple	named
		generic
	compound	contains data
		contains references to data
	operation completion	successful
		failed
	generation	simple
		compound

Table 4.3.: Comparison criteria for *event* elements

policies as a reaction to events. Nevertheless, language elements for expressing incoming (or outgoing) events or signals are not present in all process and policy languages. Event implementation can range from simple signals to more complex message passing mechanisms, e.g. the one presented in Chapter 5. Another important concept in the domain of events is the notification of successful or failed operations; languages may provide mechanisms for decisions based on such notifications.

Management processes are often invoked (or resumed) as a result of events or messages originating from a change of state in a system or from operator input to the process. In addition, such events may transport information to the process. In the same way, messages can be sent from within a process to convey information to human or machine recipients. Some of the formalisms examined rely on message exchange for the greater part of their control flow or for their information exchange. The analysis criteria with regard to messaging support are described in the following.

**Ability to expect messages** To be able to react to messages, facilities to specify expectance of a message need to be included in a formalism. Optionally, the type of the message may be included in the specification, as detailed further on in this section.

**Ability to send messages** Apart from notifications in case of errors, processes may need to send messages about their progress, successfully performed important action etc. The specification for the sending of a message may include more or less detail, such as message

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Condition	relational operators	=, ≠, <, >, ≤, ≥
	logical operators	and, or, not
	arithmetic operators	add, subtract, multiply, divide, exponent

Table 4.4.: Comparison criteria for *condition* elements

type and instructions regarding optionally enclosed information.

**Information transport in messages** As with low-level mechanisms relying on asynchronous messages (e.g. SNMP traps), it does make sense to allow piggy-backing of process internal information in messages.

**Typing and naming of messages** To allow automated reaction to messages, the format of messages must be known or accessible to the tool supporting the process parts dealing with a certain kind of message. Obviously, the format must also be advertised to the recipient. Well-known means of achieving this is unique naming of message types associated with a definition of their content.

#### 4.1.2.4. Conditions

Control flow in processes is often realised by means of conditional expressions. Policy languages often include a condition or constraint clause that determines whether the policy should be enforced or not. The language elements for conditional expressions vary in power. Support can be found for logical expressions and arithmetic expressions, nesting of expressions etc.

Requirements for conditional expressions do not originate solely from IT management process needs. In more technical management processes, however, a powerful conditional expression mechanism can be leveraged to make control flow decisions based on system state or data, thus promoting automation endeavours. For this reason, the formalisms are examined regarding the expressiveness of their conditional expressions, as well as regarding the language elements where these may be included.

**Values** Values can be constants that are part of the process definition, they can be held in attributes of the process or of the runtime system (e.g. process support tools) or they can originate from actions (as return values) or arithmetic expressions. Requirements regarding values are found further on in this section.

**Operators** Conditional expressions require operations that can be relational ( $<$ ,  $\leq$ ,  $=$ ,  $\geq$ ,  $>$ ) to compare values, as well as logical operators to create complex conditions, including *and*, *not*, *or* etc.

**Arithmetic expressions** Common cases of conditional expressions include values obtained by the evaluation of arithmetic expressions in the process definition. Hence, the formalisms are examined for the support of basic arithmetic operators (add, subtract etc.) and their applicability in different contexts.

#### 4.1.2.5. Value types

Any formal language can handle a certain set of data types. These may be numeric, logical, character values, higher-order types (e.g. functions); languages may specify type checking or employ automatic conversion between types. How a language handles values influences the way it is used while the supported data types determine its expressiveness to a high degree.

Business work-flows have a weaker relation to typed data than is common in IT-centric environments. Instead, data is encapsulated in forms or “documents” and often transported in the form of character strings. Independently of its encoding, IT management processes interacting with IT infrastructure could take advantage of clear definitions of data types.

Beside facilitating the design of process-supporting applications, a set of data types is indispensable if automation of process parts is desired. A basic set of data types includes general purpose types for the representation of numeric and textual data as well as special types such as time and date representation. To summarise, process formalisms are examined as to the support for representation of:

- Integer and floating point numbers
- Character strings

- Date/Time expressions
- Boolean values

### 4.1.3. Assessment of process formalisms

At the time of this writing, there are no established languages that are dedicated to IT management process description. This de facto convergence towards business process modelling formalisms suggests that business process modelling languages and techniques are sufficient for IT management process definition. Hence, in the following, several formalisms for business process modelling and definition are analysed according to the criteria given in Tables 4.1 through 4.4. In addition, a number of general formalisms (such as the UML and Petri Nets) are included in the survey.

#### 4.1.3.1. OMG/BPMI's Business Process Modeling Notation and WfMC's XML Process Definition Language

The Workflow Management Coalition's "Process Definition Interface – XML Process Definition Language" (commonly known as XPDL) specifies syntax and semantics of an XML based workflow language available in a final Version 2.0 since October 2005. The Business Process Modeling Notation is a graphical language for process definition. It is similar in expressive power to the XPDL.

This specification combo consisting of a graphical notation and its mapping to XML target process authors from the business domain. For instance, the BPMN defines graphical elements designed to be easily comprehensible by non-technical personnel, e.g. by use of icons representing messages or stereotype actions. Nevertheless, many of the requirements formulated for operational IT management processes are addressed by their specifications.

Because of the equivalence in expression of these languages BPMN is described representatively for both formalisms (with the exception of formalism features where BPMN and XPDL differ). Its specification includes normative text regarding the graphical representation of language elements and the behaviour of compliant modelling tools.



**Actions** BPMN specifies atomic or compound actions (“activities”) that are associated with zero or more **InputSets** providing data to the action as well as zero or more **OutputSets** that represent the action result. Input and output are defined as **Artifacts** that can be of a **DataObject** type and encapsulate documents or parameters made available to the actions.

Control flow can be modelled by means of *gateways* that specify flow forks and joins as well as conditional and event-based branching. BPML specifies different control flow variants, including the **Exception Flow**, that is triggered by an event (i.e. error notification) and constitutes an alternative control flow path that can be merged into the **Normal Flow** by means of the general joining mechanisms provided by the formalism.

**Messaging** The atomic actions (“tasks”) are typed and are optionally associated with incoming and outgoing messages. Special task types denote acceptance and transmission of messages as alternatives to action execution. Events can contain messages that are named and may contain a set of **Properties** consisting of named strings.

**Support for objects** Objects can be referred to via the **Participant** class that encapsulates an entity or role expression identified by a name. A mapping to externally defined objects is not supported explicitly in the BPMN specification, though a facility to reference such objects is present in XPDL.

**Conditional expressions** Conditions can make use of relational and logical operators. They can be associated with flow control elements (gateways).

**Data types** XPDL defines basic data types including all data types required as per Section 4.1.5, and additional data types for representation of externally defined entities as well as *participants* of the process.

Complex data types include records, arrays, unions enumerations and lists.

#### 4.1.3.2. Petri Nets

Petri Nets as such do not specifically address process definition. They are a generic means for modelling “graph-like” structures. The extension mechanisms for Petri Nets (colouring, annotation etc), however, enhance the nets to be a mechanism powerful enough to model business or management processes.

However, the criteria applied to the formalisms are bound to the semantics required for expressing IT management processes. The elements of Petri Nets, being a generic means of modelling flows, are defined in a much broader scope. Extensions are often employed in order to differentiate between nodes in a net, and such an approach could be applied for our purposes as well. In fact, some of the languages/formalisms (or parts thereof) analysed in this sections originate from such extensions. Petri Nets in their pure form are excluded from analysis.

#### 4.1.3.3. Business Process Execution Language for Web Services (WS-BPEL, BPEL4WS)

As its name suggests, BPEL is intended as a formalism for *executable* business processes. In consequence, it addresses those requirements aiming at enabling automation. IT management specific bindings, however, are less prominently developed.

**Actions** BPEL defines actions as invocations of remote API functions, typically in a business partner’s domain. The target domain to be invoked is identified by a `partnerLink` and a `portType` expression. The action to be invoked is identified by its symbolic name. It can receive literal formal parameters and be assigned a variable name to be used as a container for the return value. References to externally defined objects (e.g. MOs) are not supported.

Control flow features include parallelisation of actions and conditional branching by means of `switch` statement. Fault signalling is supported by an exception mechanism and facility for error handling is provided through `compensationHandlers`. The latter consist of alternative actions to be executed when the primary action fails.

**Messaging** BPEL specifies event handlers capable of invoking an operation or instantiating a process in reaction to receiving an event. Conversely, event handlers can be invoked (remotely) to transmit an event. Format guidelines for information transport in events are not provided.

**Support for objects** Externally defined objects are not easily made available to a BPEL process. Similarly, actions are associated with scalar variables for parameters and result.

**Conditional expressions** Conditions are specified for special language elements, such as **switch** constructs or execution thread joining. They can be temporal conditions or expressions formulated in XPath. Relational and arithmetic operators are supported.

**Data types** BPEL relies on common XML data types for user data providing integer, string and date/time types.

#### 4.1.3.4. Unified Modeling Language

The UML has been used for process definition before even though it does not provide specialised means for that purpose. It does, however, provide generic means to express most of the elements noted in Section 4.1.1. In this case, generic means are at the same time a blessing and a curse: while conveying the ability to express all required elements, they make necessary the introduction of conventions regarding accepted ways of expressing elements. This analysis focuses on Activity Diagrams. Process representation options available by use of other UML features and the use of extension mechanisms have not been taken into account.

**Actions** The requirements formulated for management actions are addressed by a group of activity diagram classes. **Activities** model the execution of primitive functions as well as invocation behaviour, transmission of signals and the accessing of object attributes. The requirements regarding functional parameters of actions and return values are satisfied by the **ParameterSet/Parameter** classes that allow the association of input/output objects to actions. Parallel execution of actions as well as conditional branching is provided by

means of the well-known **DecisionNode** (diamond) and **ForkNode** (parallelisation/synchronisation bar) elements. Error handling is modelled explicitly by use of the **ExceptionHandler** class.

**Messaging** Activity diagrams allow the modelling of transmitting and expecting typed signals (events) using specialised actions (**SendSignalAction**, **AcceptEventAction**). Similarly, event payload relaying can be expressed by means of **SendObjectAction**. A means for format definition is not provided explicitly; the type of the objects transmitted can be used to map to the payload format.

**Support for objects** The UML's support for object definitions in the context of actions and events is quite good. References to external object definitions may be facilitated by the fact that such definitions (e.g. deposited in a CIMOM) are mostly based on UML<sup>1</sup>

**Conditional expressions** The preferred way to express conditions in the UML is the Object Constraint Language (OCL). OCL statements can be associated with e.g. **DecisionNodes** in activity diagrams to implement conditional branching. Support for relational, logical as well as arithmetic operations is included.

**Data types** Support for complex data types (classes) is inherently good, while primitive types as those listed in Table 4.2 can be expressed as attributes of classes/objects. A precise primitive type definition (e.g. including value ranges of number types), however, is not provided.

#### 4.1.3.5. Event-driven process chains (ARIS/EPC)

Event-driven Process Chains embrace the concept of processes being event driven. Processes or parts thereof are “invoked” by the reception of an event and can generate events themselves if necessary. EPCs are represented graphically using symbols proprietary to ARIS.

---

<sup>1</sup>Note that although CIM uses a somewhat different meta-model, it is compatible with UML when only access to single object definition is regarded.

EPCs describe process partitions (chains) incorporating actions, conditions, objects and logical connectors for control flow. The language elements are intended for high-level, human-readable representation of processes. To achieve a consistent machine-readable form, a set of conventions regarding the inner structure of the elements addressed would need to be created.

It should be noted that the event-driven process chain formalism is embedded within the ARIS family of modelling techniques that relies among others on entity-relationship modelling for data and UML for class/object modelling. In consequence, some aspects of process modelling are addressed by other formalisms than EPCs. An extension of EPCs termed *enhanced event-driven process chains* allows reference to entities relevant to the process (e.g. organisational entities).

**Actions** Actions in EPCs are textual descriptions of tasks and can be associated with objects, thus allowing the modelling of action input and output. However, support for a formal representation is not provided. Parallelisation and joining of process threads as well as conditional branching is supported by means of logical operators (OR, XOR, AND).

**Messaging vs. EPC events** Events in EPCs serve to invoke a chain (i.e. instantiate a process) as well as to determine a change of process state (e.g. an altered attribute value) or a certain point in time. Although EPCs rely heavily on events, the event concept used is not intended for transmitting messages within or outside processes. EPC events have a declarative nature and their semantics overlap with those of conditions. In other words, an event can consist of a statement (e.g. “data available”) that triggers a process partition if deemed to be true. Readers familiar with petri nets will recognise the origin of the EPC event/condition concept.

**Support for objects** EPC actions can be associated with objects and organisational units. However, a formal representation of these entities is not supported directly.

**Conditional expressions** Conditions and events are described using the same graphical notation. They are not differentiated between explicitly.

**Data types** Data types adhering to the requirements are not defined.

#### 4.1.3.6. ebXML Business Process Specification Schema

The Business Process Specification Schema is a quite high-level business process description language. It has a focus on business document exchange and can be described as a document flow language more than a work-flow language.

**Actions and control flow** There are several definitions of activities/actions in BPSS, however they have quite different semantics compared to the concept of actions as described in Table 4.1. For instance, a **BusinessAction** is a named coarse description of an interaction with a business partner. Its instances are associated with **DocumentEnvelopes**. The BPSS describes a state-machine-like concept (capturing business state) that can be employed for modelling business interactions. The BPSS concept of activities/actions is centred around the transmission of named documents wrapped in document envelopes. To request a (business) action, a document is sent to a receiving party that will transmit an appropriate document in response. The envelope of such a response document contains a boolean attribute signifying if the requested action will be performed (true) or not (false). Parallel control flow is supported. The “BusinessState” of a workflow in progress is kept, thus supporting recovery from error conditions.

**Messaging** Processes can be described as message driven but not as event driven in the IT management sense: messages always originate with business roles (i.e. machine or human actors from different domains) and constitute an exchange of business documents. While event mechanisms employed in IT management (e.g. low-level traps, notifications from management tools) are similar in structure, the messaging concept of BPSS has more in common with documents exchanged by email. A BPSS **BusinessDocument** is a named entity

#### 4.1. Analysis of process and policy formalisms

wrapped in one or more `DocumentEnvelopes`, which contains state information to determine whether the document included is intended as a request or a response. In the latter case, a simple boolean flags a positive/negative response. Additionally, an envelope containing a document may transport one or more attachments related to the document.

**Support for objects, roles and domains** The two roles of *requester* and *responder* are the only roles defined in BPSS. There is no apparent mechanism to refine or complement these roles. Domain expressions are not supported and while objects may be described inside documents, no reference can be made to predefined objects as described in Table 4.2. Document format or structure is not given in the specification.

**Conditional expressions** Collaboration between two roles is governed by pre- and post-conditions. Conditions can be also be imposed on the association of a `BusinessDocument` to a `DocumentEnvelope` in order to determine if the envelope is suitable for the document it wraps. However, there is no formal specification of condition features or syntax.

**Data types** Data types for use in process modelling are not defined. Internally, the language relies on the common XML data types to represent information.

##### 4.1.4. Analysis conclusions

The suitability of the analysed process formalisms with respect to management process modelling is quite varied. This may be attributed to the different intentions, background and scope of their authors and the releasing organisations. It is important to keep in mind, that most of these formalism are specified for general business workflow/process specification. While the IT management processes in the scope of this thesis are similar to business processes for e.g. manufacturing in a bird's-eye view, they do differ in several important ways.

The process formalisms analysed were intended for general business processes.

<i>Category</i>	<i>Element</i>	<i>Criterion</i>	<i>BPEL</i>	<i>BPSS</i>	<i>XPDL</i>	<i>UML</i>	<i>EPC</i>
Actions and control flow	name/ID	existence of	✓	(✓)	✓	✓	×
		named	(✓)	×	✓	✓	(✓)
	formal parameters	typed	×	×	✓	✓	×
		return value	✓	(✓)	✓	✓	(✓)
	reference to MOs	complex value	(✓)	(✓)	✓	✓	(✓)
		support for	×	×	×	✓	(✓)
	as target	support for	×	×	×	✓	(✓)
		parallel execution	✓	✓	✓	✓	✓
	conditional branching	support for	✓	✓	✓	✓	✓
		error notification	✓	✓	(✓)	✓	×
error handler	existence of	✓	×	✓	✓	×	
	recovery from error	×	×	✓	×	×	
Conditional expressions	operators	support for reentry	×	×	✓	×	×
		relational	✓	×	✓	✓	(✓)
	logical	✓	×	✓	✓	(✓)	
	arithmetic	✓	×	✓	✓	×	
	expect	✓	×	✓	✓	×	
Events and messaging	message	send	✓	×	✓	✓	×
		name/ID	×	×	(✓)	✓	×
	typing	format definition	×	×	×	(✓)	×
Types	information transport	support for	✓	×	✓	✓	×
		integer numbers	✓	×	✓	✓	(✓)
	floating point numbers	×	×	✓	✓	×	
	character strings	✓	×	✓	✓	×	
	boolean type	✓	×	✓	✓	×	
Objects	date/time expressions	in process definition	✓	×	✓	✓	×
		object concept	×	×	✓	✓	(✓)
	reference to objects	in action input	×	×	(✓)	✓	(✓)
		in action output	×	×	(✓)	✓	(✓)
	externally def. objects	in conditional expressions	×	×	(✓)	✓	×
	direct ref. to	×	×	✓	(✓)	×	

Table 4.5.: Assessment criteria and Evaluation results



The opportunities for automation are less than for operational IT management processes, and there is a gap between the process support tools and the facilities used for the actual execution of a business process. For example, a business process in the domain of manufacturing and logistics describes activities which to a greater part pertain to the physical realm. The effort required to execute any of them in an automated manner extends to the purchase of specialised machines to actually perform the task. In contrast, activities within IT management processes can trigger administrative actions, if only the workflow tool tracking the process is capable of invoking the correct actions on the targets of those actions. In addition, the personnel modelling and executing a business process are not necessarily proficient in IT matters—which can obviously be claimed in the special case of IT management processes.

Before this background, it is not surprising that existing business process formalisms do not support all requirements of detailed, operational IT management processes. Some formalisms do provide the needed support for operational IT management process modelling. They do so due to their intention to be general-purpose languages or otherwise maintain a very broad scope (e.g. XPD<sub>L</sub>) – in short: explicit support for IT management processes is lacking.

Existing process formalisms need amendments to support operational IT management.

The idea of being able to specify business as well as IT management processes by means of the same formalism promises a tighter integration of the processes of the “core business” with those managing the supporting IT.

#### 4.1.5. Requirements for policy formalisms

The language employed for process representation is selected before processes are formalised. Recommendations can be issued to assist this choice, e.g. with respect to the integration of technical management procedures in the process descriptions (see Section 4.1.4). In the context of process-to-policy translation, the target language can be selected flexibly. Naturally, a number of criteria for target languages are similar to those for source (i.e. process) formalisms.

The criteria employed to assess target policy languages can be divided roughly into two categories. *Hard* criteria need to be fulfilled for the language to allow (reasonably) loss-less translation. *Optional* features of target languages may ease translation, allow translation

to a smaller number of constructs (e.g. smaller number of policies), or otherwise have a beneficial effect on process translation and/or execution. The translation approach developed in this work requires expressions of operational management policy (obligation policy). Therefore, any security-related policy constructs in the analysed formalisms are disregarded.

In this work, the ProPoliS (see Section 7.3 for details) will be employed to illustrate examples. As the language was developed with a certain process-awareness, it allows a compact representation of the process parts that are to be translated.

This does not mean, however, that this is the only policy language that can be used as a target language. On the contrary, it is important that the concepts developed herein are applicable independently of a specific policy formalism. The examination of policy languages conducted in the following takes into account the language elements without which translation is not possible. Any language that offers those elements can be employed as a target language. The number of policies generated may be larger, and some constructs may seem somewhat complicated; nevertheless, the result in terms of executed management actions should be identical.

Basic ECA rules imply means to specify events that may trigger a rule, an action to be executed, and conditions that determine the execution of that action. In addition, *target* expressions can be employed to specify the object affected by the action, and, a concept borrowed from security policy, *subjects* specify the entity responsible for the action.

**Action expressions** Policy actions can be formulated at different levels of abstraction (i.e. more or less system specific actions) and with varying reference to system state. Table 4.6 gives an overview of relevant features of policy actions.

The action(s) specified for a management policy must include, as a minimum, the name of the function (or method) to be invoked. In an object-oriented environment, the name of the object offering the corresponding interface can be specified. A function/method may take constant parameters, or names of attributes bound to runtime attributes (known to the policy runtime system), or names bound to external attributes pertaining to an MO representation, e.g. an attribute of an element accessible over SNMP[CFSD 90],[RFC 3512].

#### 4.1. Analysis of process and policy formalisms

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Action	function call	method name
		object specification
	parameters	constant
		bound runtime attribute
		bound MO attribute
	multiple actions	sequential
		parallel
		selectable
	fault handling	detection
		hook for handling
	event generation	simple
		complex

Table 4.6.: Comparison criteria for policy *action* elements

A policy may be constrained to a single action specification, or offer an option to specify multiple actions. The cases of sequential execution (in the order given), and parallel execution must be differentiated between.

Policy actions may fail. Depending on the action to be executed, failure of an action may have consequences of varying impact. A policy formalism may offer a mechanism for fault detection (and a standard reaction to the fault, e.g. notifying an administrator). A more flexible mechanism would offer hooks for fault handling.

In order to chain policies, it may be necessary to generate events/messages in order to trigger other policy rules. Such generated events can be simple events (identified by name) or complex, parameterised, named data records.

**Event** In general, the evaluation of management policy rules is triggered by an event. The nature of such events can differ in several aspects, summarised in Table 4.7. Typically, the occurrence of events is made available to a management system by means of messages. Such events can be typed (or named), and adhere to a data structure that carries additional information (e.g. the origin of the message). In addition, an information payload can be attached to an event; its structure is dependent on the type of the event.

For some applications, certain patterns of events carry a different,

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Event	simple	named
	complex	structured
		with payload
	pattern support	sequence
		time frame

Table 4.7.: Comparison criteria for policy *event* elements

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Condition	simple expressions	logical operators
	complex expressions	nested, normal forms
		arithmetics operations
	reference to values	runtime
		managed object

Table 4.8.: Comparison criteria for policy *condition* elements

augmented semantics from the event messages viewed separately. Optionally, such patterns may be constrained by a time frame, e.g. taking into account only sequences that complete within a certain time of the first message having been received. Although patterns of arbitrary complexity can be conceived, we constrain this analysis to the support for sequences of events.

**Conditions** Conditional expressions in policies are logical statements that are evaluated when a policy is applied. Beyond the lexical elements that are necessary to construct predicates, the language elements for conditional expressions differ in how complex expressions are supported, as well as in the origin and types of values that can be bound (4.8).

We differentiate between simple logical expressions and more complex, nested expressions. The ability to use arithmetic operations in conditional expressions greatly augments their power, however it does incur additional complexity in the implementation of the language. Policy conditions will typically be formulated over attribute values that can pertain to the runtime system (e.g. the current time, an attribute of the event object that triggered the policy), or to the managed system (e.g. a system attribute that is provided by an SNMP agent).

#### 4.1. Analysis of process and policy formalisms

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Target	object	unique
		role substitution
		addressable
		target for action
Subject	object	unique
		role substitution

Table 4.9.: Comparison criteria for policy meta-data elements

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Value	origin	runtime attribute
		managed object attribute
		function value
Type	type checking	
		simple types
	complex types	integer
		floating point
		string
		boolean
		time
		arrays
		objects
		documents

Table 4.10.: Comparison criteria for variables and types in policy expressions

**Subject and Target** Policies refer to managed entities (persons, systems etc). These may be affected by policy execution (thus being a *target* of the rule) or they may be specified as responsible for an activity (*subject*). With regard to management policy, the subject role is less important than in a security policy context.

**Variables, binding and types** Primarily, values may occur in policy expressions as parameters to an action, or they may be part of conditional expressions. The criteria given in Table 4.10 should not be surprising. Support for a number of basic data types is of interest. Temporal expressions may appear often, therefore they are included in the set of 'interesting' types. Support for more complex data types and data structures is also of importance. They

<i>Element</i>	<i>Aspect</i>	<i>Options</i>
Meta-data	Id	name/id specifier
	status	enabled/disabled
	origin	author
		timestamps
		change log
	comment	policy-level
		element-level
		policy-group-level
	reuse	any element type

Table 4.11.: Comparison criteria for policy *meta-data* elements

include arrays, objects (remote or local) and objects that represent documents.

**Meta-data** Policy rules specified for a managed system can become numerous. To easily manage large numbers of rules, meta-data (as listed in Table 4.11) can be added to each policy expression. In the case of generated policies, this feature becomes more important still: without it, the origin and purpose of a policy rule could not be determined.

Unique identifiers for policies (e.g. symbolic names) are a prerequisite for the management *of* policies. Also, the origin of a policy rule (written by an administrator, generated from a tool etc) as well as a change log facilitate the use of policies, in the same manner as log entries for configuration files.

The language elements for meta-data do not enhance (or hamper) the function of the policy-system itself, however. Therefore, by simple, local conventions, the meta-data items could be placed in comment lines or similar constructs. Finally, constructs that allow reuse of element definitions (e.g. objects or actions) may decrease the number of rules.

#### 4.1.6. Assessment of policy formalisms

A number of formal policy languages have been analysed according to the criteria described in the previous section. The languages considered are cross-section of available policy formalisms, having

in common the ability to express operational management policy, or obligation policy.

The rationale of this selection is that, even though obligation policy is applied to security management, it specifies actions that need to be taken under certain circumstances (given by a condition and/or event specification). This mechanism could be exploited for process implementation, if sufficient power of expression is provided by the lexical elements for obligations. However, most languages with a strict focus on security and privacy policy exhibit a narrow understanding of obligation policy, tailored to their application domain. Effectively, this obviates their application in the scope of this work. Detailed analysis results are omitted with respect to these languages; a summary of relevant features is provided instead.

The compliance to the criteria given in the previous section is discussed for the remaining, most promising candidates:  $\mathcal{PDL}$ , Ponder and ProPoliS. Table 4.12 gives an overview of the examination results. A fulfilled criterion is indicated by  $\checkmark$ , while  $\times$  indicates its failure. Partial fulfilment is indicated by  $(\checkmark)$ . Following, the specific reasons for excluding some of the languages from the candidate set are stated.

##### 4.1.6.1. $\mathcal{PDL}$

This early language was designed with management policy in mind. While it fails the criteria for some of the more sophisticated features, it provides a solid base of language constructs that are useful towards process implementation.

**Actions and error handling** The actions of  $\mathcal{PDL}$  policies are functions of arbitrary arity that take *terms* as arguments. Such terms can be constants/literals, return values of functions, or attributes associated with the triggering events. In theory, invocation of functions can address remote objects, or objects in a runtime system. There is no explicit reference to objects hosting such functions (as methods) in the specification. In fact,  $\mathcal{PDL}$ 's specification does not take *any* objects (runtime, remote or other) into account.

The language lacks error detection and control. This is a shortcoming that is difficult to address by extension, since it requires modification to the definition of actions in the language. Monitoring

<i>Element</i>	<i>Aspect</i>	<i>Options</i>	PDU	Ponder	ProPolis
Action	function call	method name	✓	✓	✓
		object specification	×	✓	✓
	parameters	constant	✓	✓	✓
		bound runtime attribute	✓	✓	✓
		bound MO attribute	(✓)	✓	✓
	multiple actions	sequential	×	✓	✓
		parallel	×	✓	×
		selectable	×	✓	×
	fault handling	detection	×	✓	✓
		hook for handling	×	✓	✓
event generation	simple	×	×	✓	
	complex	×	×	(×)	
Event	simple	named	✓	✓	✓
	complex	structured	(✓)	✓	✓
		with payload	×	×	✓
	pattern support	sequence	✓	✓	×
time frame		✓	(✓)	×	
Condition	simple expressions	logical operators	✓	✓	✓
	complex expressions	nested, normal forms	✓	✓	✓
		arithmetics operations	✓	✓	✓
	reference to values	runtime	✓	✓	✓
managed object		(✓)	✓	✓	
Target	object	unique	×	✓	✓
		role substitution	×	✓	✓
		addressable	×	✓	✓
		target for action	×	✓	✓
Subject	object	unique	×	✓	✓
		role substitution	×	✓	✓
Value	origin	runtime attribute	✓	✓	✓
		managed object attribute	(✓)	✓	✓
		function value	✓	✓	✓
Type	type	checking	✓	✓	✓
	simple types	integer	✓	✓	✓
		floating point	✓	✓	✓
		string	✓	✓	✓
		boolean	×	✓	✓
		time	×	(✓)	✓
	complex types	arrays	×	(✓)	✓
		objects	×	✓	✓
documents		(×)	(×)	×	
Meta-data	Id	name/id specifier	×	✓	✓
	status	enabled/disabled	×	×	✓
	origin	author	×	×	✓
		timestamps	×	×	✓
		change log	×	×	×
	comment	policy-level	×	(✓)	✓
		element-level	×	(✓)	✓
		policy-group-level	×	(✓)	✓
reuse	any element type	×	✓	✓	

Table 4.12.: Results of policy language analysis



of actions' success or failure could be performed at the point of their execution; event sequences could be exploited to ensure the execution of corrective actions. However, a policy set generated to implement such error control would hardly represent the intent behind a management process. If support of error handling is mandatory, a different language must be chosen instead.

Policies in  $\mathcal{PDL}$  contain one action. While the language specification does not explicitly forbid multiple actions, constructs to specify parallel or sequential execution of more than one action are missing.

The language does not support the generation of events as such. An appropriate API could, however, given the definition of actions, satisfy the generation of events in order to implement IT management process data flow.

**Events and conditions** Events are viewed as the manifestation of state changes in a system, e.g. through the transmission of trap messages if using SNMP. The language specification differentiates between these events, and so called *policy defined events* which can be compounds of events over time.  $\mathcal{PDL}$  provides sophisticated concepts for event sequences, groups etc. The language specifies flexible time frames called *epochs* for the occurrence of events.

Events are identified by symbols, i.e. named, and they may be associated with attributes; possibly, this is enough to support the transport of structured information between the evaluation of two policies.

Conditions in  $\mathcal{PDL}$  can be formulated as a sequence of predicates constructed from relations between two terms (terms are defined as the ones for function arguments). The common relational operators are supported, while arithmetic operations are realised as functions, again, in accordance to the definition of *terms*. For the same reason, values referenced in conditions include attributes of the runtime environment and, possibly, of managed objects.

**Entities and values**  $\mathcal{PDL}$  appears to focus on element management. The managed object in the scope of a given policy set is therefore implied. There are no mechanisms to specify targets or subjects of a policy. However, there is nothing to prevent the symbols that identify policy actions to specify a managed object (as a target)

in addition to the function name. Obviously, support for common grouping mechanisms such as domains is missing.

The language is typed. Integer, floating point, string and character data are said to be supported; it is unclear if these are the only data types. Complex data types, e.g. arrays, objects, and documents are not supported.

**Meta-data** The language lacks constructs which could carry meta-data for policies. Not even a comment facility is specified. This is, however, in the spirit of  $\mathcal{PDL}$ 's formal, "concepts-only" approach.

#### 4.1.6.2. Ponder

Ponder is perhaps the most versatile language of those examined here. While it is evident that it, too, focuses on security policy, the concepts and mechanisms included for management policy are well-developed. This analysis takes only into account those mechanisms, as the facilities for security policy are not in the scope of this work.

**Actions and error handling** Ponder provides actions and exceptions. A policy rule may hold multiple actions that can be specified to be executed sequentially or in parallel. The mode of execution is determined by means of operators provided in the language for this specific purpose. Actions clauses are written in the style of OO languages; however, the specification is somewhat unclear on how they are actually executed on a target. (Note: An extension to the Ponder1 toolkit that allows execution of obligation/management policy is realised by means of a Java-based agent called via the Java *Remote Method Invocation* (*RMI*).)

**Events and conditions** Ponder supports a similar range of event definitions as  $\mathcal{PDL}$ . In particular, it can define event sequences, as well as single events (or groups of alternative events) that may trigger evaluation of a policy. Conditional expressions can be formulated fairly freely and can include variables bound in the policy definition.

**References to entities** In general, entities in Ponder are represented by domain expressions. In turn, these assume access to a central

directory. In fact, the Ponder toolkit relies on an LDAP directory to store policies, as well as information about managed objects.

**Meta-data** The Ponder specification does not specify comment constructs explicitly. However, most examples given by its authors include comments (preceded by double slashes). In practice, these comments can be exploited to encode meta-data, without the need of extensions to the language or its implementation.

**Element reuse** Ponder is described to be an object-oriented language. This applies primarily to the policy types being ordered in an inheritance tree. More important, named policy types can be defined that act as templates for policy instances. Careful use of this mechanism can help reduce the number of policies necessary for the realisation of an IT management process.

##### 4.1.6.3. Propolis

The ProPoliS language has been designed with a focus on management policy, with application to process-oriented management in mind. Therefore, it is scarcely surprising that it fits the required criteria fairly well.

**Actions and error handling** Propolis supports two kinds of actions: primary management actions, and actions to be performed if the primary actions fail (errorAction). Each of these may describe the invocation of a function or method on an MO, or the transmission of an event. In practice, the latter is often the case with errorActions. Actions may have an arbitrary number of named and typed parameters. Parameter values may be literals, runtime or remote object attributes (including event objects), or return values of functions offered by the runtime system or by MOs. Events transmitted as a consequence of policy execution are named and may carry additional information, obtained as with parameter values.

**Events and conditions** Any execution of Propolis policies is triggered by an event. Events are expected to be named and may contain structured additional information. In addition, a non-structured payload may be attached to an event object.

**References to entities** Propolis can reference single entities, as well as roles and domains that are resolved (lazily) to entity sets. All of these are referenced by symbolic names.

**Meta-data** A descriptor block can be specified for every policy, including the author(s) and modification times of that policy, as well as a mandatory description. In addition, every instance of a policy element can include a (free-form) comment clause.

Specialised language elements are provided for associating policies and policy elements with IT management processes.

#### 4.1.6.4. XACML

As indicated by its name, the eXtensible Access Control Markup Language focuses on access control and privacy policy. In XACML, it is possible to express obligations associated with permitted actions. These include data obfuscation functions to ensure privacy, as well as e.g. erasure of data after a period of time for the same reason. The language does not, however, provide a means to specify exactly how such obligations are to be fulfilled. By design, it merely includes a facility to reference obligation functions defined elsewhere.

#### 4.1.6.5. WS-Policy

In contrast to the focus on expression of actual policies, the larger part of WS-Policy is dedicated to policy containment and transport. It views policy as a number of selectable options that are made available under certain conditions. This is different from management policy semantics in other languages. In fact, it can be said to constitute an independent type of policy, a *selection* policy type. Unfortunately, it does not provide a form that allows immediate expression of management policy. Therefore, although many of the criteria described in Section 4.1.5 may be fulfilled by the grammar given in WS-Policy, the different scope of this standard excludes it from the set of eligible formalisms.

#### 4.1.7. Summary of analysis results

The purpose of the analyses documented in this section was to identify suitable source and target languages for process translation. In the case of source languages a number of commercially relevant, formal process languages have been examined. The analysis was performed on the basis of a catalogue of criteria formulated according to the requirements on the representation of IT Management processes. In particular, the criteria reflected the intent to effectively automate the management processes, and to allow the integration of infrastructure management tools into the processes.

In conclusion, two alternatives prove adequate for the task. Both provide equivalent (in truth, near-equivalent) graphical and textual forms. The first pair consists of the graphical modelling language BPMN , for which XPDL 2 provides a textual notation. The second formalism is the UML Version 2, specifically the Activity Diagram portion of the specification. UML diagrams can be expressed in textual, machine readable form using XRI.

Of the other examined languages, WS-BPEL and ARIS/EPC lack important language features. They could be extended in a suitable manner to render them adequate for representing IT management processes in detail. In particular, future versions of the BPEL may incorporate the desired language elements.

The BPSS is unsuitable for representation of IT management processes. As it focuses on the document flow in business processes, the language lacks features that are critical if management processes are to be described in detail. Attempts to add such features would without doubt adulterate the intent of the language.

The preselection of the languages to be included in the analysis was founded on the their intended use. As they were designated to be used as target languages of a translation between management processes and policy rules, certain minimal segregation criteria could be applied. In particular, the expressiveness of a language with respect to obligation/management policy at an operational level was decisive. A number of languages were excluded from the main analysis procedure. In access-control centric languages such as XACML and EPAL the obligation policy construct was determined to be insufficient. Rei has a number of interesting features, but lacks important ones, such as the concept of events. WS-Policy is not designed to

express management policy; instead it provides a container that is intended to hold policy expressions.

Finally, three languages, PDL, Ponder and Propolis were examined in detail. As with the analysis of source languages, a catalogue of criteria was compiled and applied to every language. All three were found to be suitable to a satisfactory degree. The main differences between the languages are relevant only to non-critical criteria.

In summary, we can conclude that a satisfactory selection of both source and target languages exists. The approach presented in this thesis is intended to remain independent of a specific language. Therefore, for both process and policy languages, only the common features will be relied upon.

## 4.2. Meta-models of process representation

In order to accommodate a wide range of formalisms for process representation, assumptions made with regard to the structure of process definitions must be avoided. Nevertheless, it is necessary to describe the general structure of process definitions. In principle, a process can be said to be a directed graph consisting of vertices (nodes) that express e.g. activities, and edges (links) that connect pairs of vertices in one given direction.

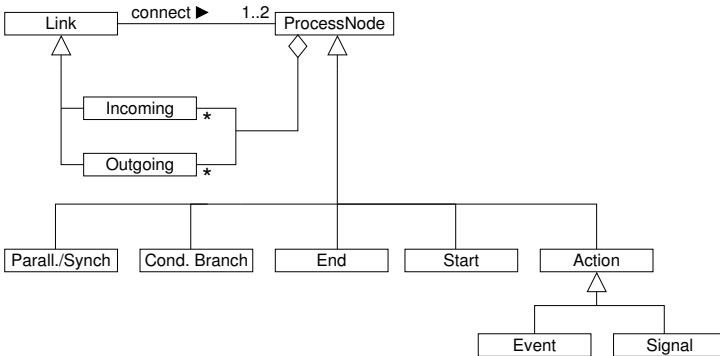


Figure 4.1.: Generic, control-flow-centric process meta-model

### 4.2.1. **Simple, generic meta-model for process definitions**

Representations of a process definition exhibit (when only considering the control flow in the process) at least five types of nodes: activities, branching/joining points, parallelisation/synchronisation points, starting nodes and end nodes. Actions (or activities) can be said to include the transmission of a signal (or message) as well as the act of expecting an event (or message). This basic meta-model for process definitions is summarised in Figure 4.1, taking into account only the control flow of the process (while signals and events may transport information, for the purposes of the model they are only viewed in their control-flow relevant role). Note that it does not constrain the “wiring” of vertices in the process graph. Arbitrary connections between nodes are allowed, as long as graph edges are connected to at least one vertex. Arbitrary bounds to association cardinalities may be imposed in the formalisms instantiated from this model. For example, starting nodes typically have no precedent nodes, while end nodes have no follower nodes.

The results of process language analysis, shown in Table 4.5 suggest that all formalisms described in this work are instances of the meta-model. The simple meta-model presented here factors out any detail of the process, as well as data flow considerations. Instead, it focuses on the control flow in a process.

The process translation approach employed in this work relies on the recognition of patterns for which corresponding, parametrised policy rule sets have been specified. Thus, input process specifications must be decomposed into partitions that can be matched against the patterns. Depending on the concrete process formalism/language employed to create a process specification, certain rules and constraints inherent to that formalism will be manifested in the process specification. However, such rules may vary across process formalisms (see Section 4.1). Hence, any assumption made with respect to the structure of a process graph would needlessly narrow the applicability of the approach presented in this work.

### 4.2.2. **Target meta-model**

Unfortunately, freedom in the choice of formalism, combined with a high degree of flexibility when modelling an IT management process results in a disadvantageous starting point for pattern-based process

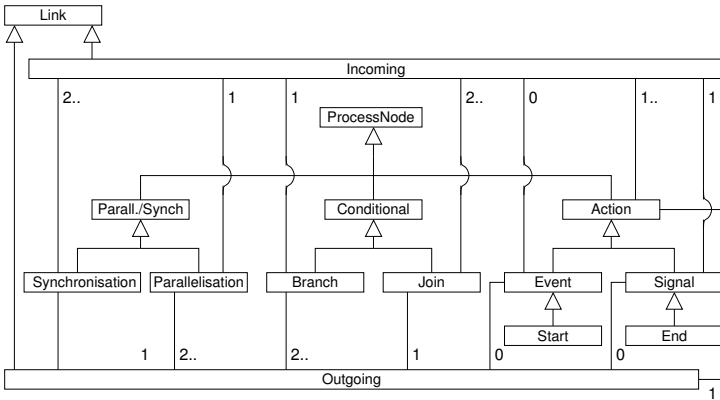


Figure 4.2.: Target meta-model

translation. Similar control flow paths may be modelled in a different manner, depending on the concrete formalism employed, the type of the process and the preferences of the person creating the model. To facilitate exact pattern matches in so diverse process definitions, the set of patterns would need to take into account these differences. In turn, this would result in a much larger set of basic patterns, more complex matching algorithms, as well as a higher probability of “orphan” process parts that are not matched by existing patterns.

A viable escape from this situation is the specification of constraints on the process definition to be translated. The *target meta-model* shown in Figure 4.2 takes into account cardinality bounds, as well as a number of assumptions that aid in the simplification of source process definitions. While the generic meta-model depicted in 4.1 describes the source formalisms for process representation, the target meta-model refers to the pattern set described in 4.5.

A number of assumptions are part of the target meta-model’s design. One of them allows the exclusion of start and end nodes by equating them with event and signal nodes, respectively. Another assumption is made with regard to process nodes that denote a branch in control flow, be it conditional and unconditional branching, or parallelisation of process execution threads: whenever execution flow is forked, it must be forked from a single thread. Hence, such nodes will always

Target meta-model specifies constraints

Assumptions in the target meta-model



<i>Process Node Subtype</i>	<i>Incoming</i>	<i>Outgoing</i>
Action	1..N	1
Signal	1	0
Event	0	1
Start	0	1
End	1..N	0
Parallelisation	1	2..N
Synchronisation	2..N	1
Conditional branch	1	2..N
Join conditional branch	2..N	1

Table 4.13.: Summary of process nodes cardinalities

have multiple outgoing links (edges) and exactly one incoming link. Similarly, nodes joining control flow, whether from parallel thread or from several possible execution paths chosen by means of conditions, will have multiple incoming links and exactly one outgoing link. The last notable feature of the target meta-model lies in the constraints imposed on actions. Actions need to be execution within an execution thread, hence they need at least one incoming link. A pure view of a process action (or activity) prohibits an action node to split or join control flow. Hence, an action will always have exactly one outgoing link. Table 4.13 summarises by node type the number of incoming and outgoing links a node may have.

The target meta-model can be employed to create a “pattern-friendly” but equivalent representation of given process. Instantiating the target meta-model yields the nodes shown in Figure 4.3. In essence, a process conforms to the target meta-model if it contains only the nodes shown in the diagram.

### 4.3. Substitution rules

Process definitions described by means of a formal language are expected to conform to the much more liberal meta-model shown in Figure 4.1, instead of to the target meta-model. They must be transformed into an equivalent form conforming to the target meta-model. This is achieved by substitution rules that translate an atomic fragment of the process into a representation consistent with the target meta-model – and thus into a form that is more amenable to pat-

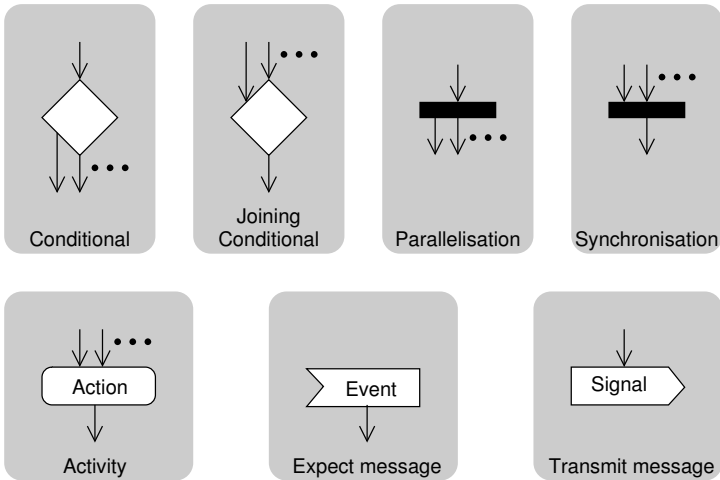


Figure 4.3.: Target process elements

tern matching. At a process graph level, the substitution rules yield production rules that may be used for node-for-node transformation of an input process.

Substitution rules effect the elimination of superfluous nodes, the elimination of superfluous links and the introduction of real as well as virtual nodes. They can be divided into context-free substitution rules derived directly from the target meta-model, and context-sensitive rules formulated with the idea of process-to-policy translation in mind.

### 4.3.1. Context-free substitution

The rules for context-free substitution regard a single node and the links attached to it. The transformation they specify does not take into account the type of the neighbouring nodes.

**R1– Elimination of branching/joining nodes** Branching and joining nodes may be considered redundant if both the number of incoming and the number of outgoing links equals zero or one (see Table 4.14). In the former case, the node is isolated and therefore useless. In the

<i>Process Node Subtype</i>	<i>In</i>	<i>Out</i>	<i>Verdict</i>	<i>Action</i>
Parallelisation/Synchr.	0	0	isolated	eliminate
Parallelisation/Synchr.	0	1	fake starting node	eliminate
Parallelisation/Synchr.	1	0	fake end node	eliminate
Parallelisation/Synchr.	1	1	ineffective	eliminate
Conditional branch/join	0	0	isolated	eliminate
Conditional branch/join	0	1	fake start node	eliminate
Conditional branch/join	1	0	fake end node	eliminate
Conditional branch/join	1	1	may be ineffective	eliminate if no guard condition

Table 4.14.: Elimination criteria for branching/joining nodes

latter case, it is ineffective. Hence, without changing the execution flow of the process, the node in question can be eliminated. Instead, its predecessor and follower (if present) are connected by a link.

In a more formal manner, the rule can be written as:

$$\{X_1\} \rightarrow Ko \rightarrow \{Y_1\} \mapsto X_1 \rightarrow Y_1$$

and

$$\{X_1\} \rightarrow Pa \rightarrow \{Y_1\} \mapsto X_1 \rightarrow Y_1$$

where  $X_1$  and  $Y_1$  are in the precedent and follower sets of the Ko/Pa node. Note that the formal notation only takes into account the case where there is one precedent node and one follower node; the remaining cases listed in Table 4.14 can be written in a similar manner. The  $\mapsto$  arrow denotes the substitution, while the short  $\rightarrow$  arrows denote links between nodes. The notation  $\{X, Y\} \rightarrow Z$  denotes a set of nodes consisting of  $X$  and  $Y$ , which all (both) link to  $Z$ . If several artefacts are created on the target side of the substitution, a  $\oplus$  sign will be used to delimit them; the right hand side of the rule can be read as “substitute this  $\oplus$  also this”.

There is a special case in which a conditional branching node has one incoming node guarded by a condition and one outgoing node. Such a node is not eliminated by this particular transformation.

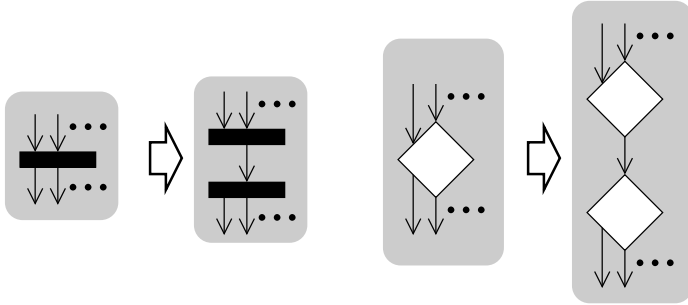


Figure 4.4.: Splitting conditional/parallelisation nodes

**R2– Splitting conditional/parallelisation nodes** In some cases, execution threads or control paths are joined only to be split again, perhaps according to the evaluation of conditional expressions. This could be expressed by conditional nodes with multiple incoming and multiple outgoing links. The same structure is conceivable for parallelisation/synchronisation nodes.

In such cases, the node in question can split into a branching and a joining node without changing the execution flow. The incoming links are connected to the joining node, while the outgoing links are connected to the branching node. The two “new”, resulting nodes are connected by a link starting at the joining node and pointing towards the branching node.

$$\begin{aligned} & \{X_1, X_2, \dots\} \rightarrow Ko \rightarrow \{Y_1, Y_2, \dots\} \\ \mapsto & \{X_1, X_2, \dots\} \rightarrow Ko_1 \rightarrow Ko_2 \rightarrow \{Y_1, Y_2, \dots\} \end{aligned}$$

$$\begin{aligned} & \{X_1, X_2, \dots\} \rightarrow Pa \rightarrow \{Y_1, Y_2, \dots\} \\ \mapsto & \{X_1, X_2, \dots\} \rightarrow Pa_1 \rightarrow Pa_2 \rightarrow \{Y_1, Y_2, \dots\} \end{aligned}$$

Figure 4.4 illustrates this substitution rule. In the case of a conditional node, the guard conditions on incoming links must, naturally, be retained.

### 4.3.2. Context-sensitive substitution

Realisation of IT management processes by means of policy rules relies on an event-based flow control. Hence, some features of a process that are implicit to an contiguous process definition need to be treated in a special manner.

One of these aspects is the exit from a process node, e.g. an action, and continuation of the control flow in another node pointed to from that action. In a graphical representation of a process, this seems trivial, however, to control execution, the termination of the action must be detected, and continuation of this execution must be facilitated.

Such events implicit to the traversal of the process graph can be introduced as artificial but explicitly modelled events into the process specification. The detection of such an event is signalled by a signal node, while the corresponding process partition (that should be notified of a change in state, e.g. the node following the action in the former paragraph) designates its expecting the signal by an event node.

As these are not “real” process nodes but runtime-dependent messages originating in the process execution infrastructure, they are called *virtual nodes*. Hence, a *virtual signal* is used to signify e.g. an action terminating, and a corresponding *virtual event* node can be used to trap the signal and react to it.

The following substitution rules make use of *virtual nodes* to decouple process partitions.

**R3– Eliminating joining conditional nodes** Some nodes join execution paths, without performing a synchronisation of previously parallelised threads of execution. They provide a common sink for multiple *possible* execution paths, and they are invariably represented by *Ko*-nodes. After having ensured that *Ko*-nodes are either opening, i.e. of the form

$$\rightarrow Ko \rightarrow \{Y_1, Y_2, \dots\}$$

or closing, i.e. of the form

$$\{X_1, X_2, \dots\} \rightarrow Ko \rightarrow$$

the joining nodes always constitute closing *Ko*-nodes.

Such a node can be eliminated by attaching a virtual signal to each of its predecessors, and creating a corresponding virtual event to link to its (single) follower. In our formal notation, this substitution can be written as follows:

$$\begin{aligned} & \{X_1, X_2, \dots\} \rightarrow Ko_1 \rightarrow \{Y\} \\ \mapsto & X_1 \rightarrow Si, X_2 \rightarrow Si, \dots \oplus Ev \rightarrow Y \end{aligned}$$

This substitution rule may seem awkward, since a clear, intuitive process structure is broken up and replaced with disparate signal/event pairs. Considering the translation target (event-driven management policy rules), however, the rule does provide several benefits:

1. it imposes an event-driven control flow
2. a passive (read: useless) process node is eliminated
3. a step is taken towards isolating process parts from each other, which will, in turn, be helpful to the pattern matching procedure

**R4– Decoupling Ko→Pa sequence** One effect of the previous substitution rule is to create a more loosely coupled process specification. The following rule has a similar scope. It aims to disconnect Ko and Pa nodes, which sometimes can create quite complex structures. As in the previous rule, it specifies a procedure to decouple a Pa node following a Ko rule by substituting a signal/event pair. The following formal notation describes the substitution rule:

$$\begin{aligned} & \{X_1, X_2, \dots\} \rightarrow Ko \rightarrow \{\dots, Y_{k-1}, Pa, Y_{k+1}, \dots\} \\ \mapsto & \{X_1, X_2, \dots\} \rightarrow Ko_1 \rightarrow Ko_2 \rightarrow \dots, Y_{k-1} Si_{Pa}, Y_{k+1}, \dots\} \\ & \oplus Ev_{Pa} \rightarrow Pa \end{aligned}$$

**R5– Decoupling Actions** Actions can be said to be the most important nodes in a process definition. They may be followers of conditional branch expressions, of parallelisation, or of each other. This does often not take into account the need to detect action termination. The following, simplified, substitution rule sketches the procedure that can be employed to retain the logical links of actions

to the rest of the process, while accommodating virtual signals and events that mark their beginning and termination.

$$\{X_1, X_2, \dots\} \rightarrow Ac\{Y_1\} \mapsto X_1 \rightarrow Si, X_2 \rightarrow Si, \dots \oplus Ev \rightarrow Ac$$

In some cases, the rule needs not be applied. Examples include the case when an action is preceded by (an) event(s). Obviously, there is no need to supplement that event by a virtual one; the possibly generated signal can be matched to the already existing event node. Another case is an action followed by a signal: there is no need to introduce an additional virtual signal. To maintain readability, the above form does not take these cases into account.

### 4.3.3. Order of application

The order in which the substitution rules are applied to a process specification impacts upon the result. While repetitive application of the rules in any order will result in the same target process specification in the end, to minimise the number of passes, the rules should be applied in the order they have been listed in this section.

## 4.4. Methodology for translation

The source and target models governing the translation have been established in the previous section. This section focuses on the methodology employed to translate a process specification, beginning with its original form, via the form ensuing as a result of application of the substitution rule, up to the point where a set of policies has been created that represents the original process (and, certainly, its intermediary forms). Once the translation is complete, the management policy rules may be distributed to their intended (sub-)domain, and provisions must be made to ensure the data flow required by the process is realised.

### 4.4.1. Outline

The high-level translation procedure is explained in the following. Details regarding some of the steps in the methodology are given in subsequent parts of the work.

- Step 1** – **Process review** In this step, the original process is assessed, to ensure that the process specification complies with the requirements imposed by the actual pattern-based translation mechanism.
- Step 2** – **Substitution** In this step, the substitution rules are applied.
- Step 3** – **Fragment matching** In this step, the process fragments are matched against the pattern catalogue.
- Step 4** – **Data flow analysis** In this step, the data flow in the process is traced.
- Step 5** – **Rule generation** In this step, management policy rules are generated according to the parametrised rules attached to each pattern.
- Step 6** – **Deployment** Once the process specification has been translated into a set of management policies, the latter must be deployed in order to execute the process.

#### 4.4.2. Description of steps

The following description of the methodology steps provides a more in-depth view.

##### **Step 1** Process review

The input to the translation method described in this work is a process specification written in a formal language. The final goal is to successfully create executable management policy rules. The objective of the first step of the methodology is to ensure that the process, as modelled, is suited and ready for translation. This pre-assessment is based on the following requirements, which must be fulfilled by the process specification:

- *Compliance to meta-model* The process specification is modelled by means of a language that complies to the generic meta-model.



- *Machine readability* The process specification is available in a machine readable form, or can be exported into such a format.
- *Sufficient detail* The level of detail in the process specification provides “enough” information to be automatable. Unfortunately, this criterion is difficult to substantiate enough to be unambiguous, due to the different context in which a process specification may be created. As a guideline, if the information required to drive the execution of the IT management process in question is present, the level of detail can be deemed sufficient. This includes taking into account the interfaces, formats etc. dictated by the managed infrastructure, the management tools and the tools for process support.
- *Syntactic correctness* Syntactic errors in the process will lead to undesired results, no matter how process automation and execution are realised. In the case of the approach discussed in this thesis, syntactic faults may lead to erroneous partitioning of the process, failed pattern matches, and of course erroneous management policy. The use of modelling tools may help fulfil this requirement while the process specification is in-the-making.

If the process specification fulfils the above requirements, the next step in the methodology may be executed. If not, the formal specification needs to be revised until the demands made in this step are met.

Iterative  
improvement

### **Step 2** Substitution

As detailed in Section 4.2, the input process specification may be compliant to any language that is possible to instantiate from a generic meta-model. In this step, the process specification is made compliant with the more constrained target meta-model. For this reason, the substitution rules described in Section 4.3 are applied to the input process.

After this step, the process specification should have a form compliant with the target meta-model. To achieve this effect, the substitution rules need to be applied in the proper order. In most cases, the resulting process specification will consist of several, isolated graph partitions (fragments). As a rule, after having applied the substitution rules, the resulting fragments tend to be small.

### **Step 3** Fragment matching

The translation mechanism employed in this work is based on the matching of process fragments against a set of pre-defined process patterns, like the ones described in Section 4.5. In this step, the process fragments acquired in the previous step are matched against the patterns in the catalogue.

Matching must be exact, i.e. the structure of the fragment in question must exactly match the structure of a given pattern, as well as the exact node denominations. However, for the scope of pattern matching, virtual nodes are viewed as equal to “natural” ones. For example, an event node occurring in a pattern can be matched by an event that was present in the process specification reviewed in step 1; it can also be matched by a virtual event node introduced in step 2.

All fragments should be matched in this step. In the case of “orphan” (i.e. unmatched) fragments, human intervention becomes necessary. A possible cause of this situation can be errors in the original process specification. This situation also occurs when a new pattern is encountered.

### **Step 4** Data flow analysis

The substitution rules, as well as the patterns, focus on control flow. In contrast, this step ensures that every process node is provided the information it needs in order to be effective. This includes identifying data records passed between activities. In particular, actions need to be supplied the required parameter valued, and conditional branching nodes need to be provided the values required by the guard conditions. Signals that pass control flow to another process part need to have access to the information required in that part of the process.

To be able to ensure that the data flow within the process is not impeded or interrupted, the data records that are to be passed between nodes need to be identified. In addition, the information items that some of the nodes may depend on need to be identified. If the information to be transmitted is of a primitive data type (e.g. a named integer or string), it can be included in an event structure. In complex and non-translatable cases, e.g. with documents, named

Handling anomalies

wrappers around these data items need to be created. In such a case, a reference to such documents can be passed within the process. The runtime environment is responsible for de-referencing the information objects in question.

Before generation of the rules in the next step, the information objects need to be attached to the process nodes they are referenced by. The actual procurement of information at runtime is discussed in detail in Chapter 5.

Some IT management process elements will include interaction with an administrator. Examples include notifications or the request for a decision. In this step, it must be ensured that the necessary infrastructure is in place to handle such cases. In simple cases, the presence of a transfer agent for internet email could suffice.

#### **Step 5** Rule generation

The preceding steps have ensured that every part of the process is matched by a pattern (see Section 4.5), and that the correct information objects have been mapped to every process node. In this step, the parametrised templates in the patterns are specialised for every instance of the matched process fragments.

#### **Step 6** Deployment

Strictly speaking, deployment does not constitute a part of the translation methodology. However, deployment and commissioning of the policy set is of great importance to the approach as a whole. After translation has been completed in the previous step, the management policy rules should be distributed, if possible, to their domain of action. If a centralised policy architecture is used, the rules should be loaded and activated. Previous rules should be deactivated (e.g. if the process had been deployed before).

The data record definitions should be passed to the entity that governs process data flow, e.g. to the service monitoring architecture described in this work. If necessary, data acquisition rules need to be specified and activated.

### 4.4.3. Discussion

The above description of the methodology for translation relies on a number of implicit assumptions. These assumptions allow the description to be kept concise and clear. However, they also lead to the neglect of plausible situations that will be discussed in the following.

One of the aforementioned assumptions is that any action can be performed by a machine. Hence, until now, we have not taken into account the actions that will be delegated to human operators, or the information/data items that need to be available to them. Certainly, this issue must be resolved, lest the process will be impeded, however, its characteristics are highly dependent on the infrastructure, tools etc. that may be in use at a site.

Another issue may be found with the handling of irregularities in the execution of the methodology. The general instruction provided in the text can be seen as guidelines to resolve problems occurring during process-to-policy translation. Nevertheless, the specific actions to be taken are, again, dependent on the actual setting, as well as possibly on the process specification.

In conclusion, such issues cannot be effectively addressed in the methodology. They must, unfortunately, be countered with “best practice” knowledge originating in the domain of application of the methodology.

As becomes apparent from the description of the translation procedure, the pattern catalogue for translation is critical to the realisation of the translation. It will be described in the next sections.

## 4.5. Fundamental patterns

Process specifications are imperative in nature. They rely on state (process state) as well as control on the degree of parallelism in their execution. The sequence of actions is rigid, given by the specification. These characteristics must be conserved by any implementation of the process.

When realising process control flow by means of management policy, a number of measures must be taken in order to ensure the correct execution of the process. Typically, policies themselves are executed

Process execution is stateful

Policies keep no state

in isolation, and transferring no state information about a system or a process. The sequence, in which policy actions are executed is difficult to predict as there is no real dependency between the execution of the actions in one policy, and the execution of the actions in another.

The approach described in this work takes these issues into account by employing *translation patterns*. The patterns consist of a small, generalised piece of process specification and a likewise parametrised management policy rules.

Pattern =  
process  
fragment +  
parametrised  
rules

A number of fundamental patterns can be identified. They represent common process fragments and include basic/ECA, condition and synchronisation patterns. Each pattern has a number of *variants* that account for different possible contexts of a pattern. The development of pattern variants yields a catalogue of patterns to match against process specifications.

Pattern  
variants

Every process activity, as well as every management policy action depend on information that originates inside or outside the process. The connection between an action and this information (e.g. systems management information, process artefacts) needs to be taken into account by every pattern translation (i.e. rule set). However, this can be done separately from the treatment of control flow.

Dealing with  
data flow

### 4.5.1. Basic pattern

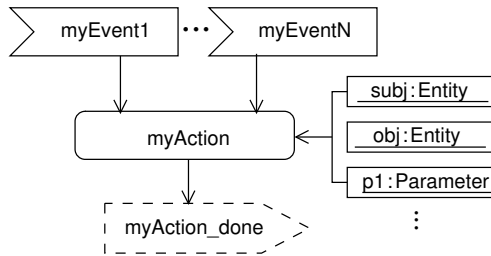


Figure 4.5.: Basic pattern

The control flow common to every process and every policy language is that of an event (either an explicit message being passed between two points, or a change of state) triggers the execution of an action.

Though some policy languages lack the concept of events and rely solely on conditions to determine when a policy is to be enforced, the evaluation of such conditions must be triggered somehow – thus, implicitly, an event concept is introduced.

The simplest patterns that can be produced with solely event and action elements is the one depicted in Figure 4.5. Note that the pattern allows different events to trigger the action. Translation of instances of this process pattern yields one or more policies, depending on the policy language used: if the language supports specification of several triggering events for its action, a single policy will suffice. If the policy formalism limits the number of events specified in a policy, the translation should produce a number of policies inversely proportional to the limit imposed, yielding at most the number of policies corresponding to the number of events specified in the pattern instance being translated.

The termination of the action is signalled by the (possibly virtual) signal

`myAction_done`. If, in the process specification itself, the action is followed directly by a “real” signal or message transmission, the virtual signal is, of course, unnecessary.

The basic pattern can be translated into a single policy as follows:

```
policy
{
    event { myEvent1, ..., myEventN }
    subject { subj }
    target { obj }
    action { myAction(p1, ...); }
}
```

#### 4.5.2. Condition patterns

Control flow decisions in processes are often based on conditions. A simple pattern supporting this feature is the condition pattern. Although it is an extension of the basic pattern described above, the condition pattern is more complex in that it can be iterated and nested.

The condition pattern in its simplest form is shown in Figure 4.6. As suggested in the diagram, the action is only executed if the event occurs and the condition holds true. If the condition does not hold

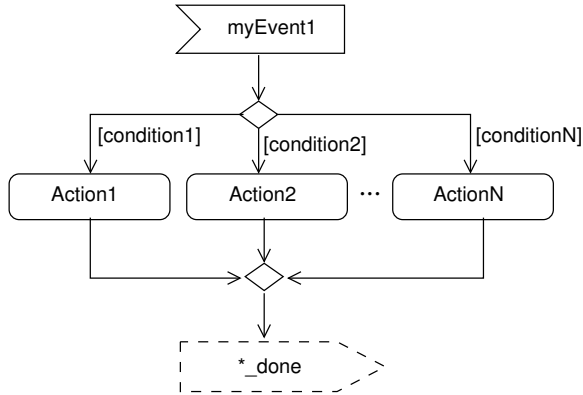


Figure 4.6.: Disjunctive condition pattern

true, the control flow branch ends. This simple form constitutes, in fact, a disjunctive condition pattern, as described below. Again, as in the case of the basic pattern, a signal is employed in order to mark the end of the conditional execution. Note that, according to the substitution rule eliminating joining nodes (see Section 4.3), this signal may be attached to each of the actions in the figure, instead of to the joining node. This allows greater flexibility when disjunctive condition patterns are nested within a complex control flow; it constitutes a variant of this pattern.

If such a pattern is encountered, it will be translated into  $N$  policies, where  $N$  is the number of branches that attach to actions. Each policy will trigger on the event (or event set) leading to the conditional branch node (in the figure, `myEvent1` occupies that role). Furthermore, it will carry the condition relevant to “its” specific branch, as well as the action to be performed in case the condition holds true. Thus, the translation of the pattern as shown in Figure 4.6 will generate policies of the form:

<b>policy</b>		<b>policy</b>
{		{
<b>event</b> { myEvent1 }		<b>event</b> { myEvent1 }
<b>action</b> { Action1 }	...	<b>action</b> { ActionN }
}		}
<b>condition</b> { condition1 }		<b>condition</b> { conditionN }

**Execution** The execution of the different policies is determined by the condition it carries. From the generated set, one rule will be executed at most when an instance of `myEvent1` is received. If none of the conditions evaluate to true, no rule is executed.

Unguarded  
default branch

To accommodate the possibility of *default branches* in processes, a number of additional steps must be taken. A default branch is one that is executed when the guard conditions on all other branches are evaluated to `false`; it carries no condition itself. There can be a maximum of one default branch in a pattern; otherwise, the process would be non-deterministic (there would be several coequal branches to select from), and that situation should be flagged as an error to the designer of the process. If a default branch  $K$  is present in the pattern, the policy that corresponds to that branch would carry a condition that negates all other conditions in the set, yielding a negated disjunctive normal form:

```

policy
{
    event { myEvent1 }
    action { ActionK }
    condition { NOT (condition1 OR ... OR conditionN) }
}

```

#### 4.5.2.1. Disjunctive condition pattern

Process actions may be due for execution of any of several conditions in a set hold true. Thus, if any in a set of conditions  $c_1 \dots c_n$  should trigger the action, the conditions are in disjunctive normal form (DNF), as shown in Figure 4.7.

Translation of such a pattern instance ideally results in a single policy, if the policy language supports a DNF in its condition field. In the worst case, a number of policies corresponding to the number of conditions in the set should be created, each of which contains the same action and one of the conditions in the set. Note that, since the condition pattern is derived from the basic pattern, the number of policies generated also correlates to the expressiveness of the policy language with regard to events. Hence, a language allowing only one event and only one single condition per policy would require a total of  $E * C$  policies, with  $E$  being the number of events specified in the pattern instance, and  $C$  the number of conditions. In the



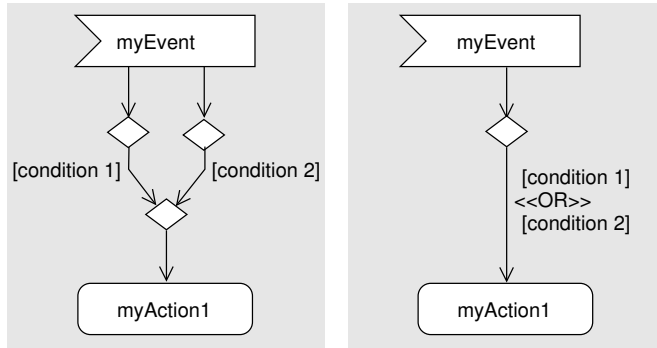


Figure 4.7.: Disjunctive condition pattern with single action

form shown in Figure 4.7, the policy generated would be of the form:

```

policy
{
    event { myEvent }
    action { myAction1 }
    condition { condition1 OR ... OR conditionN }
}

```

The left and right box in the figure are equivalent, i.e. they correspond to the same policy set.

#### 4.5.2.2. Conjunctive condition pattern

The converse case of conditional execution of an action is to pose a number of conditions that must *all* hold true (see Figure 4.8). This corresponds to a conjunctive normal form (CNF) of the conditions. Again, translating to a policy formalism allowing CNFs of conditions will yield a single policy. Formalisms disallowing CNF in its condition field would call for workarounds, since a CNF cannot be emulated by simply generating more policies. The concept of *intermediate events* described in Section 4.5.3 can be used to address this problem. Similarly to the disjunctive condition pattern the generated policy set would be of the form:

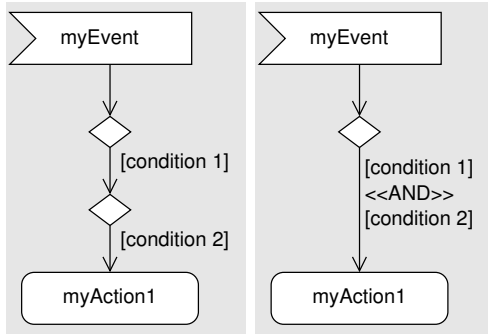


Figure 4.8.: Conjunctive condition pattern

```

policy
{
  event { myEvent }
  action { myAction1 }
  condition { condition1 AND ... AND conditionN }
}

```

#### 4.5.2.3. Mixed condition pattern

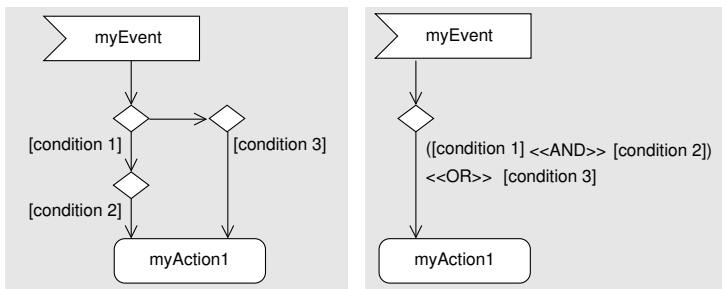


Figure 4.9.: Mixed condition pattern

Any combination of the above alternatives can be expected to occur in process specifications. Figure 4.9 shows an example of nested conjunctive and disjunctive expressions. Depending on the structure of the pattern instance to be processed, a DNF containing conjunctive

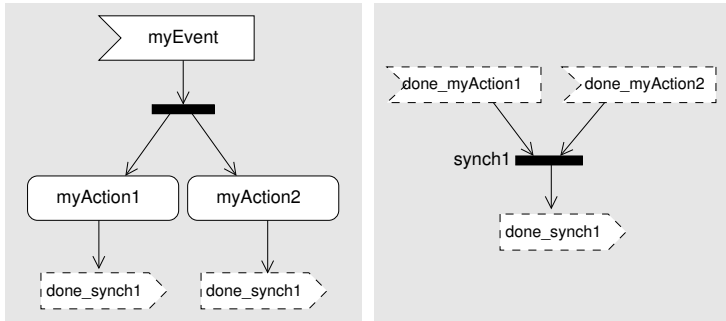


Figure 4.10.: Synchronisation pattern

expressions or a CNF containing disjunctive expressions will result as a policy condition. A pattern instance as shown in the figure would translate into a policy rule of the form:

```

policy
{
    event { myEvent }
    action { myAction1 }
    condition { (condition1 AND condition2 AND ...) OR (condition3
    ... ) OR ... }
}

```

### 4.5.3. Synchronisation pattern

To effectively synchronise control flow after the parallel actions have been executed it is necessary to determine when action execution has completed. The intermediate events shown in Figure 4.10 indicate the points where such detection is necessary. Most communication middleware facilitates detection of terminating actions so that programmatic realisation of intermediate events is feasible.

Parallelisation and synchronisation points in processes must be treated separately. There is no guarantee that every parallelisation into separate execution threads corresponds to a synchronisation point somewhere in the process, or, for that matter, that they will be joined at all. Conversely, there is no guarantee that the execution threads to be synchronised have been parallelised at the same point in the

process.

**Parallelisation** The processing of parallelisation nodes is the easier part. Assuming parallelisation to a number of “threads”, each node directly following the parallelisation node represents one control flow thread. In addition, each such node will be either an action or a signal node. As management policy rules are inherently easy to parallelise, this pattern results in a number of policy rules corresponding to the direct followers of the parallelisation node, in the following manner:

```

policy
{
    event { myEvent }
    action { Action1 }
}
...
policy
{
    event { myEvent1 }
    action { ActionN }
}
    
```

Those rules representing a signal node will carry the transmission of that signal as their action.

**Synchronisation** Synchronisation points (denoted by synch bars in the diagrams) can be seen as general *barrier synchronisation* items: all specified incoming links must be satisfied for the control flows to be joined, and for the synch bar to be “passed”. In particular, this means that all actions leading to the synch bar have terminated, i.e. that all events (virtual or not) have been observed. Depending on the features supported by the target language, this may require an external facility that is able to maintain the state of a synchronisation instance.

Several information items regarding state must be taken into account.

**Handling of different process instances** Several instances of a process specification may be executed concurrently. For example, several instances of the incident management process may be simultaneously active in order to cope with multiple, concurrent incidents or service request. The synchronisation must differentiate between arriving events or terminated actions that pertain to different process instances.

**Detection of the synchronisation condition** A synchronisation node is only useful if it possesses more than one

incoming link. In fact, after having applied the substitution rules discussed in Section 4.3, this condition should apply to all such nodes that are left in the process specification. To account for all incoming links, either an appropriate language construct can be used (if available), or a synthetic, virtual signal must be introduced. This signal can be issued to mark the completion of all incoming links, i.e. all actions have terminated and all events have been received. To avoid making assumptions regarding the capabilities of the target language, we will assume that no special language feature for event correlation is available.

**Signalling mechanisms** The knowledge required to provide signalling for the synchronisation points may be distributed over several domains. In essence, the mechanism must be able to detect the termination of actions pertaining to the synchronisation node, as well as the reception of events associated with it. In addition, the service must be able to determine the completion of a barrier synchronisation, i.e. the point in time when *all* relevant action terminations and events for a specific process instance have been detected. Detection of the termination of actions is really an implementation issue that can be easily solved when employing communications middleware. Note that the pattern example in Figure 4.10 specifies only incoming events; this is a situation that can be relied on, after the substitution rules have been applied. The correlation mechanism itself can be realised by an architecture for message aggregation, as the one discussed in Section 7.4.

**Time limits** It may be desirable to render synchronisation points “interruptible”, i.e. prevent a process instance from halting indefinitely in the face of a synchronisation point. This can be realised by artificially transmitting the event following the synchronisation bar (the one named `done_synch1` in Figure 4.10), e.g. by means of a timer that is started when the first top-side event of the synchronisation point (for a certain process instance) is received.

As synchronisation points’ only function is to correlate several events into one, their execution is left to the event management mechanism that will be employed.

#### 4.5.4. Discussion

The patterns described in this section constitute a most basic set. Together, they form a process modelling kit that could be used to address most, perhaps all, structural modelling requirements for IT management processes. However, unless the opposite is formally proven, it is possible that a process specification created in a real-life setting will employ syntactic variants which do not match any of the patterns that have been discussed in this section. Tests conducted on random processes indicate that this is not the case. Nevertheless, it is important to provide the ability to deal with such cases, should they arise.

Since it is important that processes be decomposed in their totality in every case, Section 4.7 discusses the generating system for process specifications. It provides a means to decompose any process fragment in a manner that will allow its translation into policy rules. Decomposition by means of the generating system will, in most cases, incur a penalty in the number of policy rules generated, as well as in the number of (virtual) events and signals that need to be generated and handled by the runtime system. Therefore, it should be reserved for cases when pattern-based translation attempts have failed.

Another issue with the fundamental pattern set is that of efficiency. Decomposition of process specification into their smallest components (that correspond to these patterns) may incur more communication overhead than necessary.

The techniques employed for modelling of IT management processes are still evolving at the time of this writing. As with other modelling and specification domains, it not unreasonable to expect best practices not only for the content of the processes, but also for the formal model employed to describe it. In concrete terms, design pattern catalogues like those created in the domain of software engineering could ensue from analysis and standardisation efforts. If such design patterns for process should become available (and widely used), it would certainly make sense to adapt the catalogue of translation patterns accordingly. A number of techniques suitable for dealing with the above issues are described in Section 4.8.

To effectively put into practice the pattern-based translation mechanism process fragments must be matched to patterns and, upon a match, the corresponding policies generated from the information

Totality ensured by generating system

Customised patterns to improve efficiency

present in the fragment. The next section describes the procedure to detect and translate the patterns described in this section.

## 4.6. Detection and translation

The identification of patterns is a prerequisite for the proposed translation scheme. Identification implies the decision whether a process fragment matches any of the patterns in the pattern catalogue (in the event that one is encountered that does not, the concepts in Sections 4.7 and 4.8 may be used to “force” a match). The procedure to distinguish between fragments matching different patterns is described in Section 4.6.1.

The subsequent attribution of the process fragment to one specific pattern allows the extraction of information from its nodes in order to instantiate the set of policies corresponding to that pattern. Section 4.6.2 discusses the procedure employed to achieve the generation of policies from matched process fragments.

### 4.6.1. Fragment discrimination

Basically, to identify the pattern matched by a fragment, the fragment is compared to every pattern in the catalogue until a match is found or until there are no further patterns to compare to. The strategy of exclusion employed to determine whether a fragment matches leverages the structural similarities between patterns.

All patterns presented in Section 4.5 share a number of properties. In every one of them, a “running direction” can be identified. The pattern can be said to “begin” with one or multiple event nodes and continue with nodes distinctive for one pattern or a group of related patterns. It is important to note that event nodes are found *only* at the beginning of the pattern, the *beginning* node(s) being defined as not possessing inbound links.

Common  
properties of  
patterns

Hence, given a process fragment  $F$  of the form

$$F = \{Ev_1, \dots, Ev_N\} \rightarrow \{Y\}$$

the value of  $Y$  determines whether the fragment matches a pattern or not. Considering the shapes of the patterns described in Section

4.5, the node immediately following the  $\{Ev_1, \dots, Ev_N\}$  allows a preliminary classification of  $F$ .

First  $Y$  node  
indicates  
pattern

To wit, an  $Ac$  node following the event(s) indicates a possible match to a basic pattern, a  $Ko$  node suggests that one of the conditional patterns might match, and a  $Pa$  node identifies the pair of patterns representing parallelisation and synchronisation as potentially matching.

Anomaly  
treatment

By exclusion, the only nodes not covered by the above enumeration are  $Ev$  and  $Si$  nodes. These two nodes following the set of events  $\{Ev_1, \dots, Ev_N\}$  indicates an anomaly either in the process specification (in the case  $\{Ev_1, \dots, Ev_N\} \rightarrow Ev_Y$ ) or in the fragmentation procedure (in the case  $\{Ev_1, \dots, Ev_N\} \rightarrow Si_Y$ ). The former issue cannot be corrected during translation at all; it needs to be solved in the process specification itself, as it constitutes a non-compliance to the source meta-model assumed for the original process specification (see Section 4.2).

The latter case may indicate an overzealous implementation of the substitution rules in the cases where a single virtual event is followed by a (possibly virtual) signal. This can occur at the point where natural signals are present in the original process, and, treated as actions, they are decoupled from the process specification by means of a virtual signal/event pair. In this case, the fragment can be eliminated by reconciling the natural signal with the virtual one, i.e. by substituting the natural signal for the faultily inserted virtual one. All other forms in the latter case, in particular instances of a natural signal node following a natural event, indicate non-compliance to the assumptions underpinning the fragmentation procedure.

**Algorithm for initial classification** Based on the case differentiation above, a simple algorithm for the distinction of fragments according to patterns can be given, as outlined by the following pseudo-code:

```
enumeration ptype =
    { basic, condition, para, synch, fault, none } ;
ptype candidate;

switch (Y.type)
{
    case Ac : candidate = basic; break;
    case Ko : candidate = condition; break;
    case Pa :
        if (Y.inboundLinks = 1 )
```



```

        candidate = para;
    else
        candidate = synch;
    break;
case Ev : candidate = fault; break;
case Si : candidate = fault; break;
default : candidate = none; break;
}

```

The above algorithm performs a coarse initial classification of a fragment. Its type is determined within the elements of the enumeration given in the first line to be either that of a basic pattern (**basic**), a flavour of conditional pattern (**condition**), a parallelisation (**para**) or a synchronisation (**synch**). Basic and conditional patterns are suggested plainly by the node following the event set, while parallelisation and synchronisation are differentiated between according to the number of event nodes (i.e. inbound links) of the node following the event set.

After execution, the variable **ptype** holds the notional type of the fragment. As already noted, values of *Ac*, *Ko* and *Pa* suggest a valid pattern, while values of *Ev* and *Si* indicate an anomaly.

The classification of fragments according to the algorithm given above can be viewed as negative definite: it is an indication of the patterns that a given fragment *cannot* match, i.e. all others but the one indicated. It does not guarantee that the fragment will match the indicated pattern.

Negative  
definite  
classification

#### 4.6.2. Algorithms

A positive definite match requires a pattern specific treatment of the fragment in question. In the following, the procedures for the confirmation of pattern identity are outlined as a requisite for the translation algorithms presented in this section.

The illustration of the algorithms in pseudo-code assumes the definition of the following:

1. A class or structure named **Node** that is suitable to represent a process node.
2. A class named **Policy** that contains a set of events, a set of actions, as well as condition.

The purpose and function of the methods called on these classes are obvious from their names.

Fail fast

Each algorithm pursues two goals: first, to ascertain that the fragment does in fact match the pattern in question. If it does, the algorithm proceeds to translate the fragment into a set of policy rules. If the fragment does not match, the algorithm fails explicitly. The next pattern can then be tried, as suggested in Section 4.6.1. In the following, the patterns introduced in Section 4.5 are treated in order of the complexity of their translation.

**Basic pattern** The handling of the basic pattern is comparatively simple. It consists of an event node (set), followed by an action node, followed by a signal node. Thus, it positively matches any fragment  $F_b$  of the form

$$F_b = \{Ev_1, \dots, Ev_N\} \rightarrow \{Ac\} \rightarrow \{Si\}$$

where  $Si$  may be a natural or virtual signal. If the fragment adheres to this form strictly, the identification of a basic pattern is confident.

The following algorithm matches the basic pattern (or fails explicitly) and generates the policy ensuing from the process fragment:

```

Node action, signal;
action = Ev1.follower;
if (action.type ≠ Ac) FAIL;
/* Make sure that all Evi connect to the same Ac node.*/
foreach Evi do
    if (Evi.follower ≠ action) FAIL;
/*Make sure Ac node is followed by Si node.*/
signal = action.follower;
if (signal.type ≠ Si) FAIL;

/*ID positive; Ready to generate*/
Policy p;
foreach Evi do
    p.event.add(Evi.name);
p.action.add(action.content);
p.action.add(sendSignal(signal.name));
/*Policy p is completed.*/
    
```

**Parallelisation** The parallelisation pattern matches fragments  $F_p$  of the form

$$F_p = Ev \rightarrow Pa \rightarrow \{[Ac_1 \rightarrow]Si_1, \dots, [Ac_N \rightarrow]Si_N\}$$

in which the *Pa* node is followed by either an action node that is followed by an (optionally virtual) signal ( $Ac_x \rightarrow Si_x$ ), or by a natural signal node<sup>2</sup>. To ascertain the match to a parallelisation pattern, every branch following the *Pa* node must comply to this rule. Thus, three items must be checked in order to confidently identify a parallelisation pattern:

1. The fragment begins with a single *Ev* node.
2. The *Ev* node is followed by a *Pa* node with at least two followers.
3. Each follower of the *Pa* node is of the form  $[Ac \rightarrow]Si$ .

To ascertain the match and generate the policy rules, we use the following algorithm:

```

Node event = Ev, bar, action, signal;
bar = event.follower;
Policy p; /* Ascertain base structure */
if (bar.type ≠ Pa) FAIL;
if (bar.numberofPrecedents ≠ 1) FAIL;
if (bar.numberofFollowers < 2) FAIL;

/*Generate*/
foreach action in bar.followers do
{
    p = new Policy();
    switch (action.type)
    {
        case Ac :
            signal = action.follower;
            if (signal.type ≠ Si) FAIL;
            p.event.add(event.name);
            p.action.add(action.content);
            p.action.add(sendSignal(signal.name))
            store p; //policy instance completed
            break;
        case Si :
            p.event.add(event.name);
            p.action.add(sendSignal(signal.name));
            store p; //policy instance completed
            break;
        default:
            FAIL;
            break;
    }
}

```

---

<sup>2</sup>The brackets indicate that the *Ac* node and the link are not mandatory.

The execution of the procedure above results in  $N$  policy rules that correspond to the branches of the parallelisation pattern. Each rule contains an action (or a signal to be transmitted) that is assigned to a different policy rule. The rule trigger on the same event and should therefore be executed in parallel in the policy system.

**Synchronisation** The synchronisation pattern matches process fragments  $F_s$  of the form

$$F_s = \{Ev_1, \dots, Ev_N\} \rightarrow Pa \rightarrow Si$$

with  $N \geq 2$ . Unlike its sibling parallelisation pattern, the synchronisation pattern has no variants. In consequence, it is sufficient to ascertain that

1. The fragment begins with a number  $\geq 2$  of  $Ev$  nodes, all followed by the  $Pa$  node.
2. The  $Pa$  node is followed by a single signal event (natural or virtual).

A fragment matching the synchronisation pattern specifies that a signal be transmitted when the barrier synchronisation point has been passed. The following algorithm assumes that a suitable mechanism is present to ascertain that this synchronisation condition is detected. In consequence, a single policy rule will be generated from fragments matching this pattern; the aggregated event is specified in the policy's event clause.

```
Node bar = Ev1.follower, signal.
Policy p;
/* Make sure that all Ev_i connect to the same Pa node.*/
foreach Ev_i do
    if (Ev_i.follower ≠ bar) FAIL;
/* Make sure that the synch bar is followed by a signal*/
signal = bar.follower;
if (signal.type ≠ Si) FAIL;
/*Generate*/
p.event.add(Ev1 ∧ ⋯ ∧ Ev_N); //aggregation
p.action.add(sendSignal(signal.name));
/*Policy p completed*/
```

**Conditional patterns** The conditional constructs constitute the most multi-faceted instances in the pattern catalogue. In consequence, we

must differentiate between the disjunctive form and the conjunctive form (see Section 4.5.2).

The disjunctive conditional pattern represents a number of branches that are reachable by means of links guarded by mutually independent conditions. As illustrated by Figures 4.6 and 4.7, a fragment  $F_d$  compliant to the disjunctive conditional pattern is of the form

Disjunctive  
conditional

$$F_d = Ev \rightarrow Ko \rightarrow \{Ac_1[\rightarrow Si_1], \dots, Ac_N[\rightarrow Si_N]\}$$

where each link leading to an  $A_i$  is guarded by a condition. Note that the signal nodes in the expression are not shown in the figures. An alternative form of the pattern has several guarded branches leading to a single  $Ac$  node, as in:

$$F_d = Ev \rightarrow \{Ko_1, \dots, Ko_N\} \rightarrow Ko_{join} \rightarrow Ac[\rightarrow Si]$$

Both forms rely on multiple conditions being present to make sense, i.e.  $N \geq 2$ , though a degenerate form with  $N = 1$  can be conceived. The identification of this pattern requires a number of case differentiations. A fragment is determined to match the disjunctive conditional pattern if

1. The  $Ev$  node is followed by one  $Ko$  node and
2. the  $Ko$  node possesses multiple outbound guarded links and
  - a) the outbound links lead to  $Ac$  nodes or
  - b) the outbound links lead to a (joining)  $Ko$  node that is followed by an  $Ac$  node.

or if

1. The  $Ev$  node is followed by multiple  $Ko$  nodes with outbound guarded links and
2. the outbound links lead to a (joining)  $Ko$  node that is followed by an  $Ac$  node.

The conjunctive conditional patterns are the most straight-forward variants in the group of conditional patterns. As the guarded links are in sequence, this variant is also more easily identified. A fragment  $F_c$  matching the conjunctive conditional pattern has the form

Conjunctive  
conditional

$$F_c = Ev \rightarrow Ko_1 \rightarrow \dots \rightarrow Ko_N \rightarrow Ac[\rightarrow Si]$$

where, as with the disjunctive variant,  $N \geq 2$  is expected. To identify this pattern with confidence we need to ascertain that:

1. The initial  $Ev$  node is followed by a single  $Ko$  node.

- Each *Ko* node is followed by a single *Ko* node, or by an *Ac* node.

The mixed conditional pattern results from the combination of disjunctive and conjunctive forms. For the purposes of identification, it can be reduced to a nesting of those forms. Thus, an algorithms that is able to handle fragments matching the mixed conditional will be able to handle the particular cases of pure conjunctive and disjunctive patterns.

The following recursive algorithm has been devised to handle mixed conditional patterns of arbitrary depth. To realise recursion, a function is defined that takes as formal parameter the branch node currently being examined.

```

/*Ensure we examine a conditional branch pattern*/
Policy p;
Node ko = Ev.follower;
Node action;
Condition c;
if (ko.type != Ko) FAIL;
/*Examine one conditional level*/
function examineKo (Node koroot)
{
    c.add(^ koroot.content);
    Node follower;
    foreach follower in koroot.followers do
    {
        if (follower.type == Ko)
            examineKo(follower); //recurse
        else //either Ac or Si
        {
            p.event.add(Ev);
            p.condition.add(c);
            if (follower.type == Ac)
                p.action.add(follower.content);
            else if (follower.type == Si)
                p.action.add(sendSignal(follower.name));
            else FAIL;
            store(p); //add rule to generated set...
            p = new Policy(); //...and provide a fresh instance
        }
    }
}

```

The algorithm above relies on the function `examineKo` to perform a depth-first traversal of nested conditional structures, in order to cope with mixed (combined) conditional patterns. Each consecu-

tive level of is accounted for by augmenting (using a conjunction) an initially empty conditional expression by the condition found at the currently examined *Ko* node. The follower nodes of a branch node are processes in a sequential fashion, whereupon each non-*Ko* node generates a policy whose condition contains the conjunction of conditional expressions that have accumulated to the depth where the non-*Ko* node is encountered. Since the levels of nesting are finite, the recursion will terminate when the last *Ko* node in the chain (determined by having only non-*Ko* followers) is reached.

The matching and translation of patterns as discussed in this section accounts for the process fragments that correspond to a pattern from the collection presented in Section 4.5. The next section offers a remedy for the cases where fragments are encountered that cannot be matched.

## 4.7. The generating system

The ability to decompose a process completely, so that no remainder ensues, is of utmost importance to the translation scheme proposed in this work, as noted in Section 4.5.4. Principally, the translation is to be performed based on the patterns presented in Section 4.5. This section is to present a means of dealing with process fragments where the pattern-based approach fails.

**Definition of completeness** Completeness of the translation can be attained for a process if

1. A process is decomposed completely into fragments
2. For every fragment, there is a set of parametrised policy rules
  - a) that correspond to the fragment and
  - b) that allow control flow to proceed as in the original process specification.

A sufficient means to formally ascertain that an input process specification can be decomposed and translated in its totality is to use the *generating system* for processes as a starting point for decomposition and subsequent translation.

The sought generating system contains the smallest imaginable process fragments, together with the structural information necessary to

link these fragments together. Also, every fragment must be translatable into management policy rules, while conserving the information present in the process specification.

Hence, a constructive approach to the generating system for process specifications according to our meta-model is to isolate each process node present in the original specification, and to connect it to the rest of the process by means of event-signal pairs.

### 4.7.1. Elements and transformations

The development of the generating system is guided by the principle of isolating each and every node present in a process specification and re-connecting that node by means of virtual events and signals to the original process graph. In effect, this method has already been demonstrated in connection with formulating substitution rules in Section 4.3. However, the substitution rules were intended for the decomposition of the process in a manner that allows larger fragments to be created, corresponding to the patterns described in Section 4.5.

The major difference when developing the generating system is that the same method is applied ruthlessly. Hence, all transformations performed on a process partition are context free, taking into account only the type of a node, and the number of incoming and outgoing links. The comprehensive set of elements of the generating system are shown in Figure 4.11, each element of the generating system in a numbered grey square of its own. The event and signal nodes in the figure are drawn as real (as opposed to virtual) nodes. Nevertheless, they do represent both real and virtual events/signals.

**1 Conditional branching** A conditional branching node encountered in the process specification is placed between one event node, and a number of signal nodes corresponding to the number of conditional branches available at the node. Thus, if an opening conditional node follows a node  $X$  and exhibits  $N$  possible conditional branches  $Y_1$  through  $Y_N$ , the preceding node,  $X$  is appended a signal node  $Si_0$ , and each of the nodes following the conditional branches are prepended event nodes  $Ev_1$  through  $Ev_N$ . The connections to and from the opening conditional node  $Ko$  are severed, and replaced by an event node  $Ev_0$  in the former place of  $X$  and  $N$  signal nodes  $Si_1$



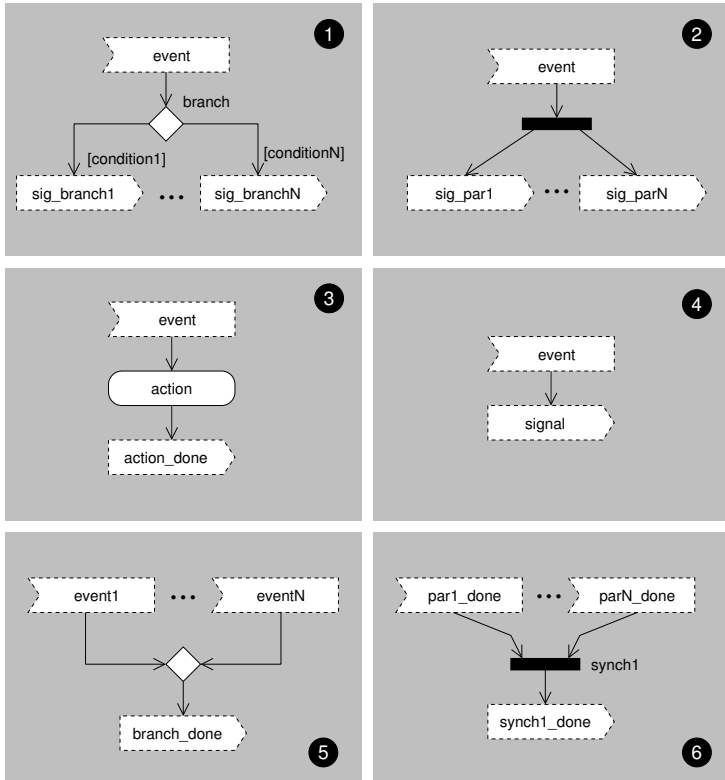


Figure 4.11.: The generating system for process graphs

through  $S_{i_N}$  in lieu of the respective  $Y_i$ . Hence, the transformation can be noted as:

$$\begin{aligned} \{X\} &\rightarrow Ko \rightarrow \{Y_1\} \cdots \{Y_N\} \\ &\quad \mapsto X \rightarrow Si_0 \\ \oplus Ev_0 &\rightarrow Ko \rightarrow (Si_1, \cdots, Si_N) \\ \oplus (Ev_1 \rightarrow Y_1), \cdots, (Ev_N \rightarrow Y_N) & \end{aligned}$$

As shown in the box marked “1” in Figure 4.11, the result is always of the form

$$Ev \rightarrow Ko \rightarrow (Si_1, \cdots, Si_N)$$

**2** **Parallelisation** The form required for parallelisation nodes (opening synch bars) is analogous to that of the conditional branching node:

$$Ev \rightarrow Pa \rightarrow (Si_1, \cdots, Si_N)$$

In the same manner, it is achieved by ensuring that the node is preceded by a single event node and followed by a number of signal nodes (rectangle marked “2” in the Figure):

$$\begin{aligned} \{X\} &\rightarrow Pa \rightarrow \{Y_1\} \cdots \{Y_N\} \\ &\quad \mapsto X \rightarrow Si_0 \\ \oplus Ev_0 &\rightarrow Pa \rightarrow (Si_1, \cdots, Si_N) \\ \oplus (Ev_1 \rightarrow Y_1), \cdots, (Ev_N \rightarrow Y_N) & \end{aligned}$$

**3** **Action** Action nodes are enclosed within an event-signal-pair (shown in the rectangle marked “3”):

$$Ev \rightarrow Ac \rightarrow Si$$

To achieve this, all nodes  $X_1$  through  $X_M$  preceding the action node  $Ac$  are appended signal nodes  $Si_X$ . The nodes  $Y_1$  through  $Y_N$  following the action node are prepended event nodes  $Ev_Y$ . Finally, the action node itself is prepended one event node  $Ev_X$  and appended one signal node  $Si_Y$ :

$$\begin{aligned}
& (X_1, \dots, X_M) \rightarrow Ac \rightarrow (Y_1, \dots, Y_N) \\
& \mapsto (X_1 \rightarrow Si_X), \dots, (X_M \rightarrow Si_X) \\
& \quad \oplus Ev_X \rightarrow Ac \rightarrow Si_Y \\
& \quad \oplus (Si_Y \rightarrow Y_1), \dots, (Si_Y \rightarrow Y_N)
\end{aligned}$$

④ **Signal and Event** Event nodes signifying the expectance of a message, as well as signal nodes representing the transmission of a message may be found embedded at odd positions within the process specification. This element functions as a complement for orphaned event or signal nodes; it substitutes a virtual event in the case of a single signal node; conversely, it substitutes a signal node in the case of a isolated event node. Hence, if a fragment

$$Si \rightarrow (Y_1, \dots, Y_N)$$

is encountered, it can be substituted by

$$Ev \rightarrow Si \rightarrow (Y_1, \dots, Y_N)$$

to ensure accessibility of the fragment based on event reception. In the same manner, if a fragment

$$(X_1, \dots, X_M) \rightarrow Ev$$

is encountered, addition of a signal node ensures proper termination of the fragment:

$$(X_1, \dots, X_M) \rightarrow Ev \rightarrow Si$$

⑤ **Joining conditional** Joining conditional elements are the “closing” correspondents of conditional branching elements. They are of the form:

$$(Ev_1, \dots, Ev_M) \rightarrow Ko \rightarrow Si$$

⑥ **Synchronisation** Synchronisation elements map a number of event nodes  $E_i$  onto a single signal node. The  $E_i$  may be natural events or, if necessary, virtual event nodes.

$$(Ev_1, \dots, Ev_M) \rightarrow Pa \rightarrow Si$$

### 4.7.2. Demonstration of totality

It is possible to prove that the graphs described in Section 4.7.1 and depicted in Figure 4.11 are sufficient to decompose any process specification adhering to the meta-model described in Section 4.2.

It is shown in the following, that a meta-model-compliant process partition that is not decomposable into the elements of the generating system becomes empty once all the transformations described in the previous section have been performed and all the contiguous partitions corresponding to an element of the generating system are removed.

**Proof** Let  $G$  represent a contiguous, non-empty process graph that cannot be generated by the generating system. Let  $G$  be compliant to the meta-model described in Section 4.2, thus containing nodes of the types  $Ko$ ,  $Pa$ ,  $Ac$ ,  $Ev$  and  $Si$  connected in an arbitrary manner compliant to the meta-model.

Let  $Ko_i$  denote all conditional nodes,  $Pa_i$  denote all parallelisation/synchronisation nodes,  $A_i$  denote all action nodes,  $Ev_i$  denote all event nodes and  $Si_i$  denote all signal nodes in  $G$ .

Without loss of generality, let all  $Ko_i$ ,  $Pa_i$ ,  $A_i$  and  $Si_i$  have at least one preceding node. (In the contrary case, the nodes would not be reached by process flow.)

Application of substitution rule R2 ensures that all conditional/parallelisation nodes are either opening, closing, or indeterminate (i.e. neither having multiple incoming, nor multiple outgoing links).

Application of the rules pertaining to elements ① (if opening) or ⑤ (if closing) to every  $Ko_j$  ensures that it is embedded in a form compliant with that respective element. The resulting fragments containing the  $Ko_j$  can be removed from  $G$ , and  $G$  may be partitioned in the process.  $G$  now contains zero  $Ko$  nodes.

Application of the rules pertaining to elements ② (if opening) or ⑥ (if closing) to nodes  $Pa_j$  is performed in analogue manner.

Application of the rules pertaining to element ③ allows removal of all action nodes from  $G$ .  $G$  now contains at most nodes of the types  $Si$  and  $Ev$ .

Partitions of the form  $Ev \rightarrow Si$  match element ④ and can thus be eliminated.

At this time, only elements of the form  $Si \rightarrow Si$ ,  $Ev \rightarrow Ev$ ,  $Si \rightarrow Ev$ , and isolated  $Ev$  or  $Si$  nodes can remain in the process graph.

Application of the substitution given in the context of ④ yields:

$$\begin{array}{l}
 Si_1 \rightarrow Si_2 \\
 \longmapsto Ev_v \rightarrow Si_1 \oplus Ev_1 \rightarrow Si_2 \\
 Ev_1 \rightarrow Ev_2 \\
 \longmapsto Ev_1 \rightarrow Si_2 \oplus Ev_2 \\
 Si_1 \rightarrow Ev_1 \\
 \longmapsto Ev_v \rightarrow Si_1 \oplus Ev_1
 \end{array}$$

This allows elimination of  $Si$  and  $Ev$  nodes, leaving  $G$  empty. If  $G$ , as assumed, cannot be generated by the generating system, a remainder should have been left.  $\square$

The generating system for meta-model compliant processes offers a means to decompose particularly intractable process fragments. It does, however, incur a penalty due to the fine grained decomposition (one actual process node per fragment in the worst case). An alternative approach to dealing with non-matchable fragments is the introduction of additional patterns. In addition to the remedy use-case of this procedure, it can be employed to optimise the translation result with regard to the number of policies generated and the number of artificial events transmitted, as shown in the following section.

## 4.8. Extending the pattern catalogue

In essence, we can differentiate between two kinds of extensions to the pattern catalogue: on one hand the creation of compound patterns for the sake of optimisation, and on the other hand the invention of completely new patterns. Both cases share a common methodology, if not a common goal. This section describes the substitution of pattern sets with single, complex patterns, as well the steps required to provide a completely new pattern.

### 4.8.1. Pattern substitution

In the following, a number of commonly encountered process fragments are analysed. These fragments are chosen so that they do not match any of the patterns. To “make them fit”, we apply one or more of the substitution rules presented in Section 4.3. A common trait of the substitution rules is that they structurally decompose the process graph they are applied to. In contrast, none of the rules ties process elements closer together than they were in the original specification. Hence, application of these rules may result in a partitioning of the process graph. Matching of the patterns can thus be attempted again in each of the partitions.

Keeping in mind that the substitution rules do not actually change the control flow in the process, a comparison with successful pattern matches before and after application of the rules can aid in optimising the translation procedure. This can be achieved in either one of these cases:

1. If pattern matching is successful and total (i.e. no process nodes are left unmatched) after application of the substitution, a *complex pattern* has been found. In this case, the set of policies generated from the substituted form can be generated directly from the original process fragment. If the new complex pattern is recorded, it can be added to the vocabulary of the translation tool. Moreover, the generated set of policies can possibly be optimised (by hand) before the optimised set can be recorded as a suitable translation for the complex pattern.
2. Depending on the capabilities of the target (policy) language, more advanced language constructs can be chosen for the translation of the original process fragment.

Figure 4.12 shows a process fragment consisting of a sequence of actions. This case is not covered by the fundamental pattern set. However, the target language for management policy may support a sequence of actions to be specified in a single policy. This would reduce the process fragment to a structure similar to the *basic pattern*: a management policy rule could be generated that unconditionally executes the three actions in the given sequence upon detection of the event.

In contrast, applying the *action decoupling* substitution rule will generate *several* instances of the basic pattern from the process fragment

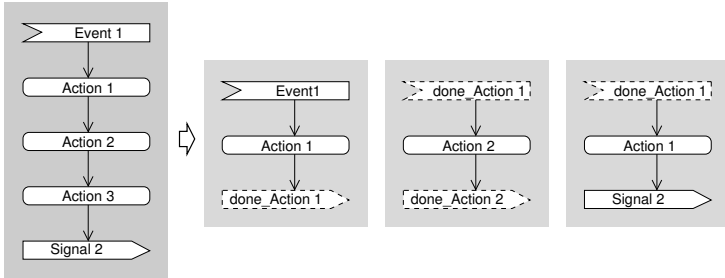


Figure 4.12.: Exemplary complex pattern

(those shown in the right part of the diagram). Translation of the resulting process fragments would create three policy rules and require the introduction of a number of virtual event and signal nodes.

Hence, the resulting policy sets will differ depending on whether the capability to specify several action instances in one policy rule is taken into account.

#### 4.8.2. Pattern extension mechanism

The methodology that may be employed to introduce novel or compound patterns into the pattern catalogue is the same one that was used in creating the collection of fundamental patterns described in this work. The following gives a summary of the required steps:

**Provide graph form.** The candidate pattern should be modelled as a process graph.

**Ensure compliance to target meta-model.** To allow effective use of the translation resources described in this work, the candidate pattern should be contained within the target meta-model.

**Provide proposed translation.** A set of parametrised management policy rules must be attached to the pattern.

**Define API extension (if required).** Patterns translations that need to keep state may require the existence of auxiliary services. Examples where such services are required include the synchronisation service that is nec-

essary for the effective translation of the synchronisation pattern.

**Example** The steps described above can be illustrated by means of the compound pattern shown in Figure 4.12. A visual form of the pattern is provided in the left side of the diagram. Formally, it can be expressed as:

$$Ev \rightarrow Ac_1 \rightarrow \dots \rightarrow Ac_N \rightarrow Si$$

The pattern complies to the target meta-model since it incorporates only the node types specified there, and these nodes are associated in a manner that does not violate the cardinalities given by the model. An appropriate translation, given a target language that supports several sequential actions, can be expressed as:

```
policy  
{  
    event { Ev }  
    action { Ac1 ; ... AcN ; Si ; }  
}
```

The new pattern imposes no requirements on a runtime API; hence, the last step of the mechanism finds no application.

## 4.9. Translation example

This section illustrates the mechanisms presented in the preceding sections by means of an example. Specifically, a formal process definition is treated according to the methodology detailed in Section 4.4, including application of the substitution rules, identification of patterns and generation of a policy rule set.

The example has been chosen to satisfy several requirements: it should have been created as a formalisation of a real-life process, and it should possess a level of complexity sufficient to illustrate most of the concepts treated in this chapter. In addition, it should be of manageable size in order to be both graphically compact and describe a self-contained process part. The chosen example is from [Clau 06a] and shows a part of the change management process according to ITIL. In order to constrain the size of the process specification, a general, non-specialised process specification is used, shown in Figure 4.13.



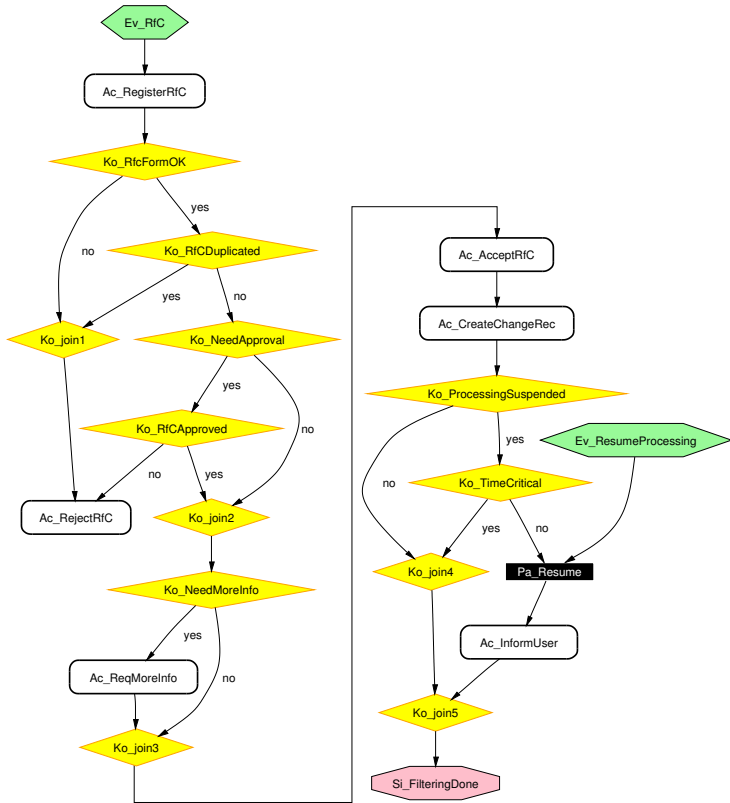


Figure 4.13.: Handling of a Request for Change (generated)

The diagram depicts the general procedure to be used at the beginning of a Change Management process instance, at the point in time when a Request for Change has been issued. The figure shows the process before any action has been taken; it can be likened to an original, “freshly modelled” process that complies to the generic meta-model.

In the following, the methodology developed in this work is applied to this process specification, and the intermediate results are discussed. Note that the diagrams shown in this section diverge from the UML activity diagram form that is otherwise used throughout the thesis. This is due to their having been generated by the translator prototype that was created in the course of this work. As there are no semantic, but only visual differences between the node representations, the mapping is given in Table 4.15.

All nodes are prefixed with their type, e.g. an action named “Register RFC” is prefixed by *Ac\_*. In addition, virtual nodes are prefixed with a “*v*”, and all nodes that are generated, i.e. not present in the original process carry suffixes to the node’s name (these can be ignored for the purposes of this example). The nodes’ graphical representation is consistent and kept close to the UML wherever technically convenient.

#### 4.9.1. Application of the substitution rules

The net effect on the process specification after having applied the substitution rules is strongly dependent on the structure of the original process specification. In our example, the context-free rules can be applied, but leave the process unchanged: there are neither *m:n* conditional or parallelisation nodes, nor are there any *1:1* nodes without guard conditions.

The context-sensitive rules, on the other hand, do effectively change the process. The branch joining nodes (denoted *Ko\_joinX* in the figure) are eliminated and signals are appended to the incoming links. The action nodes are decoupled, as are the combinations of branch and synchronisation nodes. These transformations yields process fragments that are shown in Figure 4.14.

The application of the rules has the effect of breaking up loops within the process graph. The resulting fragments are structured as trees. The root of every tree is an event node (real or virtual) with only

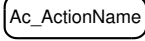
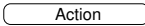



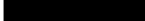

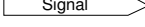
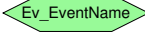
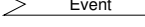
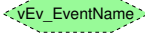
<i>Element</i>	<i>Shape</i>	<i>UML shape</i>	<i>Description</i>
Action			Actions are shown in rounded rectangles
Condition			Conditions are shown in diamond-shape.
Synchbar			Synchbars are UML-like, except that they carry a label.
Signal			Signals are octagons.
Event			Events are hexagons.
Virtual node		N/A	Virtual nodes are bordered by a dotted line; in contrast “normal” nodes are bordered by a solid line.

Table 4.15.: Comparison of node symbols

one child node, while leaf nodes are signal nodes. Exceptions are found in closing constructs, such as that of synchronisation.

### 4.9.2. Identifying patterns

The fragments resulting from the previous step correspond to the patterns described in Section 4.5. The resulting fragments for our Request for Change example can now be matched against the pattern catalogue using the following guidelines:

- Any fragment containing at least one branching node will match one of the condition patterns.
- Any fragment containing a synch bar will match one side of the synchronisation pattern. If the synch bar has more incoming than outgoing links, then the construct is closing (synchronising); if the opposite is the case, the construct is opening (parallelising). The context-free substitution rules ensure that all other possibilities have already been eliminated.
- If a fragment contains neither a branching node, nor a synch bar it is either a basic pattern or an anomaly.

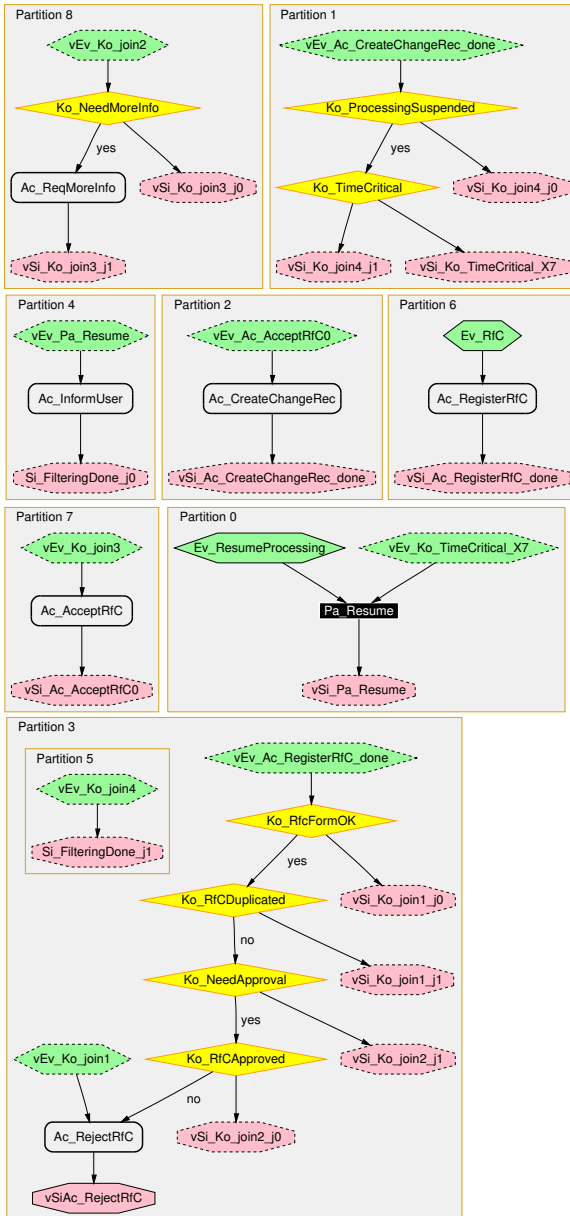


Figure 4.14.: Process graph after decomposition (generated)

### 4.9.3. Translation result

The final step in this example is the derivation of management policy rules from the process fragments created in the preceding steps. In the following, the policy set derived from each partition shown in Figure 4.13 is documented.

**Basic patterns** Partitions 2, 4, 6 and 7 are matched by the basic pattern. The policy rules shown in the following carry the partition they were generated from in a **related Process** attribute.

<pre> <b>policy</b> {     relatedProcs = "Part. 4"     <b>event</b> { vEv_Pa_Resume }     <b>action</b> { Ac_InformUser;     Si_FilteringDone } } </pre>	<pre> <b>policy</b> {     relatedProcs = "Part. 2"     <b>event</b> { vEv_Ac_AcceptRfc }     <b>action</b> {     Ac_CreateChangeRec; } } </pre>
<pre> <b>policy</b> {     relatedProcs = "Part. 6"     <b>event</b> { Ev_Rfc }     <b>action</b> { Ac_RegisterRfC } } </pre>	<pre> <b>policy</b> {     relatedProcs = "Part. 7"     <b>event</b> { vEv_Ko_join3 }     <b>action</b> { Ac_AcceptRfC } } </pre>

**Condition patterns** Partitions 1, 3 and 8 represent conditional statements. They are matched by the basic condition pattern (Partition 8) and the conjunctive condition pattern (Partitions 1 and 3).

Partition 8 can be translated into two policy rules:

```

policy
{
    relatedProcs = "Part. 8"
    event { vEv_Ko_join2 }
    action { vSi_Ko_join3 }
    condition { NOT Ko_NeedMoreInfo }
}

```

## Chapter 4. Process translation

```
policy
{
    relatedProcs = " Part. 8"
    event { vEv_Ko_join2 }
    action { Ac_ReqMoreInfo; vSi_Ko_join3 }
}
condition { Ko_NeedMoreInfo }
```

Partition 1 yields:

```
policy
{
    relatedProcs = " Part. 1"
    event { vEv_Ac_CreateChangeRec_done }
    action { vSi_Ko_join4 }
}
condition { Ko_ProcessingSuspended AND Ko_TimeCritical }
```

```
policy
{
    relatedProcs = " Part. 1"
    event { vEv_Ac_CreateChangeRec_done }
    action { vSi_Ko_TimeCritical }
}
condition { Ko_ProcessingSuspended AND NOT Ko_TimeCritical }
```

```
policy
{
    relatedProcs = " Part. 1"
    event { vEv_Ac_CreateChangeRec_done }
    action { vSi_Ko_join4 }
}
condition { NOT Ko_ProcessingSuspended }
```

Partition 3 is the most complex of the set. It translates to:

```
policy
{
    relatedProcs = " Part. 3"
    event { vEv_Ac_RegisterRfC_done }
    action { Ac_RejectRfC }
}
condition { Ko_RfcFormOK AND NOT Ko_RfCDuplicated AND
Ko_NeedApproval AND NOT Ko_RfCApproved }
```

```

policy
{
    relatedProcs = "Part. 3"
    event { vEv_Ac_RegisterRfC_done }
    action { vSi_Ko_join1 }
}
condition { NOT Ko_RfcFormOK }

policy
{
    relatedProcs = "Part. 3"
    event { vEv_Ac_RegisterRfC_done }
    action { vSi_Ko_join1 }
}
condition { Ko_RfcFormOK AND Ko_RfCDuplicated }

policy
{
    relatedProcs = "Part. 3"
    event { vEv_Ac_RegisterRfC_done }
    action { vSi_Ko_join2 }
}
condition { Ko_RfcFormOK AND NOT Ko_RfCDuplicated AND
NOT Ko_NeedApproval }

policy
{
    relatedProcs = "Part. 3"
    event { vEv_Ac_RegisterRfC_done }
    action { vSi_Ko_join2 }
}
condition { Ko_RfcFormOK AND NOT Ko_RfCDuplicated AND
Ko_NeedApproval AND Ko_RfcApproved }

policy
{
    relatedProcs = "Part. 3"
    event { vEv_Ko_join1 }
}
action { Ac_RejectRfC }

```

**Synchronisation** Partition 0 contains a closing synch node. It specifies waiting until two messages have been transmitted, before execution is resumed in Partition 4. Please refer to Chapter 5 with regard to the correlation of message reception.

#### 4.9.4. Optimisation

The elementary patterns described in this chapter ensure translation of the process specification, while disregarding the specific features of a target policy language. The number of policies resulting from translation can be reduced if more advanced constructs of policy languages (as described in Section 4.1.6) are taken into account.

Another approach to optimising the result set is to apply simple heuristics to the (already generated) set. For example:

Any two policies that execute the same action (with the same parameters!) can be combined into one by

- joining their condition clauses with an OR operator
- joining their event sets

Observations regarding the domain of virtual events/signals could also lead to a more compact representation of processes.

Any such optimisation strategy should take into account that, while a smaller result set could reduce size and complexity, it may at the same time impede effective association of policy rules with different domains or tools.

#### 4.10. Summary

This chapter has presented the core approach to process translation. It focused on the treatment of control flow in IT management processes. Analysis of formalisms for the representation of management processes, as well as of management policy has determined their respective expressive capabilities with respect to the requirements of technical IT management. The analysis has yielded a candidate set among the process languages, as well as a candidate set among the target policy languages.

A high-level, six-step methodology that describes the translation procedure has been devised. The actual translation is based on transition between two meta-models: a generic one that describes the candidate set of process languages, and a target meta-model that represents a set of patterns for process fragments. Transition between the two models is achieved by applying a number of substitution rules to the source process definition.



An catalogue of fundamental patterns has been described, that can be matched against the result of the substitution. Each pattern carries a parametrised translation template, that describes the set of management policy rules that can be generated from an instance of that pattern. Thus, every pattern match results in a number of policies that can be instantiated from the matching pattern; the sum of all policies created in this manner are the result of the translation procedure.

In order to allow the accommodation of optimised or even novel patterns, a mechanism for the extension of the pattern catalogue has been described. Both the existing pattern catalogue, and the extension mechanism have been specified independently of specific features of source or target languages.

Focused on translation of the process data flow, this chapter has disregarded the data flow within the process. One of the effects when applying the given substitution rules to source processes is the fragmentation of the source process, thereby interrupting the direct connection that may carry data between process parts. The following chapter will analyse the data flow requirements of such fragmented process specifications and propose suitable solutions.



# Chapter 5

## Process data flow

THE mechanism for control flow introduced in the previous chapter presents challenges regarding the data flow to, from and between processes as well as the interaction with different data storage facilities. Processes are data-driven in as much as they may be event driven. The decomposition of a process specification into patterns sized “chunks” will disrupt the information flow at the chunks’ borders. This makes necessary a strategy to incorporate the flow of information pertaining to the original process specification into the policy-based realisation of the process.

Disruption of data flow due to process decomposition

To tackle these issues, we examine the way information is attached to nodes in a process. This examination’s results allow us to determine how information items should be attached to process fragments being matched to pattern, and to derive requirements on the transport of these information items (Section 5.2). Section 5.3 describes a means for the specification of data flow and proposes an architecture suitable for the realisation of information transport according to those requirements.

Processes are supplied with information from different sources (e.g. documents, messages, user input). Similarly, they output information to a variety of information sinks. Within a process, the output of one process part (e.g. an activity) is frequently used as the input of a subsequent one. If a centralised form of process execution can be assumed, this intra-process communication won’t presents difficulties. It may be realised by a central workflow management engine. In the opposite case, when distributed control of the process must be taken into account, the integrity of the data flow within the process must be guaranteed by other means.

The approach proposed in this work goes beyond mere distributed execution of the process: the process specification is transformed,

Preservation of data flow

certain process nodes are eliminated while others are introduced, and the resulting fragments are intended to be executed separately from each other. The transformations may effectively alter the specification of process data flow itself. The nodes that will in fact process information within the process (action, signal and condition) are preserved, the path by which the information is routed to its destination is altered or interrupted. Figure 5.1 illustrates how the partitioning of a process can interrupt the data path at pattern boundaries.

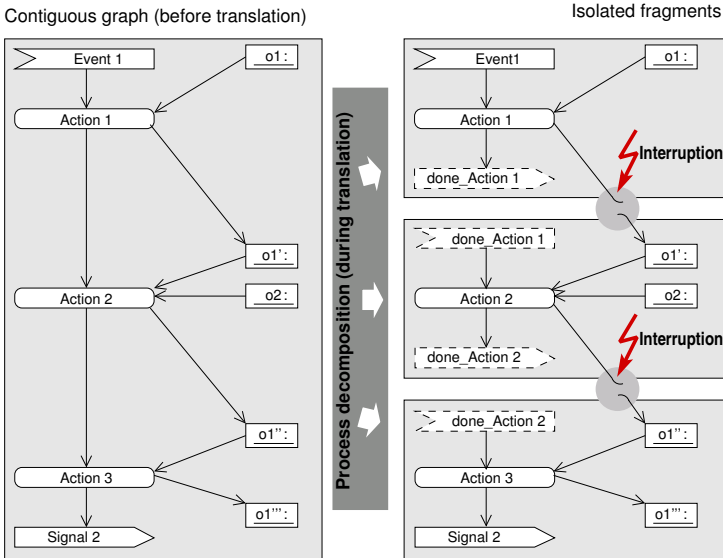


Figure 5.1.: Information flow  
Information flow before and after control flow decomposition

### 5.1. Preservation of the information flow

An IT management process—or any process, for that matter—cannot function properly if the data flow between its activities is disturbed. Therefore, it is of critical importance to devise a mechanism that maintains the flow of information even in a process’s decomposed form. Two aspects need to be taken into account in constructing this mechanism. First, the nature of the data items associated with

a process will indicate some of the capabilities necessary to a mechanism for process data flow. Second, the nature of the intended distribution will provide a frame for the requirements on transport capabilities.

### 5.1.1. **Data/information items in processes**

To devise a mechanism for the preservation of intra-process information flow, it is necessary to identify the kinds of data items that can be associated with a process, or process fragment. The information that is necessary to a certain process partition is associated with that partition in different roles.

**Heuristic to classify data items** Several process node types may be associated with data items. The following distinctions were instrumental in determining how data is attached to nodes.

**Association type.** Information associated with a process node can be attached to nodes in different manner. It can be made available as the output of a preceding node (e.g. the output of an action may serve as part of the input to a following action); it can be associated with the node as a process-external data item (e.g. as a reference to a process artefact); and it can be made available to the node as data originating in the runtime system executing the process (e.g. current time and date).

**Input and output.** Some nodes types are associated with input items, as well as with output items. Others may only receive input, or only generate output.

**Immanence.** Some information items are strictly part of the process. They are created within the process, and they find their use within its execution. Others are external to the process. They may originate in management information systems, or they may constitute input from an interaction with an administrator.

**Transience.** Information items may be persistent (e.g. information in a database relation) or fleeting (e.g. an SNMP

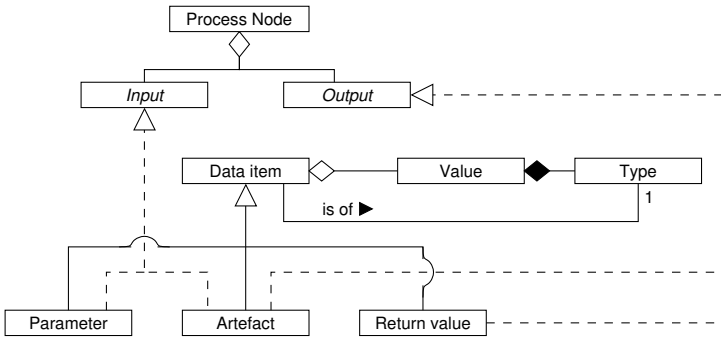


Figure 5.2.: Abstract view on process data

trap message). Both kinds may play a role in the execution of a process, but must be treated differently. Transience concerns the availability of an information item, as well as its validity. For example, the validity of a “current time” value may suffer if considerable time passes between its assignment and its use.

For any given node type, data items of different kinds may be associated with it in different roles. Figure 5.2 illustrates in principle some of the relationships between a process node and the associated data items. Process nodes may input and/or output data items, e.g. parameters/return values or process artefacts. In the following, a narrower, node-specific view of these relationships is described. It serves as a starting point for the development of a mechanism for linking process patterns and information items.

**Action** Naturally, actions constitute the process elements that exhibit the greatest number and variety of relationships to data items (Figure 5.3). The most obvious example are the values to formal parameters of actions. Often, actions operate on data that constitutes, or is part of, a process artefact. Artefacts may be stored in a management information system (e.g. in a CMDB). A reference to such an information item may be made available to an action as a formal parameter. Actions may yield return values that can be of relevance at another location in the process.

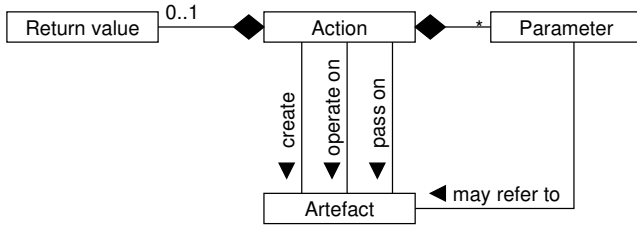


Figure 5.3.: Data items of an action node

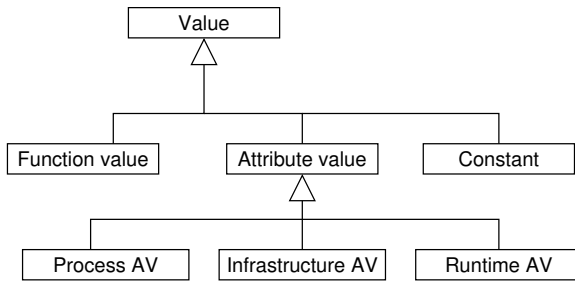


Figure 5.4.: Process data values

**Conditional branch** The evaluation of conditional expressions determines the control flow path chosen in a process instance. The expressions can refer to different data items inside, as well as outside of the process. Examples include attribute values acquired from the infrastructure (e.g. values of attribute in a MIB), data originating in or constituting an process artefacts, variable values pertaining to the runtime system (e.g. current time/date), and return values of functions/methods.

Values employed in conditional expressions (as well as in other nodes) may originate in different locations, as suggested in Figure 5.4. In turn, their origin determines the manner in which the values are extracted for use in the execution of the process.

The information needed to allow evaluation of the conditional expression is used solely in that context. The expressions evaluate to *true* or *false*, but do not pass through data to other process nodes.

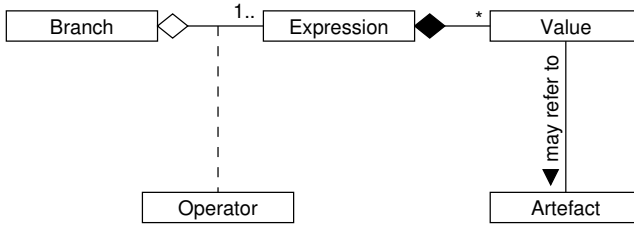


Figure 5.5.: Data items of a conditional branch node

Normally, they do not change the state of the infrastructure<sup>1</sup>, nor do they output data, as may happen in the case of action nodes.

**Parallelisation/synchronisation points** Information cannot be attached to parallelisation or synchronisation nodes at the time of formal process specification. There may be information attached, but it is not used by the process itself. Instead, it serves to maintain the execution of the process, by being instrumental to the proper parallelisation/synchronisation action specified by the process node in question. As such, it does not constitute a part of the process data flow per se; it can be seen as signalling performed for the benefit of the process support utilities. This signalling is implementation specific (the implementation determines what information is necessary to effectively perform the tasks of these nodes) and will therefore not be discussed further.

**Signals and Events** Signal and event nodes cater to different views on the same abstract process element: the message. Signals specify that the data should be transmitted at some point in the process; conversely, events specify where it should be received. Obviously, the relevance of these two classes of nodes to the data flow in processes is very high. In the following we will focus on signal nodes, as they are the active (i.e. transmitting) elements. In contrast, event (expectation) nodes symbolise the passive notion of expecting data to be delivered, along with the specification regarding *which* data will be expected—and thus available to adjacent nodes (e.g. actions).

<sup>1</sup>Changes to infrastructure state may occur as side effects, if values in a conditional expression are obtained as return values from functions.



## 5.1. Preservation of the information flow

What is still missing in the specification of signals is “which” data should be transmitted, and where it should be procured for that purpose. Similarly to actions and conditional, signals can be associated with the entities depicted in Figure 5.4. As noted at the begin of this section, information relevant to the process can originate within or outside the process itself. In both cases, it is important to ensure that the signal element carrying the information is matched by corresponding event nodes. The difference in most cases will be that, while information created and used within the process will be bound to *virtual* nodes, the information items that originate outside the process, or those being delivered outside the process will be bound to *natural* event or signal nodes. The match between signal and event nodes can be realised by using a *type identifier* for these nodes, which provides a non-ambiguous means to map the nodes to the information items they transmit or receive. In principle, if an information item necessary to a process fragment is thought of as a typed object of a class, the type identifier would be the name (type) of that class of objects. The transport of the thus marked information items will favourably be enacted by means of a messaging system, as addressed shortly in Section 5.2.

Origin of data

Type identifier

In practice, a process fragment may necessitate more than one information item. Thus, that the type identifiers employed to mark event and signal nodes should accommodate the typing of *aggregations* of information items—records or data structures.

Aggregation of data

In view of the issues induced by the fragmentation of process graphs, the one most important issue to solve is the *replication* of the original data flow, as it may be implicit in pre-translation process specifications. Most breaks in the graph result in the creation of virtual signal/event pairs (see Section 4.3). Thus, to maintain the data flow between two nodes separated into two different fragments, the information intended to cross a connection between these two nodes needs to be charged to the virtual nodes that represent it.

Intra-process data transport

Consider the left side of Figure 5.6, which is in fact a simplified version of the example in Figure 5.1. The decomposition of the process graph into the two (upper and lower) fragments results in the disruption of the flow between Action 1 and Action 2, as already discussed in the introduction of this chapter. As a result, the information item that is produced by Action 1 cannot be made available to Action 2. By relaying the information item to the virtual signal following Action 1, it will be transmitted and received at the point

Charging virtual nodes

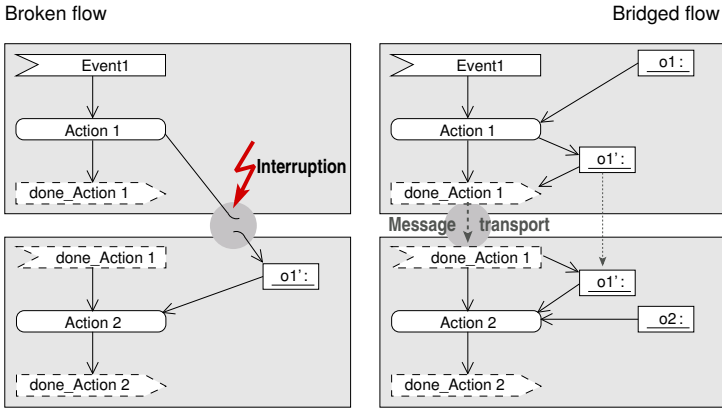


Figure 5.6.: Bridging the gap between fragments

in the process given by the corresponding event, i.e. the event node that is marked with the same type as the signal.

In this example, the types match, and the information item in question can be made available to Action 2. This match is however not accidental, or incidental to the example: it is a consequence of the application of the substitution rules in the first place! Hence, to maintain a proper intra-process data flow in the locations where it might be disrupted by the decomposition of the process, it suffices to

- identify the data items that should be transported, and
- link them to the pair of virtual signal/event nodes at the time of its creation.

Hence, staying within the concept of projecting the intra-process flow of information onto a sequence of signals being transmitted (and corresponding event nodes reacting to those signals), we can conclude that the linkage of information items to virtual nodes must occur at the time when the substitution rules are applied. The practical approach to this task is discussed in the next section, before examining (in Section 5.2) how the information transport can be enacted.

Linking information at translation time

### 5.1.2. Attaching information specification to patterns

To automate the handling of data flow in the course of the translation procedure, it is necessary to identify those information items that are provided by process fragments, and those that are required in (other) process fragments. By associating instructions regarding the preservation of data flow to the patterns described in Section 4.5, formal specifications for the flow of information within and between process fragments can be extracted.

Among the node types occurring in a process graph, action, condition and signal nodes are in need of information. Conversely, action and event nodes provide information to the process.

Context of information

Modification of information during process execution is solely a responsibility of action nodes, as they are the only ones effectively creating or processing data relevant to the process. Actions may create information, modify existing information, or pass through information (e.g. to point to a data item determined to be important for processing in later actions). Data may be stored outside the process, e.g. in a repository for management information. In this case, a reference to a specific data item may be viewed as “information” as it conveys the address of relevant data to a (subsequent) action.

In order to minimise the scope of the re-linking of information at translation time it is important to differentiate between information items that are passed along within the process—and thus are affected by the its fragmentation, and items that are not directly affected by fragmentation due to their being represented by external references in the first place.

Minimal invasiveness

Not all data items necessary to process execution constitute information items that are relevant to the process itself. Some of them are merely instrumental in securing the correct execution of the process, an obvious example being the transmission of virtual signals to trigger execution of detached process fragments.

Process management information

Four steps must be taken to ensure that the data flow within the process is replicated in accordance to the concept depicted in Figure 5.6.

Procedure

1. Determine information items to be transported.
2. Determine source and target locations.

3. Identify the signal and event nodes to be employed to accommodate the required information.
4. Adapt references to data (e.g. in actions' arguments) to point to the event.

The information to be transported is highly dependent on the management setting, the process specification and thus the selection of information items intended to be “bridged” across fragment boundaries. In practice, it is impossible to predict with some generality which information items need to be treated in the manner outlined above. Therefore, the approach of relaying *all* affected data is the only one guaranteeing a consistent result. Also, it requires the creation of “packets” of information corresponding to the collection of items required in a process fragment. This implies beforehand aggregation of the information items. A solution to specifying aggregation clauses is proposed in Section 5.3, along with architectural considerations regarding its practical realisation. Beforehand, in the next section, we examine the requirements on the transport of information after aggregation has been performed.

Need for  
aggregation

## 5.2. Requirements on information transport

Having determined the binding of information to process nodes, it is forthcoming to examine the means to make available that information during process execution. We have already noted that information relevant to a management process possessed many differentiating characteristics. In this section, we examine in detail the requirements imposed on the realisation of data flow within an IT management process, as well as between such a process and entities outside its scope.

### 5.2.1. Dimensions of process data flow

The flow of information during the execution of an IT management process is determined by many different aspects. These, again, are highly dependent on the kind of process that is being executed, the available tool set, the managed infrastructure, the facilities for information storage and the heterogeneity that can be expected in most infrastructures.

The different dimensions that need to be taken into account when analysing information or data flow in an IT management process are sketched in Figure 5.7. They apply to data being made available to a process during execution, as well as to the information items that may result from the process.

**Tool level** The execution of an IT management process may require several tools of different types. We can distinguish between tools for process support (e.g. a trouble ticket application), management tools (e.g. for network management), as well as tools being part of the infrastructure (e.g. a managed application).

**Container type** Information can be presented in different forms. It can be made available as an attribute value (e.g. by a management agent providing a MIB representation of a system), data records containing a composition of data (e.g. a synopsis generated by a security tool), or complex documents. These different information containers can additionally be differentiated between by the level of formalisation typically employed to encode the information they transport.

**Data type** The data that represents an information item may be of one of several types. The most simple form, and at the same time the most easy to handle, is the primitive data type, e.g. integer or float number, boolean values, character strings etc. Homogeneous aggregations of primitive types (e.g. arrays) again constitute types of their own. Heterogeneous data structures constructed from primitive types also need to be considered separately, as well as sets of such structures.

**Number of data items** A process may perform activities on several series of data items. Also, the execution of an action may require multiple instances of a datum.

**Data source type** The data required for process execution may originate from different kinds of sources. They include repositories for persistent data, such as database systems or directories. Data relevant to a process can also originate in files of different formats,

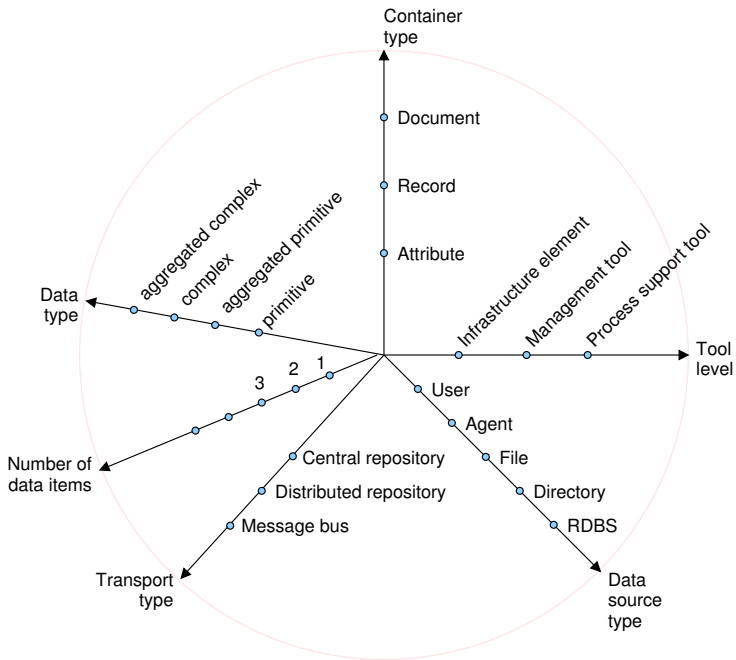


Figure 5.7.: Dimensions of data flow in processes

or it can be retrieved from management agents. Finally, IT management processes will, in most cases, require interaction with an administrator at some point. This interaction will most likely result in data items that are used as input to an activity within the process, or as a base for a branching decision.

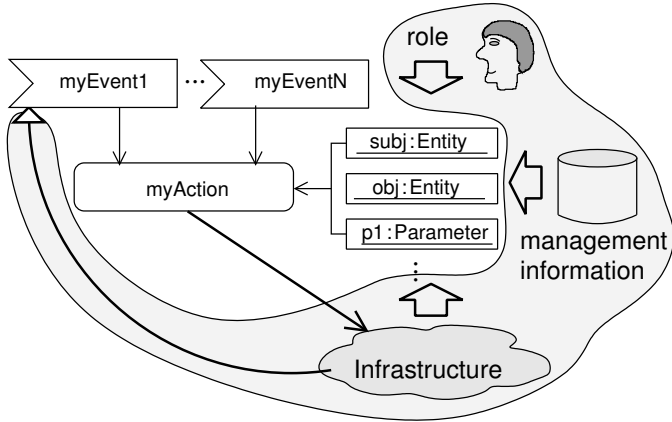


Figure 5.8.: Sources of management process data

**Transport type** Depending on the origin of the data, several methods may be employed to retrieve it and make it available to the process. The most obvious alternative is the existence of a central repository for information. This implies that the information can be accessed at any time by any actor (machine or human) that participates in process execution. This alternative will exhibit a high level of consistency, but also inflexibility and the risk of creating an information bottle-neck or a single-point-of-failure within the infrastructure. In an inter-domain setting, the creation of a central repository may present insurmountable difficulties, when compared to the benefit derived from it.

Centralised

A second option is the concept of distributed (e.g. federated information stores). These have the benefit of scalability and may be located in the domains where the data originates. Research on distributed database systems and directories has made available concepts and means to allow for consistent data storage and retrieval.

Federated

Nevertheless, the initial effort to set up such a federation may be high.

Message based

The third alternative is to use a message bus to transport information items to their place of destination. It has the benefit of being very flexible, but carries a substantial risk to lose information. In principle, an IT management process may require information along any of these dimensions.

### 5.2.2. Policy-based process realisation

Realisation of IT management processes by means of policy rules introduces several constraints to the concept of a process data flow facility. Execution of management policies is event-driven. Hence, asynchronous message passing may be expected to play an important role in the control flow of the process.

It may be distributed, leading to an isolated execution of single rules. The information needed by a specific process node must be provided at the location of a policy rule that has been generated from that node. As the process specification is decomposed into isolated fragments, transfer of execution to another domain may occur at arbitrary positions in a process. Information necessary to a given process fragment must therefore either be available in the target domain, or be transported there whenever needed. Note that information originating in several domains may be required.

In summary, the realisation of the approach presented in this work benefits from

- event-based information transport
- distributed or message-based information provisioning
- inter-domain collection of required information items

The following section discusses the realisation of process data flow, taking into account aggregation requirements. In particular, a facility for the specification of event-based flow is delineated, together with an architectural outline indicating the functional components necessary for its projection onto existing information sources.



### 5.3. Realisation of process data flow

The data transport needs inherent to a process specification must be made available to an aggregator function. This can be achieved by means of a declarative information specification language that allows the formal expression of aggregations. Section 5.3 gives an outline of a suitable specification language.

#### Overview of the Service Information Specification Language

The SISL is a declarative, XML-based language [DgS 07] intended to describe aggregation specification in the domain of service management. As such, it is suited to describe the (simpler) aggregations necessitated by the realisation of data flow in processes.

The basic SISL element is the **aggregation** which enables the user to specify the data values to be collected, the sources they should be collected from, as well as timing parameters with regard to the frequency of sampling values.

Data sources are abstracted by means of a naming scheme that obscures the physical sources in favour of named attributes. SISL includes elements for the description of preprocessing instruction such as the computation of mean values or sums. Guard conditions can be specified that trigger the transmission of a data packet containing the aggregated values. In particular, these configurable triggers are very useful for use with the inherently event-driven, policy-based process execution mechanism.

The SISL has been specified as an XML Schema grammar that is given in Appendix 10. It is implemented in the SMONA architecture presented in Section 7.4, according to the outline in the following section 5.3.

#### Architectural outline

A facility for the aggregation and transport of process data needs to provide the functions sketched in Figure 5.9. It should possess the ability to acquire data from the relevant sources (platform specific layer), to express the data (that may be acquired in different formats) in one agreed-upon format (platform independent layer), and

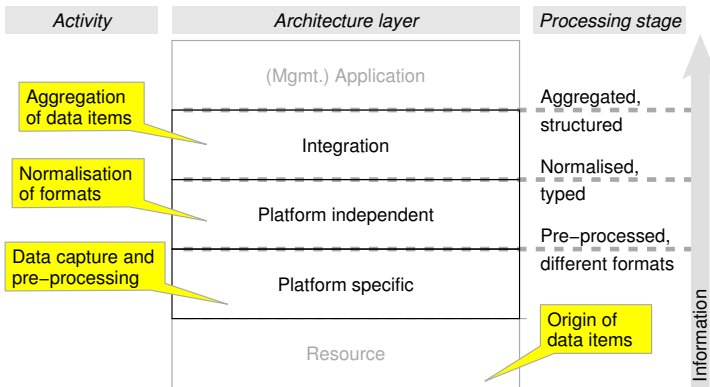


Figure 5.9.: Aggregating architecture (functional view)  
Functional building blocks of an aggregating architecture

aggregate data items from different sources according to pre-defined data structures (integration layer), as needed by a process partition. An aggregating monitoring architecture that implements the three required layers is detailed in Section 7.4.

## 5.4. Summary

Interruption of data flow

Due to the fragmentation of management process specifications during translation into policy sets, the data flow between the—formerly contiguous—fragments may be interrupted. To counter this effect, we have devised a message-based scheme for information transport within processes. Thus, the flow of data between process nodes, as well as between process nodes and external entities can be realised by means of signal/event node pairs that are already present in the original process specification, or introduced as *virtual* nodes at the time of their decomposition into fragments.

Binding of information items

In order to allow precise specifications of the information flow, we have analysed the way in which data items can be bound to process elements. In particular, the preservation of the data flow at the borders of patterns, as described in Section 4.5, has been addressed.

Specifying information items

The formal projection of the information transport needs of a process

can be realised with the Service Information Specification language outlined in Section 5.3. Though developed for the aggregation of information in order to create a service view onto infrastructures, SISL is suited for the (in truth simpler) task of specifying aggregations for use with management processes. The SISL grammar can be found in Appendix 10. The practical realisation of the thus specified data flow can be pursued by means of an architecture that provides configurable data aggregation features, as outlined in Figure 5.9. An implementation of such an architecture is discussed in Section 7.4.

Aggregating  
architecture

The mechanisms for data flow within processes addressed in this chapter, combined with the mechanisms for control flow treated in Chapter 4 provide the fundament for the development of a management architecture that actualises the vision of policy-based execution of IT management processes, as proposed in this thesis. The following Chapter 6 describes the functional components necessary to implement an end-to-end workflow ranging from the specification of IT management processes up until the execution of operational policy rules generated from the process specifications.



# Chapter 6

## Architecture

**S**PECIALISATION and adaptation is implied by all generic approaches in order to provide operable solutions. Therefore, it is important that such approaches provide facilities to perform such specialisation or adaptation.

The mechanisms for process translation and preservation of the process data flow described in the preceding chapters are generic regarding process and policy languages, as well as regarding the management framework or the process/infrastructure management tools used. The latter are specific to the needs and resources of the infrastructure to be managed, while the former two constitute design time choices. From that perspective, the approach is also independent of the information model employed—and thus the data model derived from it—for the description of the managed system and the information transferred during process execution.

This chapter describes an integral architecture supporting the derivation of management policies from process definitions, taking into account the complete life-cycle of a management solution. Therefore, the proposed architecture is based on the functional needs of the management life-cycle phases that the approach relies on.

The architecture developed in Section 6.2 is projected onto a set of concrete tools in Chapter 7. While the boundaries between components are blurred by such a projection, their functional identity remains untouched.

The interactions implied by the life-cycle phases addressed in Section 6.1 are assigned to pairs of functional components to form a collection of inter-component relationships in Section 6.3.

## 6.1. Management process life-cycle

The description of a life-cycle is an instrument that allows the consideration of its object over time. In analogy to the life-cycles described for software components or services, a management method may have a life-cycle of its own—in the domain of process-oriented management we could call it a *meta-process*. The following discussion illustrates the phases in the existence of a process that is realised by means of the approach proposed in this work.

### 6.1.1. Initial workflow

The *initial workflow* can be described to start with requirements from active business processes and to finish with provision of a detailed, operational IT management process. In contrast, the overall management workflow continues until the enforcement of IT management policies on the infrastructure is made possible.

The first part of the workflow describes the creation of detailed operational IT management processes at an operational level. This first part (depicted in Figure 6.1) focuses on the formalisation of IT management procedures into operational, technical processes that take into account the business processes of the organisation, its infrastructure, the distribution of management roles as well as best practices for IT management.

Derive  
requirements

The core business of the organisation determines the demands regarding IT services. In more concrete terms, the organisational (corporate etc) IT must support the business processes that drive the actual core business. In consequence, these business processes are a source of requirements to the management processes controlling the IT infrastructure and services.

Model  
high-level  
process

Another input to the modelling of IT management processes are service management frameworks that have evolved from management experience and describe generic, process-oriented best practices. Guided by a best practices framework and focused on the goals at hand (core business support), IT managers can develop high-level process definitions adapted to their organisation.

However, best practices frameworks tend to be generic in order to fit all organisations and business requirements may be abstract as

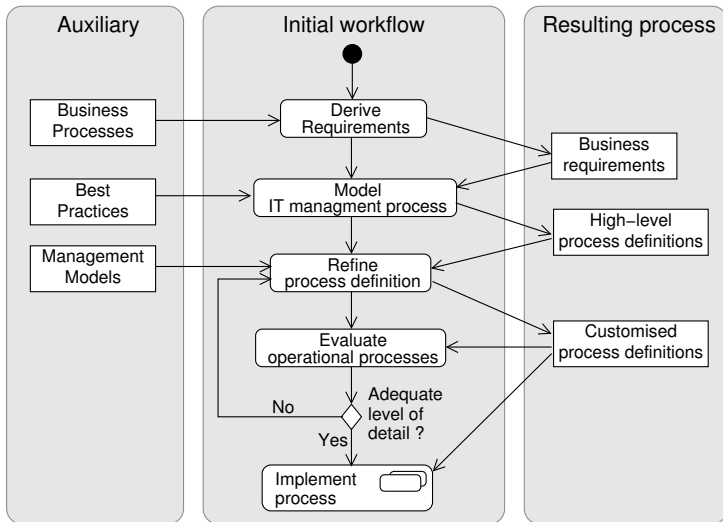


Figure 6.1.: Steps to customised process definitions

well. Through iterative refinement, management processes can be adapted to the characteristics of the the infrastructure and to the role distribution in the organisation. This procedure results in detailed process definitions, that are tailored to the operational requirements of the IT organisation.

By taking into account the significance of a particular process, ideally, the process definitions are minimal in that they only specify the activities that are in fact necessary for the management of the IT services in the organisation. A process specification at this level can be documented in a formal language and thereby rendered machine readable, for use in a facility for tracking and/or executing the process, e.g. a workflow engine.

Refined process

Refinement of a high-level process implies the introduction of semantics specific to the IT organisation. This includes assumptions regarding the actual workflows of real persons assigned the roles in the process, reliance on a set of management tools, as well as prognoses regarding the current and future field of responsibility of the IT organisation. These assumptions may or may not be accurate. Speculations regarding the necessary amount of detail in a process

Fulfilment of operational needs

specification imply assumptions regarding the knowledge of the people executing the process. Reliance on certain timing assumptions is dependent on the behaviour of administrators, as well as external parties (e.g. subproviders). Specification of interfaces, as well, is a subject that must take into account customers' and subproviders' systems.

Possibly, several iterations must be performed to adjust a process specification to the actual needs of the IT organisation. When a process specification becomes adequate with regard to its management goals, it is ready to be implemented.

### 6.1.2. Implementation phase

Translation

The process specification can be treated as proposed in the methodology described in Section 4.4. Accordingly, the control flow of the process is captured in a set of management policy rules, and the data flow is ensured by deriving instructions regarding generation and aggregation of composite events. As a prerequisite for process-to-policy translation, the management process specification must be available in a formal, machine readable representation. References to roles, to entities, to management applications should be made according to the management information databases employed by the IT organisation.

Deployment

The results of translation must be made available to the components that evaluate the policies and those that are responsible for data handling, respectively. This requires on one hand that a form of policy repository be present that can accept and verify a set of policies. On the other hand, a facility for the processing of events must be available that is capable of executing the aggregation instructions produced by the translation.

Distributed evaluation and execution of policy rules require a strategy for distributing each rule to one or more destination, according to its scope. Likewise, instructions pertaining to data flow control may need to be distributed, if a specific management infrastructure requires it.

Initiation and execution

When the management policies are in place, and the information transport facility has been successfully configured, the operational process can be placed in operation. Its execution phase begins effectively when the first process instance is launched.



In analogy to the service life-cycle, the execution phase of a process may be interrupted by the need to alter the process specification.

### 6.1.3. Change and iterative refinement

Operational management processes are intended to be stable over time. Changes to the process specifications incur cost and should not be undertaken unnecessarily. There are situations in which altering the process specification is beneficial, e.g. by exploiting an opportunity to simplify or shorten the standing management procedures. External pressure, e.g. by actors in the customer role can make necessary a review of operational processes. Internal circumstances in the IT organisation, e.g. organisational restructuring, could entail changes to the management processes.

The need for modifications of process specifications often constitutes a consequence of change in other management disciplines. A number of exemplary reasons for changes to process specifications are given in the following:

Actuators and catalysts

**Business practices or high-level policies.** A change in business strategy may force the IT organisation's to adapt its operations, in order to serve the business processes as needed. Requests on the lines of "we need to serve the customer more quickly" or "we need to provide services more cheaply" may prompt a review and evaluation of management processes, which in turn may suggest improvements to the process specifications.

**Services.** The introduction of new services, significant variations in the volume of usage or the number of customers, as well as outsourcing of parts of the service provisioning may impact on the IT organisation's processes. The formal process specifications must be adapted to reflect that impact.

**Contracts.** Contracts with customers, e.g. SLAs, are subject to modifications. Some processes are quite susceptible to changes in operational agreements. In particular processes (or process functions) that directly interface with the customer/user may have to be reviewed.

**Personnel situation.** Fluctuations of the number of personnel

available to the IT organisation, as well as the location of personnel can have great impact on the correct functioning of the IT management processes. Changes to the personnel situation as a consequence of organisational change (e.g. mergers/fusions, establishment or shutdown of subsidiaries) may require modification to the IT management process specifications.

**Infrastructure.** Technological evolution, growth or contraction in the infrastructure may warrant a review of the processes governing its management. Virtualisation of infrastructure parts, as well as re-centralisation or decentralisation concepts lead to different distributions of systems and different requirements on the interconnecting networks. In consequence, new management procedures may become necessary, prompting a revision of management process specifications.

In a highly automated management environment, where infrastructure elements are manipulated directly, as part of the execution of a process activity, smaller changes to that infrastructure may suffice to force modifications of the process specification.

**Tool set.** The tool sets used for process support and for technical management tasks evolves over time. Old tools may no longer be supported after a period of time (e.g. being discontinued), and other tools may introduced in their stead. Tools with novel, desirable capabilities may be adopted. To retain the coupling between process execution and the current tool set, the process specification may need to be adapted.

**Specification language.** The process specification language preferred for the representation of operational processes may change. Improved versions or new languages may emerge, that warrant a migration of the process specification. Obviously, such changes have a great impact, since every process is affected.

The above by no means constitutes a comprehensive list of the original reasons for changes to processes' specification. It is meant as an overview of the categories of changes that will need to be reflected in the process specifications.

Once a process specification has passed the implementation phase, entering its execution phase, any change to the specification must be reflected in an already deployed implementation.

**Change phase** In principle, an altered process specification can be admitted directly into the Translation phase of the life-cycle. However, as changes are made to the control of an operating management process set, a number of issues must be taken into account.

Two obvious provisions are necessary to ensure consistence in the execution of processes in the context of a change to the process specification.

Issues and requirements

**Continuance permit.** Running process instances must be allowed to complete, lest they leave the infrastructure or information base in an inconsistent state. A process instance may interact with external parties (e.g. customers). In such cases it is particularly important to ensure a normal termination of a process instance. However, some processes may be in execution for a long time (compared to the time needed to alter their specification). Even in such cases, at the very least, process instances being executed should be aborted in a controlled manner.

**Retention of operations.** In many settings, it may be unacceptable to suspend operations during review and change of process specifications. Therefore, it may be necessary to allow new process instances to be created, e.g. if such instantiation is performed as a consequence of users' or customers' requests.

The above provisions seem to suggest that perhaps process specifications cannot be changed at all, once they are admitted into their execution phase: running instances may not be stopped, and at any time new instances may be created. A number of measures can be taken to allow changes to be introduced without delay.

Facilitating change

Execution checkpoints incorporated in the process specification can promote rapid and controlled termination of a process instance. However, such checkpoints may need to be set manually, in order to ensure a known, consistent state of the infrastructure, the tools, and the interfaces to external parties.

Process  
versions

Versioning of process specifications allows the differentiation between multiple concurrent versions of a process. Version identifiers may be propagated by means of the events linking the execution of different process fragments. The management policy rules may evaluate “their” version of the process in the same manner in which different process instances are separated by an instance identifier. In consequence, multiple sets of rules can exist concurrently, that represent the same process or process fragment.

Management of process versions requires the inclusion of date clauses in process specification documents. As changes may apply only to small parts of a process specification, a process formalism that supports flexible timestamps for process elements will prove helpful during the change phase.

Differential  
change

In most change instances, only the part of a specification having been altered, together with dependent parts, needs to be re-generated. Identification of the effectively altered process fragment, and its subsequent translation allows minimal modifications to the policy set and the data provisioning instructions.

#### **6.1.4. Decommission and retirement**

Under certain circumstances it may be desirable to place a process out of operation. This may be a temporary measure, in which case the process is expected to be reintroduced at a later point in time. In other cases, it may be expedient to remove the process specification permanently. Both alternatives should be executed with care, taking into account the active instances of the processes (see the discussion on change in Section 6.1.3). In particular, it may prove adverse to simply delete all traces of a process from the management infrastructure.

## **6.2. Functional components**

The architecture presented in the following combines the canonical PbM architecture with facilities for process specification handling, event transport and process-to-policy translation. In the following, the architectural building blocks shown in Figure 6.2 are motivated and outlined from a functional point of view. The purpose of the each of this functional building block is illustrated by means of its

role in the management life-cycle. The required functionality to be provided is described, along with the principles for its realisation.

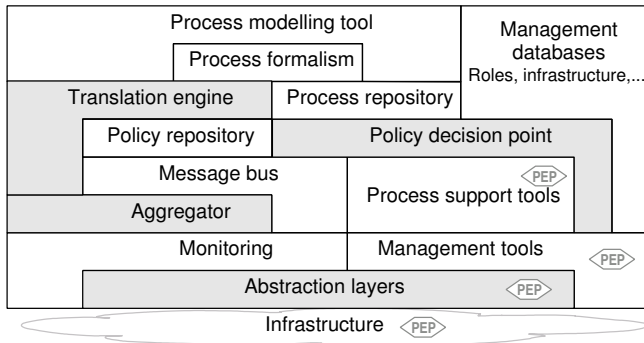


Figure 6.2.: Functional view on architectural components

The building blocks of the architecture sketched in Figure 6.2 are arranged with regard to their associations, i.e. connected blocks cooperate in some way. Also, they are placed according to dependencies, i.e. upper blocks tend to rely on lower blocks. Please note that the position of a building block in relation to its neighbours does not constitute a decisive statement regarding interoperability; a detailed discussion of the interaction between the functional entities in the Figure is found in Section 6.3. The Policy Execution Points (hexagons) that are indicated in several building blocks are logically functional entities, but the actual execution of a function is performed by the “host” component. The functional entities treated in this work are dyed grey.

### 6.2.1. Process management station

An IT organisation performing process-oriented management requires facilities to design, model and maintain process specifications. The results of this task may be of great impact on the correct functioning of the infrastructure and the services provided. Therefore, it is often placed in the responsibility of a process manager (e.g. by ITIL recommendation). The functional components required for the creation and maintenance of process specifications are subsumed in the *process management station*.

**Modelling tools** Graphical modelling tools are employed in order to design and refine the process specification. They should support easy editing of the process and offer different views on the process. Important distinctions of views are the levels of detail, as well as the organisational boundaries relevant to the process. Tools that can perform verification of process detail are advantageous, if subsequent translation is intended. In particular, references to elements described in management information databases could be verified automatically.

**Repositories for management information** Process specifications concern the management of services and infrastructure. To effectively adapt a process to one IT organisation in particular—which is desirable, if it is to be employed in that organisation—it must be enriched with knowledge regarding the specific services being offered by that IT organisation, regarding the specific infrastructure being used to provide these services, and regarding the organisational structure of the personnel manning the IT department.

Such knowledge is commonly captured in different kinds of information repositories. Knowledge about the infrastructure is formalised in models that can be kept in a database, e.g. a Configuration Management Database. Knowledge about services is often more disparate, as several aspects of a service may be worth recording. Some approaches, e.g. the one described in [DgS 07], target a unified management information base to describe services. Today, more often than not, technical information regarding a service is spread between different repositories. The adoption of IT Service Management frameworks will alleviate this problem. Finally, knowledge about entities, persons and roles is often maintained in directories. Often, these directories serve as an instrument for authentication and authorisation (e.g. *Lightweight Directory Access Protocol (LDAP)*-enabled directories).

IT management process specifications constitute “recipes” for the operation of the IT organisation. They must refer to current and accurate information in order to produce beneficial results. Hence, it is important to separate the information repositories from the actual process specifications. Instead, a process should refer to information items in those repositories, thereby accessing the most current state of information available to the IT organisation. Therefore, the structure of the management information repositories must

be accessible in the design phase of the process, thus rendering the repositories themselves important auxiliary components of a process management station.

**Process repository** When the specification of management purpose, goals and actions is held solely in formal process descriptions it is important to manage these descriptions for reference, as well as for the instant when change to the processes must be reflected in their policy-based implementation.

The most important function of a process repository is to safeguard the IT management specifications during all phases of their life-cycle. It should allow easy access to any of the deposited specifications while optionally enforcing access control. Ideally, it should ensure that process specifications are in a consistent format (e.g. in the same process representation language), and provide validation functions to ensure that only syntactically valid specification documents are introduced into the repository.

Process  
repository  
duties

A repository for processes will most probably be part of a process management tool. It may be realised by means of a specialised data schema being implemented with the aid of a general-purpose database system (e.g. an RDBMS) in addition to the program logic that ensures its functions.

Process  
repository  
realisation

### 6.2.2. Management policy architecture

For the purposes of this work, an event-driven policy architecture, resembling the canonical one described in Section 3.4.3 is sufficient.

**Policy repository** The policy repository is part of the canonical policy architecture. It serves as a central point of storage for policies and provides an integrated view on the base of policies available in the system.

**Policy Decision Point** Evaluation of policy is performed by a policy decision point that is specified as a part of the canonical policy architecture. Upon receipt of events, the PDP identifies the policy rules that are to be evaluated. It evaluates the identified set, taking into account their condition clauses. For every policy that is deemed

to be enforced, i.e. its action(s) executed, the PDP is responsible for triggering the execution of these actions at the correct Execution Points. Implicitly, the PDP is responsible for the resolution of role, domain and other wildcard expressions. If a certain action is to be executed on all elements in a domain, the correct set of elements must be determined by the PDP in order to allow proper execution of the policy.

**Policy Enforcement Point** The architectural element topologically closest to the managed infrastructure is the policy enforcement point (sometimes denoted policy *execution* point). It should be seen more as a functional, abstract component than as an actual software package: it denotes the location in a system architecture where policy is enforced. Of course, it can be realised by agents residing on a device or system. It may also be a component or function that is manipulated directly. Note the PEP indication in Figure 6.2, and the absence of any independent PEP building block.

Logical point  
of policy  
execution

However the nature of the PEP, it is responsible for the execution of the action or action set specified in a policy. Therefore, it must be equipped to both comprehend the action specification given in a policy rule, and be able to invoke the necessary functions on its “host” component.

### 6.2.3. Process-to-policy translator

The translator’s duties revolve around the translation of process specification into policy expressions. Consequently, it should provide the capabilities required by the procedures described in Chapter 4. The translator needs to read process specification in a given process language, parse them, and create a process graph. It must be able to apply the substitution rules described in Section 4.3 to that graph and acquire a fragmented form of the process specification.

Process  
decomposition

The translator must be able to match patterns from a pattern catalogue against the fragments of the process specification, and it must be able to instantiate corresponding policy sets from each matched fragment. Conversely, it should be able to extract a specification for data flow, as described in Chapter 5. Implicitly, it must therefore handle expressions in the chosen policy language, as well as the formalism employed to specify event definitions and data aggregation

Pattern  
matching



instructions.

Additionally, the deployment and installation function for both kinds of results are within the duties of the translator. It should therefore be able to introduce policy rules in the policy repository, as well as make available event generation and data aggregation instructions to the respective functional components. These, in turn, should they be distributed, should present a single (each), consistent interface to the translator.

Deployment

#### 6.2.4. Facilities for information transport

Process execution necessitates an appropriate, uninterrupted flow of information. Several of the architecture's building blocks are dedicated to providing it. Together, they form a chain of functions that extracts, processes and refines data into information relevant for an IT management process. Accordingly, they are arranged in different layers, each with its own responsibilities regarding concentration and transport of process-relevant data.

**Monitoring facility** The most basic function in any management setting is to obtain information about the objects being management. As an IT management process may react to changes in the state of infrastructure elements, a monitoring facility is paramount. Additionally, some reference process collections prescribe monitoring as a means to acquire tactical knowledge about the infrastructure, and about the services provided. For example, ITIL's Availability Management and Capacity Planning processes rely on monitoring data to discover weaknesses in the infrastructure and propose remedies.

The monitoring function can be fulfilled by a single tool, or by a whole collection of tools. However the realisation, the main requirement on a monitoring facility remains that it be able to provide the information necessary to the proper execution of all processes. This implies that the collection of monitoring tools may have to be extended to support all information needs.

One or more  
monitoring  
tools

**Data aggregation facility** Process fragments are translated to sets of single policies, that trigger on events. Policy actions can have parameters that reference information items in management databases

or managed objects' representations. Nevertheless, it is desirable to prepare customised bundles of information, and to make them available to the policy rules as they are evaluated and executed.

Preparation of information bundles

This approach takes into account transient information (e.g. messages generated in the infrastructure) that is only available a short time after being generated. Also, it takes into account aggregation of e.g. monitoring data. For example, a process fragment may perform its action in dependence of statistics on a service, e.g. the average number of transactions over the last hour.

Arbitrary composition of data elements

The purpose of an aggregation function is to provide a generic means to aggregate any data relevant to the process into self-contained messages that can be distributed over an adequate transport facility. Ideally, it should accept data both from a monitoring facility, and from the management information databases employed by the IT organisation. It should be sufficiently flexible to allow any combination of data elements from these sources.

The structure of the aggregations produced by this functions must correspond to the information needs of a process fragment, as determined by the translation engine. Therefore, the aggregation facility must be flexibly configurable to fulfil those needs.

**Message bus** Information transport between—as well as within—process activities can be performed by means of events, once transition to policy representation has been completed. A common way of implementation is to use an *event bus*; most of the time, such a bus employs an observer pattern and allows pushed or pulled delivery of events. Examples include the CORBA Notification Service, the Java Messaging Service, as well as messaging facilities based on web-services. In particular, an transport function for process information must be able to carry information bundles, as produced by the aggregation facility. Also, it needs the capability to ensure that an information packet was delivered, and to be able to determine if, for some reason, delivery failed.

Broadcast characteristics

The information transport facility is called a message *bus*, since the characteristics of a bus may be necessary to satisfy the transport requirements of the policy-based implementation of IT management processes. It may be the case, that multiple management policy rules will need the same type of information packets. If those policies are evaluated at different physical locations, as may be the case

if a distributed policy architecture is used, the required information must be delivered to them nonetheless. A simple solution, that maintains the separation of concerns regarding information transport and policy distribution, is to broadcast all messages on an information bus. Considering that the messages transmitted consist of refined, aggregated data, the additional signalling overhead can be kept at reasonable levels, provided an adept configuration of the aggregation function.

### 6.2.5. Tools

The approach proposed in this work strives to support the tool set already deployed within the IT organisation (see e.g. the requirements [23ff] listed in Section 2.4). Two categories of tools become relevant in this context: the tools employed to support the processes themselves, and the management tools that are used to facilitate certain aspects of infrastructure management.

**Process support tools** Several classes of tools can be effectively used for process support. However the IT management process is controlled, these tools are necessary to interact with the management personnel (e.g. group-ware applications, email lists) and the customers/users (e.g. web forms), to track the progress of processes-in-execution (e.g. workflow management facilities) and to administrate process-related information (e.g. ticketing tools). Most often than not, the tools for process support are stand-alone applications whose interoperation is realised by hand—or renounced.

As a building block of the functional architecture, the tools for process support contribute to the automation of the policy-controlled process specification. Their capabilities can be invoked to e.g. manipulate a trouble ticket, or to send a notification to customers.

Exploiting  
process tool  
functions

**Tools to support technical management** At the time of this writing, hardly any dedicated service management applications are available that truly allow management of services. However, a great variety of tools have been developed for the benefit of the disciplines of element, systems and network management. They can be categorised by the functional area they target. Diagnostic utilities and tools for discovering and managing the configuration of systems and networks

Integration  
over FCAPS

ensure that the infrastructure is correctly configured. Benchmarking and auditing applications, as well as utilities for enforcement of quotas, and for usage accounting help control the performance and usage of the services provided by an IT organisation, as well as deliver a foundation for planning ahead.

Tool suites

Professional management tools are often fused into integrated management applications that attempt to cover a great number of aspect of a management discipline. Some are made available as “management platforms”, intended to be constructed by selecting interoperable modules to form a management system according to the IT organisation’s needs (HP’s *OpenView* and IBM’s Tivoli constitute examples of such management application suites). Naturally, such application suites are not customised to the exact needs of the organisation that will employ them. Instead, they offer a selection of general-purpose functions that are useful in most environments.

Heterogeneity and change

In most organisations, there are specific management tasks that need to be supported in a highly specialised manner—in dependence of the particularities of the IT organisation. This requirement is fulfilled by creating “home-grown” utilities. In addition, given the cost of off-shelf management utilities, and the effort employed to evaluate management utilities, many managers take a pragmatic approach to tool choice. Tool sets vary, as some tools are no-longer supported, and others offer new, desirable functions. Such changes in the tools set add to the heterogeneity already present in a typical enterprise setting.

Functional view

As the case may be, most IT organisations employ a diversity of tools to cope with the management of their infrastructure. To allow process-driven automation of management tasks, the use of these tools must be embedded in the specifications of operational IT management process. This integration must take into account the variance in interfaces, capabilities and data formats.

### 6.2.6. Abstraction layers

In a way, management is the art of abstraction. Only certain details of resources, systems, networks, and services are relevant where their management is concerned. In order to contain complexity and scale, various approaches have been described. A common concern is the creation of models, information bases and software that provide an

abstract view of the concrete object of management.

Thus, it is most probable that in a given IT organisation, devices will be in place that create simplified *manageable* views on its infrastructure and its services. The functional component entitled “Abstraction layers” takes this into account. This is appropriate in that a number of other architectural building block may rely on, or profit from, the existence of such devices. Obvious examples include monitoring facilities, as well as management utilities.

### 6.3. Interoperation

After having described the necessary components in the previous section, this section focuses on the interaction between these components along the life-cycle of management specification and implementation.

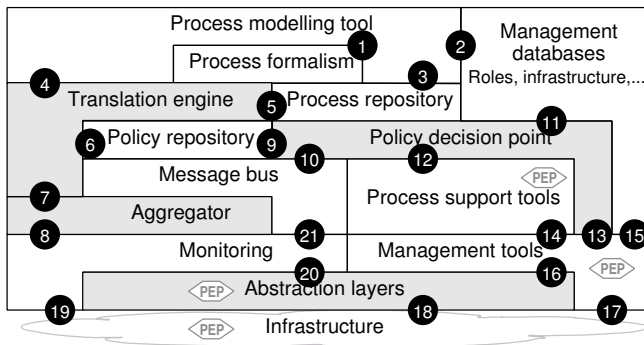


Figure 6.3.: Interactions between functional building blocks

#### 6.3.1. Fundamental interactions

The points of interaction in the functional architecture are indicated by numbers in Figure 6.3. A number is placed between two functional building blocks that interact. In general, the numbers’ sequence follows that of the interactions. This does not apply in every

case, however, and should only be considered to be a coarse indication. In the following, the individual interactions within the architecture are described in the sequence of their respective number.

- 1 The modelling tool must be able to issue process specification in a representation formalism that is acceptable to the translation engine. This function may be built-in, or it may be provided by a library or translation utility.
- 2 To incorporate information captured in management databases, the modelling tool needs to access these databases. Optionally, repositories may push notifications of changes in the management information base to a modelling session
- 3 *Optional.* A finished process model needs to be introduced into the process repository. This can be performed by the modelling tool itself, or by a separate utility. In the former case, the modelling tool needs to collaborate with the process repository; in particular, the formalism employed to represent the process specification within the modelling tool must be equal, or translatable to, the one employed within the process repository.
- 4 *Optional.* Translation of a process specification may be performed after the modelling phase is complete. As an effort towards an integrated process development and maintenance system, the modelling tools should be able to initiate the translation procedure. Alternatively, the translation can be initiated manually.

If errors (e.g. syntactic faults) are encountered in the process specification, the translator should be able to present a notification of the errors to the modelling tool, in order to aid the process designer in improving the process specification.

- 5 To perform the translation from process specification to the target policy set, the translation engine needs to access the input process specification stored in the process repository.
- 6 The primary result output by the translation engine is the set of policies corresponding to the input process specification. The translator should be able to deposit the generated policies in the policy repository, or store them in a corresponding facility.

- 7 The translation engine analyses data flow in the process and generates specifications for aggregation of events. To effectively enable the event-based data flow, the translator needs to make available these specifications to the aggregator component.
- 8 In response to the aggregation specification provided by the translator, the aggregator component configures the components that provide the monitoring function.
- 9 To initiate the execution phase of the process life-cycle, the PDP reads the policies stored in the policy repository. In the case of a distributed policy architecture, all participating PDPs need to access the policy repository.
- 10 Under the circumstances specified in the data flow specification produced by the translator, the aggregator function transmits an information packet (rich event) to the message bus.
- 11 Upon receipt of an event, the PDP selects relevant policies, and evaluates them. In particular, the condition clauses of policies are evaluated. For this purpose, the PDP accesses the available management information repositories (e.g. directory services or models).
- 12 The PDP may decide to execute the management action(s) specified in a policy rule whose the target is among the process support tools. In this case, the PDP must access the interfaces of the tool in question in order to initiate execution of the action.
- 13 A policy action to be executed may target low-level management tools. In this case, the PDP must invoke the corresponding tool in order to effect the execution of the action.
- 14 Process support tools may rely on technical, low-level management tools for some of their functions, in order to perform actions that directly influence the infrastructure. If the PDP initiates an aforementioned function on a process support tool, that tool must be able to invoke the management tool it relies on, in order to effectively complete the function requested by the PDP.
- 15 To effectively perform their function, management tools may need to access the management information repositories.

- 16 Management tools access the abstraction layer that provides a management view of the infrastructure.
- 17 *Optional.* The management databases update their information to reflect the state of the infrastructure (e.g. by using discovery functions).
- 18 The abstraction layer represents the infrastructure. Thus, it must be able to represent current infrastructure data, as well as manipulate its state.
- 19 The components realising the monitoring function access the infrastructure.
- 20 The monitoring components access the abstraction layer.
- 21 The monitoring function transmits events.
  - A policy execution point issues a notification when an action completes.

The interactions describe above suggest a number of mandatory interfaces that allow the functional building blocks to interoperate. In addition, some of the interactions rely on common data formats to allow communication between the architectural components.

## 6.4. Summary and discussion

This chapter has concerned itself with the functional components necessary to realise the management process implementation approach presented in this work. A number of functions were identified and described, along the life-cycle of management processes. Their specific duties and interactions were subsequently described.

The life-cycle of a management process can be broken down into two parts. The first, the *initial* workflow, is responsible for the creation and first-hand refinement of a process into a detailed, operational process specification. The complete life-cycle contains a number of additional phases that are specific to this work. It encompasses the complete process management workflow, starting with the initial workflow part, taking into account implementation, execution and change phases, and finishing with the retirement of a process specification. A number of suggestions are made with respect to



procedures applicable in some of the life-cycle phases, notably the discussion regarding change in process specification.

The life-cycle description provides a roadmap for the introduction, use and decommission of a process specification. It was specified to be independent of specific languages, and, wherever possible, unencumbered by technological or operational assumptions.

To project the management process life-cycle onto a functional framework, a number of functional building blocks have been organised into an architecture, and their duties during the life-cycle have been specified. Several known building blocks, originating both in policy-based and in process-oriented management have been integrated into the functional architecture. Other functional components are instrumental to the approach developed in this work, though they may be identified as general-purpose building blocks. Others, again, are direct products of this work.

Functional building blocks

To complement the enumeration of required functional entities, the interaction sequences between the architectural building blocks have been described. The interactions, naturally, follow the coarse path indicated by the management life-cycle. However, they pertain to pairs of functional components and can therefore be described in more detail than the generic process life-cycle. In particular, interaction points between two components can be identified, along with the direction of a transaction (i.e. which component in a pair delivers information to its counterpart). In a concrete embodiment of the functional architecture, its interactions may differ from those described in this chapter both in granularity and in sequence.

Interactions

Effective interactions between functional components require that, in real components, interfaces are made available that support a given interaction. Based on the interaction sequences described beforehand, a collection of requirements regarding the interfaces of the architectural components are derived.

Interfaces

An exemplary, concrete architecture based on structural, real components is proposed in Chapter 7. That architecture fulfils the purpose of the more abstract, functionally specified architecture presented in this chapter. It becomes apparent, however, that the functional separation of the components differs between the abstract and the concrete architectures, though the basic functional building blocks remain invariant.

Concretisation of the architecture



# Part III

## **Proof of Concept**



# Contents – Part III

<b>7. Exemplary design</b>	<b>225</b>
7.1. Components overview . . . . .	227
7.2. SLPR – A minimal process language . . . . .	231
7.3. The Process-aware Policy System . . . . .	238
7.4. A facility for information aggregation . . . . .	243
7.5. Summary . . . . .	247
<b>8. Evaluation</b>	<b>249</b>
8.1. Fulfilment of requirements . . . . .	249
8.2. Issues and hazards . . . . .	254
8.3. Applicability . . . . .	257



# Chapter 7

## Exemplary design

PRIOR chapters of this work have detailed the ideas behind process-to-policy translation. In particular, the procedure employed to derive suitable partitions of a process and match these to a given set of patterns has been described in Chapter 4. A functional outline for a management architecture suitable to implement an end-to-end workflow based on the notion of process-to-policy translation was developed in Chapter 6. The description of the architecture was in terms of functional components, interactions between components, and required interfaces between functional building blocks.

In contrast, this chapter shows how these ideas can be put into practice using existing software components. To achieve that purpose, it concerns itself with two main topics: first, the composition of software packages to form a management system, and second, the detailed description of the components particular to this work, i.e. those that contribute to the translation of process specifications to management policy rules.

As becomes apparent from the examination of process languages conducted in Section 4.1, a language suitable to represent IT management processes must support a large number of different constructs. Thus, industry strength process languages carry a great amount of overhead, if used only for the sake of quick experimentation with pattern-based translation with a focus on control flow. For this reason, a minimal language, SLPR, has been devised, that conforms to the source meta-model presented in Section 4.2 and offers a concise syntax. Details of the SLPR language definition are found in Section 7.2.

Minimal  
process  
language

For SLPR, a parser/interpreter has been implemented, that reads and validates a process specification and creates a graph data structure suitable for further processing. Based on that generated data

Process  
decomposition

structure, the substitution rules denoted in Section 4.3 were implemented, thus allowing transition of a process specification into the target meta-model described in Section 4.2.2.

Process-aware policy system The management approach proposed in this thesis relies on the existence of a policy architecture that is capable to execute operational management policy. ProPoliS (see e.g. [DaKe 04]) is an implementation of such an architecture. It has been designed and implemented with the purpose in mind to be used in a manner consistent with the approach proposed in this work. Section 7.3 gives a summary overview of the ProPoliS policy language, which was also included in the examination of policy languages in Section 4.1.6, and the implementation of the ProPoliS engine.

Information aggregation As discussed in Chapter 5, the realisation of data flow within a decomposed process can be implemented by means of a message-based scheme that makes available the correct pieces of information to every process fragment. The *Service Oriented Monitoring Architecture (SMONA)* has been devised to allow the aggregation of information items of different origin into messages suitable for use in conjunction with the policy-based process execution scheme that has been developed in this thesis. It has been designed compliant to the generic architecture presented in Section 5.3, and it is presented in Section 7.4.

Concrete architecture Real-life correspondents of the building blocks comprised by the afore-mentioned generic architecture (see Figure 6.2) are presented in Section 7.1. Together, they frame a management system that uses the afore-mentioned software components and uses the SLPR language. It is notable that in some cases, the distinctions between functional areas in these components are diffuse, compared to the generic architecture described in Chapter 6. From a functional point of view, though, the generic and concrete architectures are equivalent. Also, it is important to keep in mind, that production implementations would use one of the industry standard process languages examined in this work, as well as different and varied combinations of tools, monitoring facilities, abstraction mechanisms and so on. Indeed, to provide the flexibility to do so has been one of the major goals of the approach developed in this thesis.

Flexible configuration



## 7.1. Components overview

The components that constitute the design must conform to the functional building blocks discussed in Chapter 6. The functions attributed to those building blocks must be provided, however their grouping within software components need not adhere strictly to the grouping given in the description of the functional, generic building blocks.

The design of a management system is highly dependent on the tools and utilities already deployed. The management system depicted in Figure 7.1 illustrates a possible phenotype of the functional architecture described in Chapter 6. The correspondence between the components in the figure and their generic counterparts described in Section 6.2 is indicated by the light outline surrounding them (compare Figure 6.2). The numbered circles correspond to the interactions between the functional building blocks described in Section 6.3 (compare Figure 6.3). While this section provides an overview of the components, describing the minimalist example chosen for the purpose of demonstration, but also discussing the selection of components for production use. A more detailed view of the building blocks in the focus of this work is given in Sections 7.2 through 7.4.

### Process modelling tool, repository and translator

SLPR has been designed so that the role of the modelling tools can be performed by a simple text editor. The grammar of the SLPR expressions is easily proofread, and the SLPR parser will flag syntactic errors. Hence, the process repository can be realised as a simple text file containing a process specification conforming to the SLPR grammar. A production environment modelling facility should employ a industry strength modelling formalism. Recently BPMN seems to eclipse stereotyped UML in the domain of process modelling by providing similar expressive power in combination with an graphical set of stereotypes; another reason, one may speculate, could lie in the association of UML with “technical”, software engineering matters. At the time of this writing about fifty BPMN tools were registered at the overview page maintained by the OMG. As our translation mechanism relies on a formal (hence textual) process representation, a tool should be chosen that includes suitable export functionality, preferably the option to output an XPDL2 version of the process.

Minimal  
solution

Tool selection

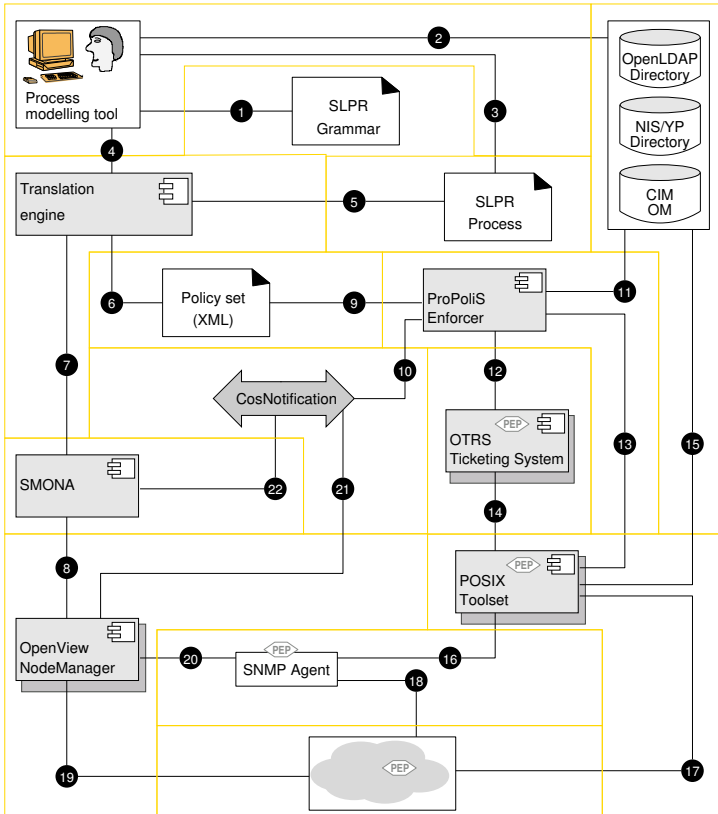


Figure 7.1.: Components of the exemplary design

The translation engine is an extension of the SLPR implementation. Hence its operations are performed directly on the SLPR code which can be read from a file or from standard input (see Section 7.2). As most process formalisms' textual form or counterpart (see 3.2) are specified as XML grammars, the text file approach may be enhanced by one of the "XML-capable" database systems that have been introduced latterly.

XML-based grammars

Obviously, the translation engine is highly dependent on its input language, i.e. the formal, textual process specification format provided by the modelling tool. Another issue is the realisation of the pattern library. While it may be satisfactory to include it in the translator implementation, as was done for the SLPR translator, a more flexible implementation that allows the inclusion of new (optimised) patterns may be desired.

The target policy language selected for production deployment must take into account the available implementations, in addition to the criteria listed in Section 4.1.5. In particular, the ability of the policy enforcement facility to interface with existing management components (tools, information systems etc) constitute important selection criteria.

## Policy architecture

The concept of management policy has emerged in academia, and it has not been widely adopted in the industry (see Section 3.4). As a consequence, only a small number of implementations of policy architectures are available. Beside the ProPoliS, which is described in more detail in Section 7.3, the most notable is the Ponder toolkit. The toolkit (see [Marc 05]) has since been superseded by the Ponder 2 implementation, which appears to be under active development at the time of this writing.

Though policy frameworks generally use textual policy languages for the the specification and (initial) input of rules, the repository may be realised as a text file, or implemented by means of a directory or DB service. Both alternatives satisfy the requirements of the architecture: policies can easily be introduced into the repository as a result of translation, and the policy engine (PDP) can readily access them.

Policy repository

The most important function in the policy architecture is that of

Policy decision function

evaluating policies and deciding whether their action clauses are to be executed. It is performed by a Policy Decision Point (see Section 3.4). As becomes apparent from the interactions indicated in Figures 6.3 and 7.1, it is the duty of the PDP to react to events, select the applicable policy rules, resolve references to external (i.e. not literally in the policy rules) information items, and finally execute the actions included in a policy rule. In the ProPoliS framework, these duties are performed by the enforcer/executor component pair.

Execution

The execution of actions requires interaction with the PEP, which may be an infrastructure element, a management tool and so on. This implies the ability to communicate with these management object using their own interfaces. Alternatively, wrapper code must be written to make accessible the MO interfaces as PEPs.

### **Monitoring, aggregation and data transport**

The specification of data aggregation for the benefit of detached process fragments is performed with the SISL, and it is realised by means of the SMONA system described in Section 7.4. The SMONA monitoring system generates name-value pairs that contain the necessary process data, according to the specification aligned to the information items required for the execution of action (e.g. parameters), the transmission of signals (e.g. passed through data) and the evaluation of conditional expressions (e.g. values within the expressions).

The CosNotification standard CORBA service is employed by SMONA to distribute the aggregated information; however alternatives such as the Java Messaging Service or Web-based Messaging can be employed instead.

The transport of events across the borders of administrative domains require adequate agreements with regard to authorisation and technical access (e.g. agreement on open firewall ports).

### **Management information, tool sets and abstraction mechanisms**

The management and process support tools shown in the diagram constitute examples of common tools that may be available in a large number of installations.

It is futile to attempt a representative selection of tools, or even a comprehensive one. The tool set employed in any IT organisation is particular to that organisation, her management needs and investment history.

A number of tool classes can be named, however, that are likely to be represented in many IT organisations. In the domain of process support tools we can expect ticketing systems to be employed (e.g. the Open Ticket Request System, an open source trouble ticket application), workflow engines, as well as repositories for management information, such as RDBMS-based solutions and directory services, as well as specialised applications for management information (e.g. a *CIM Object Manager (CIMOM)*).

Standard tools

In the domain of more technical management tools a plethora of monitoring tools (e.g. cacti, ganglia) and integrated management applications (e.g. Nagios, HP OpenView, the Tivoli suite) is deployed, along with site and platform specific tools like software package managers, application management utilities and network management tools.

The SNMP Agent component acts as a placeholder for more elaborate infrastructure abstraction mechanisms. However, in most IT organisations, this form of abstraction mechanism is the sole one available. The message bus is realised by means of the CORBA Notification Service (CosNotification) to maintain middleware compatibility to the SMONA implementation, as well as to the ProPolis components.

Abstraction layer

As a selection from the building blocks described in this section, a number of applications in the focus of this work are described in more detail in the following sections 7.2 through 7.4

## 7.2. SLPR – A minimal process language

In Section 4.2 we have established the meta-model that governs common process languages. Most of these languages are either graphical or XML-based. Both of these features can be of benefit to a production environment, either by providing an easy-to-use modelling facility, or by facilitating interoperability. However, to demonstrate the applicability of the concepts developed in this work, we require a

minimal, textual, easy-to-read formalism that adheres to the meta-model shown in Figure 4.1.

The Silly Little Process Representation (SLPR) has been developed for precisely this purpose. It supports the node types described in aforementioned process meta-model and offers no expressions in excess of it. The most important design criterion for the language is simplicity, followed by a concise syntax that should be easy to read and to edit.

SLPR is intended to be used as a “process sketching” language. Process specifications can be sketched quickly, and the simple internal representation of the process allows for straight-forward post-processing of the sketch (e.g., applying the substitution rules described in Section 4.3). While SLPR could be enhanced to represent processes in more detail, this is not a goal of the language. Several real-life, standardised languages exist that can accomplish detailed representation.

### 7.2.1. Language overview

The central idea is that processes are in essence directed graphs. In consequence, SLPR’s main concepts are the *node* (graph vertex) and the *link* (graph edge). Nodes always carry a unique name, while links may carry a guard condition if pointing to conditional branch nodes. There are five types of nodes<sup>1</sup>, listed in Table 7.1. The nodes can be given arguments within parentheses. All nodes take a mandatory argument which becomes the unique name of the node. For example, an event called `HostDown` could be written as `Ev(HostDown)`. A node is created the first time its name is encountered. Every subsequent instance of that name references the node.

Links between action are created by means of a link operator. The link operator is denoted by the two-character string `->`. Thus, to link an action called `NotifyAdmin` to the event, we could write:  
`Ev(HostDown) -> Ac(NotifyAdmin)`.

The `Pa` and `Ko` nodes can take multiple arguments that denote predecessor or follower nodes. These are differentiated between by means

---

<sup>1</sup>Note that the condition nodes are deliberately marked `Ko` (instead of `Co`, as one might have expected). This is to ensure that the keyword cannot be confused with an opening parenthesis, especially when sketching a process definition by hand.

<i>Token</i>	<i>Name</i>	<i>Description</i>
Ac	Action	A process activity/action
Ev	Event	An event/message is expected
Si	Signal	An event/message is dispatched
Pa	Parallelisation	A parallelising or synchronisation node
Ko	Condition	A conditional branch or join

Table 7.1.: Node types in the SLPR

of the link operator. However, with arguments, one of the nodes the operator will link together is the (enclosing) Pa or Ko node. Hence, `Ev(HostDown)->Pa(par1, ->Ac(NotifyAdmin), ->Ac(AutoFailover))` denotes an event, followed by two concurrent actions. In Ko-expressions, guard conditions can be specified by prepending a bracketed string to the link operator, e.g. `Ko(decision1, [condition]->Ac(action1), ->Ac(action2))`.

### 7.2.2. Grammar

After having explained the main features of the language informally, the following specifies the formal grammar of the SLPR.

**Process** A process definition consists of at least a node. A node may be linked to another by means of the link operator. Note that processes may consist of several, disconnected graph partitions.

$\langle \text{process} \rangle ::= \quad ( \langle \text{node} \rangle \{ \langle \text{linkop} \rangle \langle \text{node} \rangle \} )$
--

**Node types** Nodes may be simple (i.e. actions, events and signals) or complex (i.e. parallelisation, branching and joining nodes). A simple node consists of a prefix denoting its type and its name in parenthesis. In contrast, a complex node may have several arguments in addition to its name. These denote “inbound” nodes, i.e. nodes that have a link pointing towards the complex node, or “outbound” nodes, i.e. nodes that are pointed to by a link originating at the complex node in question.

<code>&lt;node&gt;::=</code>	<code>&lt;simplenode&gt;   &lt;complexnode&gt;</code>
<code>&lt;simplenode&gt;::=</code>	<code>&lt;sprefix&gt;" (" &lt;name&gt; )" "</code>
<code>&lt;complexnode&gt;::=</code>	<code>&lt;cprefix&gt;" (" &lt;name&gt; { " , " &lt;inbound&gt;   &lt;outbound&gt; } " )"</code>
<code>&lt;inbound&gt;::=</code>	<code>&lt;node&gt;&lt;linkop&gt;</code>
<code>&lt;outbound&gt;::=</code>	<code>[ " [" &lt;condition&gt; " ] " ] &lt;linkop&gt;&lt;node&gt;</code>

Any mix and number of inbound and outbound nodes is legal within a complex node's arguments. Note that, syntax-wise, the "in" or "out" bound characteristic is denoted by placing the link operator before (meaning: the current complex node is origin of the link) the argument in the outbound case. Conversely, in the inbound case, the link operator is placed after the argument, meaning that the node given in the argument points to the complex node.

The type (simple or complex) of the argument nodes is arbitrary. Of course, any complex nodes given as arguments may contain arguments of their own in addition to their name. Due to the uniqueness of node names, it is legal to only give the name of a complex node as an argument. The arguments of that complex node may be given in another context.

Example:

<pre>Ev(HostDown)-&gt;Pa(par1, -&gt;Ac(NotifyAdmin), -&gt;Ko(FailoverAvailable)) Ko(FailoverAvailable, [yes]-&gt;Ac(AutoFailover), [no]-&gt;Si(NoFailover))</pre>
---

**Node prefixes** Node types are designated by their prefixes, as shown in Table 7.1. Only the Pa and Ko nodes are complex; all others are simple nodes, accepting only a name.

<code>&lt;sprefix&gt;::=</code>	<code>" Ac "   " Si "   " Ev "</code>
<code>&lt;cprefix&gt;::=</code>	<code>" Pa "   " Ko "</code>
<code>&lt;linkop&gt;::=</code>	<code>" -&gt; "</code>

**Guard conditions** Argument nodes in Ko-Expressions may carry guard condition if they are outbound, i.e. following the Ko-node. Such conditions may be truth constants or simple expressions. The SLPR is not intended to model conditional expressions in detail. Thus, the following part of the grammar is only useful for constraining the syntactic forms that may be used as expressions. Often, process definitions tend to use natural language to express truth



values. In consequence, the words “yes” and “no” are accepted as synonyms for “true” and “false”.

<code>&lt;condition&gt;::=</code>	<code>&lt;booleanvalue&gt;   &lt;unaryOP&gt;&lt;value&gt;</code> <code>  &lt;value&gt;&lt;binOP&gt;&lt;value&gt;</code>
<code>&lt;booleanvalue&gt;::=</code>	<code>"true"   "false"   "yes"   "no"</code>
<code>&lt;value&gt;::=</code>	<code>&lt;id&gt;   &lt;expression&gt;</code>
<code>&lt;unaryOP&gt;::=</code>	<code>"!"</code>
<code>&lt;binOP&gt;::=</code>	<code>"i"   "j"   "e"   "l"   "r"   "+"   "*"   "/"   "j="   "i="  </code> <code>"&amp;e"   "&amp;l"   "—"   "="</code>

**Comments** C-style comment blocks, between “/\*” and “\*/”, are accepted; nesting of such comments is considered an error. C++ style comments where “//” begin a comment that encompasses the current line can also be included in the process definition. In addition comments introduced with a “#” (shell-style) are accepted. The C++ style, as well as the shell-style comments may be nested within a C-style comment block.

**Whitespace** SLPR tries to be lenient with regard to the use of whitespace. In principle, space and tabulator characters as well as line breaks are accepted anywhere in the process definition. However, operators, truth constants and node prefixes must not contain whitespace.

### 7.2.3. SLPR Example

The example introduced in Section 2.3 could be formulated in SLPR as shown in Listing 7.1.

Listing 7.1: Example process in SLPR

---

```

2 Ev(UrgentPatch)->
  Pa(synch1o,
4     ->Ac(QueueIncr),
     ->Ac(QueryCMDB)
6 )

8 Pa(synch1c,
     Ac(QueueIncr)->,
10    Ac(QueryCMDB)->
    )
12 ->Ac(ClassifyInc)

```

## Chapter 7. Exemplary design

```
14 Pa(synch2o,  
16     Ac(ClassifyInc)->,  
18     ->Ac(NotifySD),  
18     ->Ac(FileRfC)  
20 )  
20 Pa(synch2c,  
22     Ac(NotifySD)->,  
24     Ac(FileRfC)->  
24 )  
26 ->Ko(TriggerStdAction,  
28     ->Ac(ReqIntervention),  
28     [yes]->Ac(ExecStdChange)  
30 )  
30 Ac(ReqIntervention)->Ac(ManualHandling)  
32 Ko(joinTriggerStdAction,  
34     Ac(ManualHandling)->,  
36     Ac(ExecStdChange)->,  
36     ->Ac(PostImplReview)  
38 )  
38 Ko(PIRsuccess,  
40     Ac(PostImplReview)->,  
42     [yes]->Si(IncidentResolved),  
42     ->Ac(Rollback)  
44 )  
44 Ac(Rollback)->Si(PatchFailed)
```

---

Note that the names of the process nodes are contractions of the ones in the original example, with space characters removed. Line 31 is an example of a previously created (in Line 27) node that is linked to a “new” node. The lines 28 and 42 show the use of (simple) truth expressions. Figure 7.2 shows a graphical rendering<sup>2</sup> of the process specification in Listing 7.1. Note the striking similarity to the example shown in Figure 2.4.

### Summary

The Silly Little Process Representation is a minimal, “toy” language, designed to quickly outline a process using a simple syntax. The language complies to the generic meta-model for processes which was described previously in this work. The formal grammar described in Section 7.2.2 has been implemented as a `javacc`-based parser that generates a directed graph of the process nodes.

---

<sup>2</sup>The graph has only been edited by hand in order to fit it on the page.

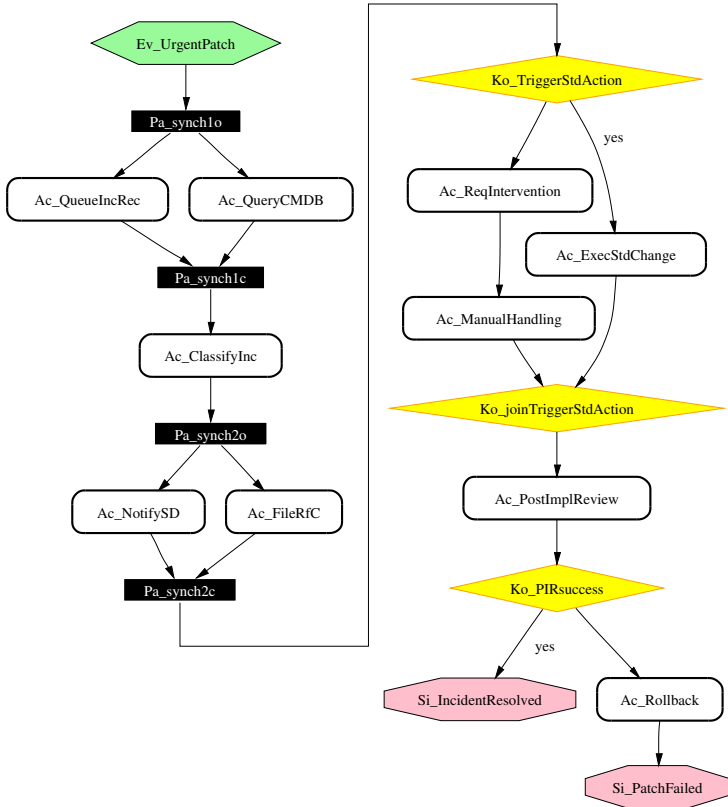


Figure 7.2.: SLPR process  
Graphical rendering of the SLPR process from Listing 7.1

The SLPR has been used as a base for the implementation of the substitution rules for transformation between the two meta-models (generic and target) that offer the foundation for process partitioning and pattern matching.

## 7.3. The Process-aware Policy System

The ProPoliS was conceived as a tool for evaluating and enforcing management policy rules. Its design takes into account the case of policy rules generated from process specifications. The system consists of four separate, distributed and replicable components that interoperate to implement the policy decision and execution task. In particular, the ProPoliS fulfils the functions required by the architecture described in Chapter 6 with regard to the PDP/PEP and policy repository building blocks. Incidentally, it has also been used for research projects unrelated to process-oriented management.

### 7.3.1. Language

The ProPoliS policy language was designed with management policy in mind, and as suggested by its name, for the purpose to be used in the context of management processes [DaKe 04]. In addition, it seemed appropriate to facilitate extensions to the language, in order to accommodate future needs. The concepts included in the language's design reflect those intents.

Representation of policy rules, as well as all of their parts, is achieved by means of a class hierarchy rooted at the abstract class `Policy`. Any policy rule consists of *component sets* that contain policy components of one specific type, e.g. the action set of a policy rule will contain the action specifications pertaining to that rule. Figure 7.3 shows a class diagram of the package that represents policy rules and their components.

**Modularity and references** Policy components can be defined in the local context of a rule, or in a global context. Component sets can thus contain actual component specifications, or references to components that have been defined in the global context. By means of this modular composition of rules, component values can be separated from the composition (and thus the intent) of a policy rule.

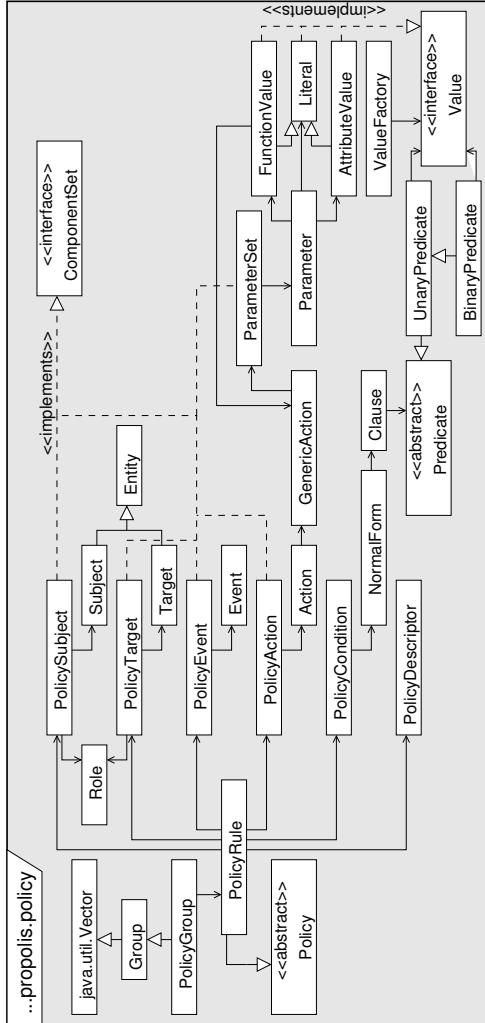


Figure 7.3.: Classes representing ProPolis policy components

Local and  
global  
components

For example, an entity definition can be varied with regard to form or content (as a consequence of change to the database providing such definition, or as a consequence of replacing an infrastructure element) without concerning the policy rule itself. The chief benefit, however, is that components need only be defined once, in the ideal case. Obviously, the use of references demands the presence of an unique identifier for every component, and that the referential integrity of rules be maintained.

Policy  
descriptor

**Management information** The ProPoliS language provides support for a variety of management information items pertaining to the management of the policies themselves. In particular, every rule carries a descriptor component that contains information about the creation—or generation—of the rule. Comments can be added to every component as well as every rule. To allow retracing of policies to their origin process partition, every component can be tagged with the identifier of the process or fragment that it was generated from. In the current version of the grammar, these tags are free form, though a more rigid addressing scheme could be employed.

Tagging

**Grammar** The ProPoliS language is specified by an XML Schema [XMLS-0, XMLS-1, XMLS-2] grammar. The grammar definition is given in Appendix 10.

### 7.3.2. Components

The ProPoliS implementation consists of four components written in the Java programming language and distributed by means of a CORBA [OMG 04-03-12] *Object Request Broker (ORB)* library. The components are outlined in Figure 7.4.

**Repository** The policy repository is a standalone component that manages the policy set available at any one time. It offers an interface to a componentised storage service; for the time being, only a FileStorage version is implemented, i.e. the input policy rules are stored in their XML form in a text file. The repository component will read, parse and verify the rule set on request.

Successfully parsed and active (i.e. marked as enabled) policies are made available to the enforcer component. Taking into account the

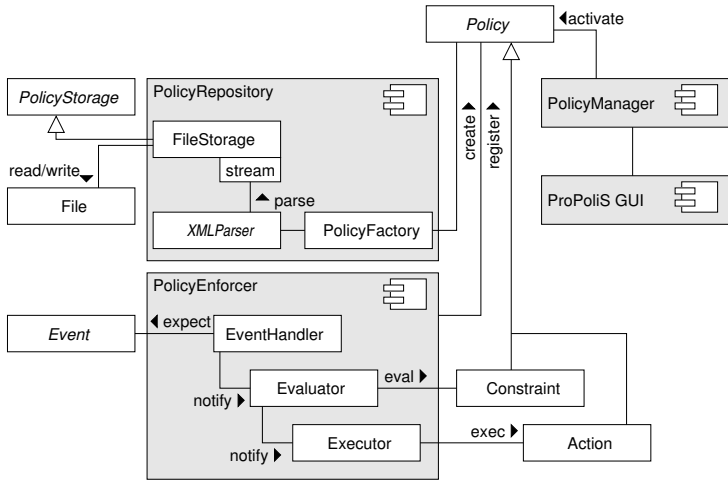


Figure 7.4.: Basic components of the ProPolis

modularity aspects discussed beforehand in this section, beside policy rules a number of unattached components may be held in the repository. In consequence, a snapshot of the repository can be expected to exhibit the structure shown in the instance diagram in Figure 7.5. From the defined set of rules and components some are assembled into policies, some are defined in-line and therefore pertain to the rule where they were defined, and some are neither.

**Enforcer** The enforcer component is responsible for selecting applicable policy rules upon receipt of an event, deciding which of these are to be executed, and executing the actual actions within those rules. To fulfil these tasks, the Enforcer must detect the receipt of an event, select the policies to be triggered on that event, and evaluate the conditional expressions of that rule set. For all conditions evaluated to *true*, the actions given in the corresponding rules are executed on the target objects specified within the rules.

The actions executed on target components are executed by means of dynamically composed calls. That means that all parameter values other than constants are determined immediately before the call. Also, domain and role values are re-read to determine the group

Dynamic call  
composition

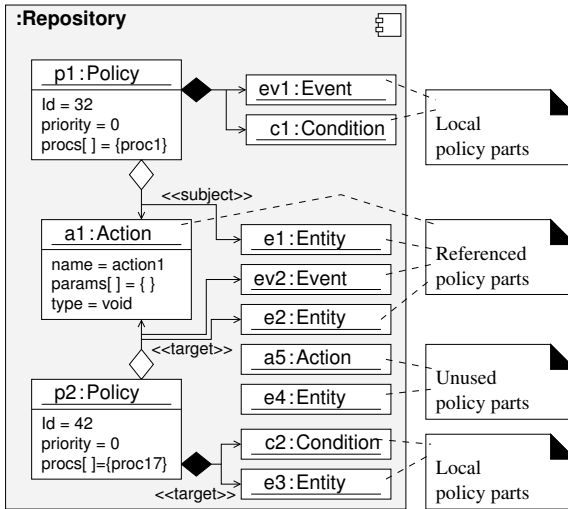


Figure 7.5.: Snapshot of policy repository

of targets that the action is to be executed upon. The CORBA *Dynamic Invocation Interface (DII)* is employed to compose the calls from information present in the policy representation.

Evaluation of conditions

The condition clause of a policy rule must be evaluated to *true* in order for the policy action to be executed. As indicated in Section 5.1.1, the values used within conditional expressions may be attribute or function values, to be retrieved when needed. Thus, to effectively evaluate conditions, additional interaction with MOs must be performed.

Evaluation and execution

In summary, the evaluation and execution workflow follows the strategy of sifting through the policies that are to be executed as a result of the reception of an event. As suggested in the overview of the Enforcer component in Figure 7.4 an event received by an event handler allows the selection of the policies declared to trigger on that event. This selection is made available to *Evaluator* instances that evaluate the conditions given for any single of these policies. For every evaluation that fails (i.e. yields *false*), the corresponding policy is discarded. The remaining set of policies is cleared for execution, and it is assigned to *Executor* instances that invoke the actions specified



in the policy rules.

The CORBA-based calls presently supported require a corresponding interface on the target side (i.e. on the MOs to be affected by policy actions). The enforcer component can be extended to support different communications middleware, such as the *Simple Object Access Protocol (SOAP)* [LaMi 07, MML<sup>+</sup> 07, KHN<sup>+</sup> 07] for WS-based targets, and protocols, such as the SNMP [CFSD 90] for targets possessing an Internet management agent.

Accommodating additional technologies

**Manager** The manager component coordinates the interactions between the ProPoliS components. It determines the state of the ProPoliS components (i.e. if they are available and operational), relays user commands from the GUI as needed and tracks the evaluation and execution of policies by the enforcer component. Changes in state, as well as successful or failed execution of policy actions are echoed to the GUI, to be displayed to the user. The manager is responsible for registering changes to the repository and communicating them to the enforcer component, whether these changes are performed by hand or by elements of an architecture as described in Section 7.1.

Connection to GUI

Changes to repository

**GUI Front-end** The ProPoliS GUI, as shown in Figure 7.6, provides a user view onto the state of the policy system. Policy rules, or groups of rules, can be (de)activated by an operator (top, left part of the screen-shot). After policies are added to or removed from the repository, a reload of the policy sources can be initiated by the operator (top, right). The progress is shown in the bottom part, detailing which policy was triggered by which event, and whether its action(s) were in fact executed or not. The centre part of the GUI provides a browser for the policy rules and policy components which are available in the repository.

## 7.4. A facility for information aggregation

IT management processes require information from a variety of different sources. They include management models, management databases (e.g. CMDB, CRM-databases), storage for process state, as well as sources residing in the IT infrastructure. Furthermore, the information needed by a process activity may constitute a mix of information

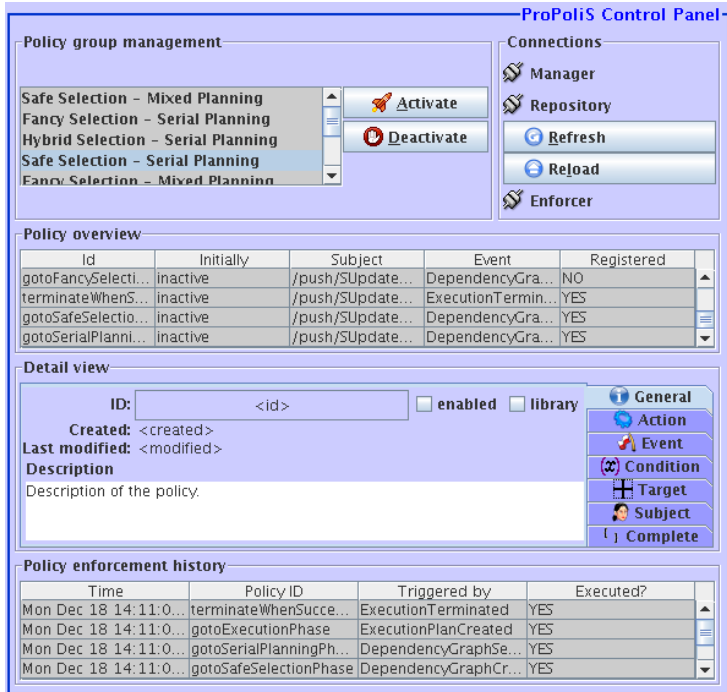


Figure 7.6.: Screen-shot of the ProPoliS GUI front-end

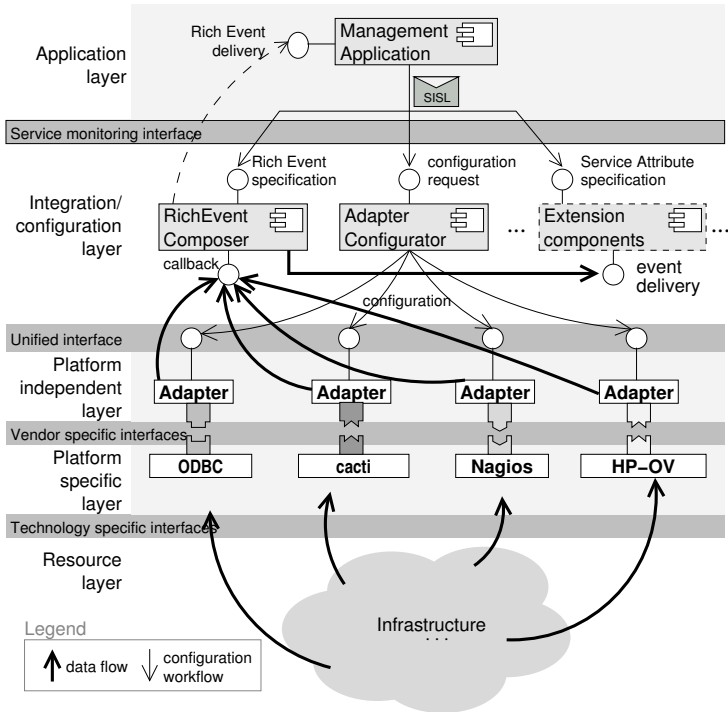


Figure 7.7.: Architecture for service data composition

from many such sources. For example, a comparison between the service level actually received by a customer with the value specified in a *Service Level Agreement (SLA)* requires information about the customer, the service, the SLA, as well as monitoring data pertaining to the service.

Management information comes from many different sources.

The latter class of information sources is addressed by the layered architecture presented in this section. It was designed for generic aggregation of monitoring data into useful management information [DaSa 05]. One primary objective of the architecture is to facilitate service-oriented management efforts [DgS 07, Sail 07, DHHS 06]. The basic data acquisition and aggregation functionality will suffice for the majority of the data flow requirements of this work.

The Service Monitoring Architecture

A basic requirement to an architecture supporting aggregation of information is to support reuse of existing data sources, such as already deployed management tools. Obviously, such data will be delivered in different formats. To interact with such tools, different APIs will have to be used, e.g. for requesting data or configuring monitoring options. Therefore, the consistent specification noted in SISL needs to be broken down into different “languages” according to whatever tools are used as data sources. In addition, the data received from all sources must be converted into a common syntax that can be used to describe the service.

The resource layer This layer encompasses infrastructure components, applications and other sources of “raw” data. The data available at this layer is specific to each resource/component, though some standardised interfaces and data formats may be available.

The platform specific layer Resources are often managed by means of more or less specialised management tools (including scripts and “homemade” tools) that are found in the *platform specific layer*, they provide information pertaining to the infrastructure. Typically, information extracted from the resource layer will be processed and made available in a variety of formats.

Platform independent layer To overcome the heterogeneity in the data sources at the platform independent level, *adapters* provide unification of the data format and basic configuration options. Every adapter needs to be capable of configuring the underlying resource (or tool) and of extracting the required data. The adapters present a common interface towards the higher layers.

The adapters’ main task is to harvest the data and perform pre-processing as required by applicable **function** statements and deliver it in a common format to the *RichEvent Composer* component. Delivery of aggregated data is governed by conditions that pertain to temporal aspects, the method of delivery (push/pull) as well as conditions regarding other events in the data gathering process. Examples include the number of samples collected and thresholds for collected values.

Integration and configuration layer To produce the service information required, as defined e.g. in a SISL document, adapters may need to be selected and configured accordingly. The task of managing the adapters, including deployment, activation and configuration, is performed by the *Adapter Configurator* component. The configurator uses the **resource** declarations to

determine the appropriate adapters, instantiate and configure these according to the `interval`, `function` and `source` clauses (see 10).

The *RichEvent Composer* performs the composition of data according to SISL specification. It gathers the (pre-processed) data from all adapters related to a service attribute and produces data records in accordance to `aggregation` specifications. The `notification` clauses determine when data records are compiled and dispatched. The resulting records (called *RichEvents*) can be made available to a management application (indicated by the dashed line in Figure 7.7) and/or relayed to other components (as . In practice a middleware bus is used for notification transport.

The SMONA is extensible by means of downstream components, as indicated by the *extension component* in the diagram. A number of ongoing efforts in the domain of service and grid management supply specialised components in this manner, that rely on the overall SMONA framework but perform functions specific to the intent of the aforementioned efforts. For the purposes of this work, however, such extensions are not necessary; the basic SMONA framework components do suffice.

Extensions

## 7.5. Summary

This chapter has reviewed a possible realisation of the architecture presented in Chapter 6 with a focus on components specific to the management approach presented in this work. The components developed in the context of this work have been examined in detail.

The ProPoliS policy language examined in Section 4.1.6 has been found suitable as a target language for the realisation of process-to-policy translation. In this chapter, the ProPoliS implementation has been reviewed in the context of using it as a means for automated, policy-based execution of management processes.

Developed components

A simplistic process language, SLPR, has been developed as a research tool. Based on the implementation of its parser, the mechanisms for process decomposition and translation presented in Section 4 have been discussed. It is emphasised that SLPR, being a demonstration tool, should be replaced with one of the process representation formalisms discussed in Section 4.1 when aiming for a production implementation.

Finally, a monitoring and aggregation framework, SMONA, that corresponds to the generic architecture for an information processing facility described in Section 5.3 has been presented.

Expositive  
character

As already noted, these components together with the remaining ones depicted in Figure 7.1, assemble an illustrative management architecture. Its purpose is to demonstrate how an end-to-end realisation of the approach proposed in this work—beginning with the specification of an IT management process and concluding with the execution of management policies generated from that specification.

Dependence on  
legacy

A real-life implementation of the approach would substitute some of the components described in this chapter, as recommended in Section 7.1. One of the core goals of the approach is to reduce the amount of change to the management infrastructure when introducing or extending process-oriented management objectives. Therefore, it is natural that the shape of any production deployment is highly dependent on the existing tool set.

Evaluation

The illustration of an management architecture design in this chapter concludes the development of the approach. As it has been developed in accordance to a set of requirements put forth in the beginning of this text (Section 2.4), we have now the opportunity to examine the results according to those same criteria in the following Chapter 8.

# Chapter 8

## Evaluation

THE approach presented in this thesis may be advocated on the grounds that the combination of process-oriented IT management and a rising degree of inter-domain collaboration makes necessary a flexible facility for process realisation. This work targets the domain of IT management, and it must as such be judged according to the benefits that can be reaped from its implementation. Conversely, the impact of its shortcomings must be estimated to allow comparison with existing or future alternative approaches.

### 8.1. Fulfilment of requirements

The requirements described in Section 2.4 constitute the touchstone for the approach developed in this work. Their degree of fulfilment, summarised in Table 8.1 (page 251), is discussed in the following. The rightmost column of the table indicates if, and how well, a requirement was satisfied. A checkmark “√” indicates satisfactory fulfilment of the requirement, a light checkmark in parentheses “(√)” indicates partial success in fulfilling the requirement, while a cross “×” indicates failure. The weights assigned to single requirements have been retained for reference in the table. As in Table 2.1, the weight increases with the number of stars.

As indicated by the results column of the table, the majority of the important (three and two stars) requirement are satisfied by the approach presented in this work. In particular, the constraints imposed on the approach by the critical set of requirements, have not been violated.

Requirements 1 and 2 are central to the translation of process specifications to policy rules. The pattern-based approach ensures that for a process fragment there is a corresponding set of parametrised

policy rules that corresponds exactly to the fragment in question. The results of policy-based process execution are the same as when a process would be executed by other means. In fact, even the control flow path will be predictable, except in those process parts which deliberately specify parallelism.

The robustness of process execution (requirement [3]) is in part dependent on the implementation of the architectural components that realise the management system. Failures in components may result in suspended flow of data and/or triggering events. Certainly, such issues can be dealt with at the implementation level, however the introduction of this dependency renders the fulfilment of this requirement partial only.

Applicability

The applicability criteria [4], [5] and [6] are satisfied due to the selection of management policy for the realisation of process specifications. By distributing the actions within the process over a number of policy rules, different infrastructure and tools set peculiarities (e.g. interfaces) can be catered for. Support for arbitrary process specifications is provided [7], as long as the specification is syntactically correct, and the formalism used corresponds to the meta-model described in Section 4.2, as stipulated. Experimenting with the translation method suggests that any process specification can be decomposed into suitable fragments, and hence every process part will be accounted for. In addition, an extension method for the pattern catalogue has been described, which allows the incorporation of new patterns, to handle particularly recalcitrant fragments. However, a formal proof of totality is owing, so that requirement [8] must be considered as partially satisfied. The organisational structure of an IT organisation, or the organisation(s) that it caters for, do not influence the approach directly. Therefore, requirement [9] can be considered to be fulfilled.

Scalability

Process specifications are translated and executed separately, hence the approach is not limited by the number of specifications to be handled (req. [10]), nor by the number of concurrent process instances (req. [11]), or services (req. [12]). A major benefit of policy-based management is its scalability; in our context, this benefit pays off in terms of scalability regarding infrastructure size and complexity (req. [13]).

Flexible application of changes

Changes in services, contracts (req. [14]), infrastructure (req. [15]), the tools employed to manage the infrastructure or to provide pro-



### 8.1. Fulfilment of requirements

<i>Nr.</i>	<i>Requirement title</i>	<i>Weight</i>	<i>RESULT</i>
1	Compliance to process specification.	***	✓
2	Deterministic results.	***	✓
3	Robustness.	**	(✓)
4	Applicability to any service.	**	✓
5	Applicability to any infrastructure.	**	✓
6	Applicability to any tool set.	**	✓
7	Applicability to any process specification.	***	✓
8	Applicability to all parts of a process specification.	*	(✓)
9	Applicability to any organisational structure.	*	✓
10	Scalability to any number of process specifications.	**	✓
11	Scalability to any number of process instances.	**	✓
12	Scalability to any number of services.	**	✓
13	Scalability to any size of infrastructure.	**	✓
14	Changes in services and contracts.	**	✓
15	Changes in infrastructure.	***	✓
16	Changes in tool set.	***	✓
17	Changes in process specification.	***	✓
18	Graceful retirement of process specification.	**	(✓)
19	Inter-domain process execution.	**	(✓)
20	Resilience to changes of process spec. in remote domain.	*	×
21	Scalability to any number of inter-domain dependencies.	*	(✓)
22	Sovereignty of domain owner.	***	✓
23	Cost-efficient process realisation.	***	✓
24	Process execution without expert involvement.	**	✓
25	Reuse of existing/deployed process support tools.	**	✓
26	Reuse of existing/deployed management tools.	**	✓
27	Automation of inter-tool procedures.	*	(✓)
28	Automation of management actions on the infrastructure.	**	✓
29	Accommodation of manual procedures.	***	✓
30	Implementation/deployment without manual steps.	**	(✓)
31	Independence of process formalism.	**	✓
32	Independence of reference process frameworks.	**	✓
33	Independence of management inform. repository type.	**	✓
34	Independence of middleware and protocols.	**	✓

Table 8.1.: Fulfilment of requirements

cess support can be accommodated (req. [16]) by altering the detailed (input) process specification accordingly, and re-generating the policy set that realises (the corresponding part of) the process. The same argument applies when the process specification is altered for reasons other than the execution of a change in the management environment (req. [17]). Retirement of a process is in itself a type of change to the process specification(s). A number of provisions have been discussed in this work, that allow the existence of retired (but not yet terminated) process instances. While practicable, the effectiveness of these measures, and the issues that may arise from their use have not been examined in detail; therefore, we consider requirement [18] to be only partially fulfilled.

Inter-domain management presupposes agreements regarding the control and data flow between domains. In simple terms, managers in two cooperating domains must agree on the actions that may be requested by one domain to be performed in the other. In addition, they must agree on what information is to be shared or exchanged. Thus, cooperative execution of management processes implies agreements on which parts of a process is to be executed in a domain (see e.g. the ASP scenario in Section 2.1) and which process-relevant information set is to pass domain borders. The approach presented in this work allows the separation of process fragments on a per-domain basis, and the message-based mechanism for process data flow facilitates a narrow specification of the format and content of the information items intended to be transmitted between domains. Hence, domain owners maintain their sovereignty—fulfilling requirement [22]—while at the same time, the execution of processes is possible. However, execution of processes across domains in an *automated* manner requires the adoption of the same (or functionally equivalent) management facilities in all participating domains. This condition may prove prohibitive with short-term partners and specialised subproviders, thus reducing the value of the approach regarding this requirement; the partial fulfilment noted for requirement [19] reflects that constrained option. Another inter-domain issue is the introduction of changes in remote domains' process specifications. The approach does not in its current form provide the means to detect and handle these situations, thus failing requirement [20]. Likewise, interdependencies of services, networks etc. across domain borders may change without explicit notice (req. [21]). Yet, management procedures must adapt to these changes in environment. While the policy-based approach employed in this work allows for

flexible adaptation, it does not offer automation support for detection and handling. Since the frequency of change may correlate to the number of inter-domain dependencies, this requirement must be deemed to be only partially fulfilled.

The tools commissioned for technical as well as process management constitute an investment of the IT organisation. One of the principal goals of the approach developed in this work is the protection of that investment. This goal is achieved by the flexible enactment of management processes: process specifications can be designed in dependence of the available tool set. The realisation of the resulting specifications by means of policy rules can accommodate any tool set, as long as the policy architecture possesses the appropriate (technology-dependent) enforcement mechanism. This holds true for management tools (fulfilling req. [26]), as well as for process support applications (fulfilling req. [25]) and information repositories. Interactions between tools (req. [27]) can only be automated if additional “glue” software is available to translate between potentially different data formats and protocols. Such utilities may be available in some environments (see e.g. the gateways between accounting, charging and billing applications described in [Radi 02d]).

Reuse of  
resources

Automation of management actions executed on the infrastructure can be handled directly from within a process, as long as appropriate management tools are available. As the execution of iterative manual infrastructure management actions is prohibitive in all but the smallest-scale environments, the presence of such tools can be relied upon (fulfilling req. [28]). The approach encourages the creation of detailed process specification, in order to maximise the amount of management performed automatically. Expert knowledge is thus supposed to be incorporated in the process specification. In all process parts covered by this conservation of know-how, the involvement of specialists in the actual execution of process instances is no longer necessary (fulfilling req. [24]). In other cases, manager decisions must be made during the course of a management process. Any communication or collaboration tool (e.g. common Internet email utilities, or the dialog facilities of a workflow management system) can be employed to request and collect the decisions or the input data (req. [29]). As the translation of process specifications can be performed in an automated manner, most of the implementation and deployment effort for a process can be put into practice without manual steps. However, a number of manual steps

Automation  
and economy  
of labour

(please see Section 4.4) must be taken, in order to ensure the integrity of the translated process, and thereby to secure the desired result. Therefore, requirement [30] is only partially fulfilled.

Management cost is increasingly dependent on labour cost. The cost-efficient realisation of processes (req. [23]) can therefore be deemed to be achieved, given the amount of automation that may be introduced by means of the approach presented. In addition, the retainment of investments in tools complements the controlled commitment of expert time during the process design phase.

A number of process formalisms have been examined in Section 4.1. Some of these were found suitable for use in the context of the approach, while some failed the criteria employed in the examination. Although the approach relies on the presence of some capabilities in process languages, no elements specific to any of the formalisms examined are indispensable (req. [31]). Process frameworks that provide reference process definitions are taken into account, but do not limit the scope of this work (req. [32]). The same is valid for repositories for management information, as well as processes and policies (req. [33]), communications middleware and management protocols (req. [34]). While their functions are necessary, the capabilities of *specific* products are not relied upon.

## 8.2. Issues and hazards

Disregarding the formal fulfilment of the scenario-derived requirements, the approach presented in this work cannot be evaluated isolated from other common management issues. This section discusses the risks that can ensue from the application of the concepts developed in this work, as well as the areas in which further study is necessary before application is effectively possible.

Depending on its concrete implementation, the approach presented in this work may carry hazards originating in the proposed architecture, the technology selected for its realisation, as well as assumptions made regarding the process modelling approach.

Policy-based management is susceptible to policy conflicts, which, in the case of management policy manifest similarly to the multiple-manager problem: several actions that pursue different goals may be performed concurrently on the same targets (e.g. infrastructure

elements). Highly distributed concurrent execution makes conflicts difficult to detect and handle, even in a single-PDP environment. Distributed PDP schemes aggravate the issue of policy conflicts.

The issue of policy goals is not a new one, and several approaches can be employed in order to marginalise or to eliminate this issue (see Section 3.4.2). However, the additional effort implied by these solutions must be weighed against the risk of conflicts actually happening: processes are essentially serialised in nature. However their control flow is realised in practice—be it by means of policy rules—the opportunities for conflicting actions can be assumed to be few. Where concurrency is specified (explicitly), it is the responsibility of the process designer to ensure that race conditions or conflict of actions are excluded. Nevertheless, conflict situations during the execution of a process cannot be excluded completely.

The progress of execution of a management process is dependent on proper functioning of all entities performing single activities of a process instance. Another type of hazard to the management system can be identified in the functional building blocks that make up the architecture described in Chapter 6 (see Figure 7.1). Some of the components may be implemented as singletons, e.g. the Policy Decision Point, or the Message Bus. In consequence, failure of single components can halt process execution. In other cases, failure may not be central, but still pertain to a component critical to the progress of a current process instance. It may have to be detected and remedied in a distributed environment. Thus, globally perceptible problems can ensue from locally limited failures.

Put into a different perspective, however, failures in process-relevant entities usually stop execution of a process, independently of how the process is realised technically. Therefore, the policy-based realisation proposed in this work does not introduce this hazard as a novel one.

The approach presupposes “correctness” of the management process. This claim applies to syntactical correctness, as well as accuracy regarding the representation of management goals. In simple terms, the formal process specification must respect the grammar of the formal language used; also, it must properly address the management tasks that it purports to organise. In addition, the integration of management tools into the execution of a management process implies current and accurate consideration of (and referral to) their interfaces, data formats, as well as knowledge of their respective *modus operandi*.

Single points of failure

Local failures, global implications

Dependence on correct processes

This premise of correctness may not always hold true. Syntactical correctness and, to some extent, integration of management tools, may be supported by modelling tools. The reflection of management goals in the specification of the operational processes is, however, always a candidate for expert evaluation.

Although, as with component failures, this is not an issue specifically pertaining to the approach proposed herein, diagnostics of processes realised by means of management policy rules may be encumbered by the distributed, event-driven nature of their execution.

Security In a distributed, in-band management setting, security and privacy considerations must not be neglected. The policy-based process execution scheme proposed in this work results in that effective enforcement of policy rules must be ascertained at different PEPs. If the integrity of one PEP is compromised, the results of a management process become questionable. Process data flow is routed between many entities (origin, target and transit systems). It may be corrupted, suppressed or examined along the way.

Encryption and signature systems and procedures could be employed to maintain confidentiality, availability and integrity (CIA) of data flow; however, such measures imply a maintenance effort for the security infrastructure and increase complexity. Common security infrastructures imply the deployment of a Public Key Infrastructure to provide an unbroken chain of trust between secured components. The installation of a PKI that is adequate to the general setting of the mechanisms proposed in this work (see Chapter 2) may prove difficult to establish. It requires a central point of trust (a *Certification Authority (CA)*) within a inter-domain setting, as well as reliable and secure distribution and revocation of keys. In addition, encryption and signature operations are computationally intensive and may slow down process execution. Therefore, providing an adequate security envelope for the automation approach presented in this work constitutes an important topic for further study.

Privacy The concepts for process data flow developed in Chapter 5 allow the distribution of data relevant to process execution across multiple domains. They do not take into account the protection of sensitive information from cooperation partners. Depending on the design of information item aggregations (e.g. in the form of rich events), a subcontractor participating in the management processes of an IT organisation (e.g. the call centre operator described in the ASP scenario, Section 2.1) may receive data that discloses confidential

information about the IT organisation. Apart from the general issue of unnecessarily disclosing surplus information, such disclosure may have legal implications, e.g. when information pertaining to customers or users is involved.

To uphold a level of privacy adequate to operations and legal constraints, careful specification of the data flow must be ensured. In this context, the term “careful” implies the creation of concrete guidelines regarding the information that may be shared across domains. This topic has been the focus of research, e.g. within the scope of identity management in inter-domain settings (see e.g. [Homm 05]).

### **8.3. Applicability**

The approach proposed in this thesis used medium and large scale scenarios as its starting point and benchmark. It is therefore reasonable to discuss its suitability for different types of IT management settings. Unfortunately, it is difficult to accurately assess the applicability of the concepts developed in this work with regard to the management of IT infrastructures of different scale.

Process-oriented IT (service) management is foremost beneficial to rather large IT organisation. Increasingly, it is being introduced in medium-sized and small organisations. The necessity for process support and process automation, as well as the potential for savings increases with the number of persons (IT managers, administrators, etc) that participate in operations and management. Smaller organisations with smaller number of services, tools, and processes may opt for more informal process specifications (i.e. textual description) and approximations their execution (i.e. decisions left to the discretion of the management/operations personnel).





# Part IV

## **Conclusions and Further Work**



# Contents – Part IV

<b>9. Future prospects</b>	<b>263</b>
9.1. Issues for further study . . . . .	263
9.2. Applications . . . . .	265
<b>10. Summary and conclusions</b>	<b>269</b>



# Chapter 9

## Future prospects

As with most theses, the present one can only solve a small number of issues. This chapter discusses interesting issues that have been sidetracked from the scope of this work, as well as problems that arise because of it.

In addition, given the results presented in this thesis, a number of follow-up ideas can be cultivated. Two of these, in the domain of IT management and business processes, are presented in the following sections. The first concerns itself with the generalisation of the results of this work towards a yet more flexible realisation of IT management processes. The second regards the application of the ideas developed in this work to the domain of business processes.

### 9.1. Issues for further study

This work concerns itself with the principles of IT management process automation based on management policy. Several important topics have been disregarded in order to limit the focus of the thesis to a manageable number of problems. In the following, a selection of topics are discussed which are of interest in real-life realisation of management processes.

#### 9.1.1. Metrics for process detail

One important requisite for the concepts developed in this thesis to be effective is the creation of adequately detailed process specifications. For the purposes of this work, it has been sufficient to require process specifications that incorporate the information necessary in

order to drive management and process support tools. A more general definition or metric of the “detailed” process specification has not been given.

Comparability  
of detail levels

For the practical task of prepare specification documents whose level of detail is adequate, process modellers require guidelines and rules that determine what information items are obligatory, i.e. what exactly *is*, in effect, sufficient technical detail in a process specification. Obviously, the concrete content of a process document is dependent on the available management framework, its APIs, the “syntactic sugar” available in the formalism etc. Reference process frameworks are, in general, too generic as to be of assistance where technical, site-specific issues are concerned. Therefore, a common metric is required that allows the assessment and comparison of process specifications with regard to their level of technical detail.

### 9.1.2. Security and privacy considerations

IT management processes constitute critical resources to the IT organisation. Their execution influences the provisioning of services substantially. Therefore, security breaches directly targeting process execution may have grave consequences with a high impact on business. As already noted in Section 8.2, the introduction of an adequate security infrastructure for the approach proposed in this work is both a challenging and an important task.

At the same time, privacy considerations need to taken into account in an inter-domain setting. Assuring adequate protection of privacy during distributed management process execution is an interesting topic for further study.

### 9.1.3. Independent policy

The management approach presented in this work introduces a policy system as part of the proposed management architecture. A beneficial consequence of this fact is the opportunity to specify management policy not originating from process-to-policy translation in addition to the policy rules generated by the translation procedure.

Taking into consideration the potential for policy conflicts, an assessment of the risk for the disruption of process execution if additional

policy is specified is difficult to perform. A related question is the suitability of existing conflict handling approaches to that situation.

## 9.2. Applications

Based on the concepts developed in this thesis, a number of conceivable research projects can be formulated, possibly in areas unrelated to process-oriented management.

### 9.2.1. Generalisation

The approach presented in this thesis uses management policy rules to drive IT management processes. As has been shown in the ELABORATION part, several automated processing steps are performed in order to adapt a formal process specification to the requirements of the distributed control flow that is realised by the management policy rules. In the same manner, the data flow within the process is realised by means of *rich events* that are able to transport information to wherever a certain process fragment is actually executed.

If we generalise these terms, the sequence of actions executed at any policy enforcement point corresponds to a *program*. At the same time, the incoming and outgoing events observed at such a point define a *protocol*. Hence, by performing this generalisation, we have “gone full-circle”, back to the program-like form of an IT management process – with the distinction that it is now effectively distributed and much looser coupled than before applying the procedures described in this work.

What remains interesting is the observation that *locally* deployed policy together with the expected and transmitted events specification correspond to a program and protocol pair. At this level of abstraction, it appears possible to employ other forms of enforcing the management goals than actual policy rules. However, since the characteristics of the program are known, it could be generated automatically from a process specification using the same techniques as described in this work.

One possible target platform could be that of stationary or mobile agents. An agent program could correspond to a part of a process specification, to be executed at a certain point in the infrastructure.

Mobile agents  
based  
execution

This makes for an interesting approach to establish cooperation between sovereign domains: a contract (e.g. an SLA) could include provisions that a jointly specified, automated process specification is to be implemented by one of the contract parties. From the detailed process specification, an agent could be generated and deployed to run within that contract party's domain.

This, of course, raises several questions, the most important ones regarding security issues. In particular, it is necessary to ensure that the code being executed corresponds to the process specification, that it is not altered during transit or at the point of execution, and that the integrity of the event flow to and from the point of execution is maintained.

As mobile agents have once again gained attention recently in the management domain, it is possible that a closer examination of the possibilities they offer in this context will prove rewarding.

### **9.2.2. Bottom-up assessment of OSS tool requirements**

The policy set applicable to a tool set may allow conclusions regarding the functional requirements of that tool set. These requirements may state what the tool does *not* need on the one hand, while on the other hand opportunities to render the interface of the design of the functions in a better manner (i.e. offer facade interfaces targeted at a specific process activity, subsume the sequential use of several functions into one etc).

Reference process frameworks are applied in a top-down manner, i.e. the high-level process specifications in the framework are projected onto operational management. Experimentation with the process realisation approach presented in this thesis may allow conclusions with regard to the requirements on tools and technical procedures that are employed at a technical, operational level. As reference process definitions refrain from detail in that domain, the technical requirements could complement the generic process specifications provided by ITSM guideline collections such as the ITIL.

### **9.2.3. Self-management**

Given a distributed execution of management policy rules, a rule set together with the components it applies to can be said to form



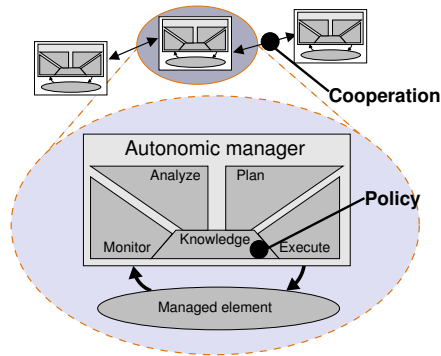


Figure 9.1.: Autonomic elements

a self-managed system: a stand-alone entity that reacts to sensor input (incoming messages/events), makes decisions that lead to action (execution of policy rules), and communicate changes in state outward, if these are deemed to be of interest to external parties (transmission of messages/events).

Although the approach presented in this thesis is primarily targeted at the domain of process-oriented management, it could perhaps be employed in the area of self-managed systems as an aid to the specification of the behaviour of such systems. Instead of attempting to determine the behaviour of each system by specifying policies directly (for each system), the specification of an overall workflow may lead to a more “harmonic” interrelationship (as suggested in Figure 9.1) between systems that are supposed to “help themselves”.

The policies that govern single self-managed cells can originate from a process specification intended to govern an aggregation of multiple self-managed systems with different capabilities and purpose. The mechanisms proposed in this work may serve as a base for the development of such an approach.

## Summary

This short chapter has presented a number of issues for further study in the context of the work presented in this thesis. Open issues arising from the approach presented herein have been indicated as

*Chapter 9. Future prospects*

topics for further study. In addition, areas of application different from process-oriented management have been investigated.

# Chapter 10

## Summary and conclusions

**I**N recent years, IT management has been undergoing a change of focus, from the technical perspective to an organisational one. In particular, the demand for a process-oriented view onto service management has been growing at a quick pace. The significance of documented processes, based on collections of best practices is increasingly being accentuated by the advocates of process-oriented management. In fact, the introduction of process-orientation into IT management does offer a multitude of benefits. The workflows of IT managers become visible, open to optimisation and re-design, and perhaps quantifiable. However, it is important to remember that the actual infrastructure still needs to be managed at a technical level.

This work has presented an approach to coupling the process-oriented view with a flexible means to realise IT management processes. The starting point of the development of the approach has been given by two scenarios that illustrate the requirements imposed by process-oriented management of today's IT organisations. The scenarios imply several plausible assumptions that can be observed to apply to the evolution of the IT infrastructures, as well as to their management. First, we can safely assume that IT management processes will continue to be introduced at a quick pace. Increasingly, they are presented in a formalised manner, i.e. written in a formal process language. To conserve investments, the continued use of existing tools for process support is desirable. Obviously, the continued use of existing tools for technical management is desirable as well. As hitherto, tool sets will change over time, and the administrator population will fluctuate. Services will adapt to market and business needs, and the infrastructure will need to evolve to be able to implement them. Needless to say, the strong trend toward reduction of IT cost is unbroken; it is improbable that it will turn in the foreseeable future.

Rationale and prerequisites

These prerequisites call for a highly flexible technical base for the realisation of management processes. The use of management policy rules appears to offer adequate flexibility to allow the projection of process specifications onto actual management tasks. Therefore, this work has pursued a translation between formal process specifications and management policy rule sets. Processes, as well as policies, are expressed by means of formal languages. Hence, one important part of the problem was a method of translating a process formalised in process language into a set of policies written in an adequate policy language.

Analysis of formalisms

Formal process definitions have been used in domains other than IT management for quite some time. In consequence, several languages have been designed for the expression of processes in a broader sense. Analysis of a number of such languages has shown that while some of these have a strong focus on some aspects of business interaction, others are general-purpose languages that are suitable for the representation of IT management processes. In the same manner, policy languages have been analysed to ensure that the information contained in a process specification is retained in the target language of the translation. A number of languages have been found to exhibit the necessary characteristics.

Decomposition

The actual projection of an IT management process specification onto a set of management policy rules is performed according to a step-by-step methodology. The projection of control flow and the preservation of the data flow within the process have been treated separately. A meta-model pair has been devised that describes a process in in source and target form. A set of transformations have been specified that can be applied to a source process (i.e. one that is modelled in one of the applicable process languages) in order to change its structure.

Patterns and translation

Following the application of these transformations, the process specification may be described by means of a collection of patterns, each pattern describing the structure of a process fragment. This procedure preserves the control flow specified in the original (source) process. Every pattern carries a parametrised policy set that can be instantiated using the information present in the process fragment that matches the pattern. To ensure the totality of translation for any correct process specification, a generating system for patterns has been formulated, that ensures complete decomposition of any process graph and allows its equivalent representation by means

of policy rules. In addition, a procedure for extending the pattern catalogue, e.g. for the purpose of optimisation has been devised.

The decomposition of a process specification into pattern-sized fragments posed the challenge to preserve the information flow encoded into the process specification. Processes transfer information between different tools, persons and domains. The information will be transported in a variety of heterogeneous forms and data types, by means of different transport mechanisms. In addition, as the execution points for policies may be distributed across the sphere of action of the process. Therefore, a mechanism was needed that will provide the information needed to the location where it was needed. These issues are addressed by the combination of a language supporting aggregation of information items, SISL, and a corresponding architecture designed for procurement of data from heterogeneous sources, SMONA. Data transport is performed by means of an event bus that propagates Rich Events, i.e. event notifications that may be employed to trigger the execution of policy rules, enriched with data needed by the execution of those rules. The management architecture necessary to realise the approach in practice comprises a modelling station where the process specifications are created using a formal language, the translator component that generates policy rules from the process specification, and the components employed to execute a thus translated process. The most significant ones among the latter include the components of the policy architecture, as well as facilities for the acquisition and the transport of process-relevant data. In the context of the architecture, an end-to-end workflow has been described, ranging from the conception of the management process specification to the execution of the corresponding policy rules on the provisioning and management infrastructures. The discussion of this general architecture and its associated workflow was complemented by a description of the interfaces required between the included functional building blocks.

Data flow in processes

Architecture and workflow

A concrete realisation of the above-mentioned architecture can be realised by combining standard management components with the building blocks that are particular to the approach presented in this thesis. Among the latter are the Process-aware Policy System, which is the implementation of a management policy architecture, used in conjunction with the SMONA information aggregation facility. A minimal process language compliant to the source process meta-model, SLPR, has been devised in order to experiment with the control flow translation procedure. Based on the SLPR parser, the

Process-aware Policy System

Proof-of-concept

decomposition of process specifications and the matching of patterns have been implemented.

In conclusion, the implementation operational IT management processes by means of management policy rules exhibits a high degree of flexibility and scalability, while introducing a number of issues, most of which are due to process decomposition—thus, the greatest strength of the approach is at the same time a source of weakness. Such remaining issues constitute a collection of topics for further study, either in continuation of this work, or with regard to adjacent topics.

Trends in IT  
management

The complexity of IT and telecommunications infrastructures continues to increase, instead of being reduced. Every generation of technology—whether it be truly innovative, or simply consist of available techniques based on existing concepts—introduces new management challenges. Current examples include mobility aspects, virtualisation and re-centralisation of operations centres, as well as increasing importance of the consideration given to physical space, and power and cooling requirements.

Fortunately, the awareness for the need of management functionality in new designs, products and installations is increasing. Even so, the techniques employed to cope with upcoming management challenges are developed and deployed long after they are actually in demand. A general strategy towards tackling this issue may consist in raising the degree of flexibility available to the IT manager. Similar approach vectors exist in other domains, as well. For example, approaches that strive to leverage the benefits of loose coupling over those of a compact, more closely coupled system are being explored in the domain of software engineering.

~ ~ ~

## List of Figures

1.1. Management pyramid . . . . .	6
1.2. IT management hierarchy . . . . .	8
1.3. Creation of an IT management process . . . . .	9
1.4. Dimension space of management challenges . . . . .	11
1.5. The life-cycle of an IT management process definition	12
1.6. Approach rationale . . . . .	14
1.7. Phases of process-to-policy translation . . . . .	16
1.8. Simple example of pattern translation . . . . .	17
1.9. Overview of this work . . . . .	22
2.1. ASP scenario . . . . .	27
2.2. Grid management scenario overview . . . . .	34
2.3. A Grid computing service . . . . .	35
2.4. Example: The handling of an urgent security patch .	44
2.5. Management tools involved in the example setting. .	46
3.1. The BPM component model . . . . .	74
3.2. Standardisation in workflow and process management	76
3.3. Levels of the Capability Maturity Model . . . . .	78
3.4. Pattern example: Exclusive choice pattern. . . . .	79
3.5. Policy refinement hierarchy . . . . .	81
3.6. Canonical policy management architecture . . . . .	83
3.7. Selected classes in the PCIM . . . . .	84
4.1. Generic, control-flow-centric process meta-model . .	128
4.2. Target meta-model . . . . .	130
4.3. Target process elements . . . . .	132
4.4. Splitting conditional/parallelisation nodes . . . . .	134
4.5. Basic pattern . . . . .	143
4.6. Disjunctive condition pattern . . . . .	145
4.7. Disjunctive condition pattern with single action . . .	147
4.8. Conjunctive condition pattern . . . . .	148
4.9. Mixed condition pattern . . . . .	148

*LIST OF FIGURES*

4.10. Synchronisation pattern . . . . .	149
4.11. The generating system for process graphs . . . . .	163
4.12. Exemplary complex pattern . . . . .	169
4.13. Handling of a Request for Change (generated) . . . . .	171
4.14. Process graph after decomposition (generated) . . . . .	174
5.1. Information flow . . . . .	182
5.2. Abstract view on process data . . . . .	184
5.3. Data items of an action node . . . . .	185
5.4. Process data values . . . . .	185
5.5. Data items of a conditional branch node . . . . .	186
5.6. Bridging the gap between fragments . . . . .	188
5.7. Dimensions of data flow in processes . . . . .	192
5.8. Sources of management process data . . . . .	193
5.9. Aggregating architecture (functional view) . . . . .	196
6.1. Steps to customised process definitions . . . . .	201
6.2. Functional view on architectural components . . . . .	207
6.3. Interactions between functional building blocks . . . . .	215
7.1. Components of the exemplary design . . . . .	228
7.2. SLPR process . . . . .	237
7.3. Classes representing ProPoliS policy components . . . . .	239
7.4. Basic components of the ProPoliS . . . . .	241
7.5. Snapshot of policy repository . . . . .	242
7.6. Screen-shot of the ProPoliS GUI front-end . . . . .	244
7.7. Architecture for service data composition . . . . .	245
9.1. Autonomic elements . . . . .	267



## List of Tables

2.1. Weighted summary of requirements on the approach	60
4.1. Criteria for action and control flow . . . . .	100
4.2. Comparison criteria for <i>object</i> elements . . . . .	102
4.3. Comparison criteria for <i>event</i> elements . . . . .	103
4.4. Comparison criteria for <i>condition</i> elements . . . . .	104
4.5. Assessment criteria and Evaluation results . . . . .	114
4.6. Comparison criteria for policy <i>action</i> elements . . .	117
4.7. Comparison criteria for policy <i>event</i> elements . . . .	118
4.8. Comparison criteria for policy <i>condition</i> elements . .	118
4.9. Comparison criteria for policy meta-data elements . .	119
4.10. Comparison criteria for variables and types . . . . .	119
4.11. Comparison criteria for policy <i>meta-data</i> elements . .	120
4.12. Results of policy language analysis . . . . .	122
4.13. Summary of process nodes cardinalities . . . . .	131
4.14. Elimination criteria for branching/joining nodes . . .	133
4.15. Comparison of node symbols . . . . .	173
7.1. Node types in the SLPR . . . . .	233
8.1. Fulfilment of requirements . . . . .	251

*LIST OF TABLES*

## List of Abbreviations

<b>ASP</b>	Application Service Provider
<b>BEIDTF</b>	Business Modeling & Integration Domain Task Force
<b>BPMI</b>	Business Process Management Initiative
<b>BPML</b>	Business Process Modeling Language
<b>BPMN</b>	Business Process Modeling Notation
<b>CA</b>	Certification Authority
<b>CI</b>	Configuration Item
<b>CIMOM</b>	CIM Object Manager
<b>CMDB</b>	Configuration Management Database
<b>CMM</b>	Capability Maturity Model
<b>CMMI</b>	Capability Maturity Model Integration
<b>DHS</b>	Definitive Hardware Store
<b>DII</b>	Dynamic Invocation Interface
<b>DSL</b>	Definitive Software Library
<b>ebXML</b>	Electronic Business using eXtensible Markup Language
<b>eTOM</b>	Enhanced Telecom Operations Map
<b>eTOM</b>	Extended Telecom Operations Map
<b>FSC</b>	Forward Schedule of Change

## *LIST OF TABLES*

<b>IDS</b>	Intrusion Detection System
<b>ITIL</b>	IT Infrastructure Library
<b>ITSCMM</b>	IT Service Capability Maturity Model
<b>ITSM</b>	IT Service Management
<b>KPI</b>	Key Performance Indicator
<b>LDAP</b>	Lightweight Directory Access Protocol
<b>NGOSS</b>	New Generation Operations Systems and Software
<b>OASIS</b>	Organization for the Advancement of Structured Information Standards
<b>OMG</b>	Object Management Group
<b>ORB</b>	Object Request Broker
<b>PbM</b>	Policy-based Management
<b>PCIM</b>	Policy Core Information Model
<b>PDP</b>	Policy Decision Point
<b>PEP</b>	Policy Execution Point
<b>PIR</b>	Post Implementation Review
<b>PKI</b>	Public Key Infrastructure
<b>RfC</b>	Request for Change
<b>RMI</b>	Remote Method Invocation
<b>RO</b>	Real Organisation
<b>SAMM</b>	System Administration Maturity Model
<b>SAP</b>	Service Access Point
<b>SID</b>	Service Information/Data

<b>SLA</b>	Service Level Agreement
<b>SLM</b>	Service Level Management
<b>SMONA</b>	Service Oriented Monitoring Architecture
<b>SOA</b>	Service Oriented Architecture
<b>SOAP</b>	Simple Object Access Protocol
<b>SPOC</b>	Single Point Of Contact
<b>UC</b>	Underpinning Contract
<b>UML</b>	Unified Modeling Language
<b>UN/CEFACT</b>	United Nations Centre for Trade Facilitation and Electronic Business
<b>VO</b>	Virtual Organisation
<b>WfMC</b>	Workflow Management Coalition
<b>WS</b>	Web Service
<b>WS-BPEL, BPEL4WS</b>	Business Process Execution Language for Web Services
<b>WSDL</b>	Web Service Definition Language
<b>XACML</b>	eXtensible Access Control Modeling Language
<b>XMI</b>	XML Model Interchange
<b>XPDL</b>	XML Process Definition Language

*LIST OF TABLES*

## Bibliography

- [ACD<sup>+</sup> 03] ANDREWS, TONY, FRANCISCO CURBERA, HITESH DHOLAKIA, YARON GOLAND, JOHANNES KLEIN, FRANK LEYMAN, KEVIN LIU, DIETER ROLLER, DOUG SMITH, SATISH THATTE, IVANA TRICKOVIC and SANJIVA WEERAWARANA: *Business Process Execution Language for Web Services Version 1.1*. Technical Report, OASIS, May 2003.
- [AHK+02] VAN DER AALST, W. M. P., A. H. M. TER HOFST-EDE, B. KIEPUSZEWSKI and A. P. BARONS: *Workflow Patterns*. 2002.
- [Arki 02] ARKIN, ASSAF: *Business Process Modeling Language*. Draft, BPMI, November 2002.
- [BKR 03] BECKER, J., M. KUGELER and M. ROSEMAN (editors): *Process management – A guide for the design of business processes*. Springer-Verlag, 2003.
- [BLR 03] BANDARA, AROSHA K., EMIL C. LUPU and ALESSANDRA RUSSO: *Using Event Calculus to Formalise Policy Specification and Analysis*. In *Proceedings 4th IEEE Workshop on Policies for Distributed System and Networks (Policy 2003)*, Lake Como, Italy, June 2003. .
- [BPS 01] *Business Process Specification Schema*. Technical Report, UN/CEFACT and OASIS, 2001.
- [Bren 06] BRENNER, M.: *Classifying ITIL Processes — A Taxonomy under Tool Support Aspects*. In *Proceedings of First IEEE/IFIP International Workshop on Business-Driven IT Management (BDIM 06)*, volume 2006, pages 19–28, Vancouver, Canada, April 2006. IEEE.

## BIBLIOGRAPHY

- [Bren 07] BRENNER, MICHAEL: *Werkzeugunterstützung für ITIL-orientiertes Dienstmanagement – ein modellbasierter Ansatz*. Dissertation, Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München, München, 2007.
- [BSSG 06] BRENNER, M., M. SAILER, T. SCHAAF and M. GARSCHHAMMER: *CMDB — Yet Another MIB? On Reusing Management Model Concepts in ITIL Configuration Management*. In *Large Scale Management of Distributed Systems (Proceedings of DSOM 2006 — 17th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management)*, volume 2006 of *Lecture Notes in Computer Science, Volume 4269/2006*, pages 269–280. Springer Berlin / Heidelberg, October 2006.
- [CCM 94] CERVESATO, ILIANO, LUCA CHITTARO and ANGELO MONTANARI: *What the Event Calculus does and How to do it Efficiently*. In APUENTE, M., R. BARBUTI and I. RAMOS (editors): *1994 Joint Conference on Declarative Programming — GULP-PRODE'94*, pages 336–350, Peñíscola, Spain, 19–22 September 1994. .
- [CCMW 01] CHRISTENSEN, ERIK, FRANCISCO CURBERA, GREG MEREDITH and SANJIVA WEERAWARANA: *Web Services Description Language (WSDL) 1.1*. Technical Report, W3C, March 2001, <http://www.w3.org/TR/2001/NOTE-wsdl-20010315> .
- [CFSD 90] CASE, J.D., M. FEDOR, M.L. SCHOFFSTALL and J. DAVIN: *RFC 1157: Simple Network Management Protocol (SNMP)*. Request For Comments, IETF, May 1990, <ftp://ftp.isi.edu/in-notes/rfc1157.txt> .
- [CIM 05] *Common Information Model (CIM) Version 2.9*. Specification, Distributed Management Task Force (DMTF), June 2005.
- [Clau 06a] CLAUSS, C.: *Entwicklung und Anwendung einer Methodik zur Verfeinerung von ITIL Prozess-*



*beschreibungen am Beispiel des ITIL Change Managements.* Master's thesis, Ludwig-Maximilians-Universität München, November 2006.

- [DaKe 04] DANCIU, V. and B. KEMPTER: *From Processes to Policies – Concepts for Large Scale Policy Generation.* In *Managing Next Generation Convergence Networks and Services: Proceedings of the 2004 IEEE/IFIP Network Operations and Management Symposium (NOMS)*, volume 2004, Seoul, Korea, April 2004. IEEE/IFIP, <http://www.nm.ifi.lmu.de/pub/Publikationen/dake04>.
- [Danc 06] DANCIU, V.: *Formalisms for IT Management Process Representation.* In *Information Technology Management from a Business Perspective*, pages p. 45–54, Vancouver, Canada, April 2006. 1st IEEE/IFIP International Workshop on Business-Driven IT Management, IEEE.
- [DaSa 05] DANCIU, V. and M. SAILER: *A monitoring architecture supporting service management data composition.* In *Proceedings of the 12th Annual Workshop of HP OpenView University Association*, number 972–9171–48–3, pages 393–396, Porto, Portugal, July 2005. HP.
- [DDLS 00] DAMIANOU, N., N. DULAY, E. LUPU and M. SLOMAN: *Ponder: A language for Specifying Security and Management Policies for Distributed Systems. The Language Specification Version 2.3.* Imperial College Research Report DoC 2000/1, Imperial College of Science, Technology and Medicine, University of London, Department of Computing, October 2000.
- [DDLS 01] DAMIANOU, N., N. DULAY, E. LUPU and M. SLOMAN: *The Ponder Policy Specification Language.* In *Proceedings of the Workshop on Policies for Distributed Systems and Networks (Policy 2001)*, pages 18–39, Bristol, UK, January 2001. Springer-Verlag.

## BIBLIOGRAPHY

- [DEIS 05] DEISA KONSORTIUM: *DEISA Integrated Infrastructure Initiative, Annex I - Description of Work*. Sixth Framework Programme (Research Infrastructures, Communication network Development), Contract FP6 - 508830, December 2005.
- [DgS 07] DANCIU, V., N. GENTSCHEN FELDE and M. SAILER: *Declarative specification of service management attributes*. In *Moving From Bits to Business Value: Proceedings of the 2007 Integrated Management Symposium*, volume 2007, München, May 2007. IFIP/IEEE.
- [DHHS 06] DANCIU, V., A. HANEMANN, H.-G. HEGERING and M. SAILER: *IT Service Management: Getting the View*. In Kern, E. M., Hegering, H.-G. und Brügge, B. (Hrsg): *Managing Development and Application of Digital Technologies*, volume 2006, pages 110–130. Springer-Verlag, München, Germany, June 2006.
- [DSP 0108b] DMTF POLICY WORKING GROUP: *CIM Core Policy Model White Paper*. White Paper, June 2003.
- [FKT 01] FOSTER, IAN, CARL KESSELMAN and STEVEN TUECKE: *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*. International Journal of High Performance Computing Applications, 15(3):200–222, 2001.
- [GB921] *enhanced Telecom Operations Map (eTOM), The Business Process Framework For The Information and Communications Services Industry*, June 2002.
- [GB922-0] *Shared Information/Data (SID) Addendum 0 – SID Primer*, January 2004.
- [GB922] *Shared Information/Data (SID) Addendum 4SO – Service Overview Business Entity Definitions*, August 2004.
- [GHKR 01] GARSCHHAMMER, M., R. HAUCK, B. KEMPTER, I. RADISIC, H. ROELLE and H. SCHMIDT: *The MNM Service Model — Refined Views on Generic Service Management*. Journal of Communications and Networks, 3(4):297–306, December 2001.

- [GLI 07] *gLite - Lighthouse Middleware for Grid Computing*. <http://glite.web.cern.ch/glite/>, January 2007.
- [GT 07] *The Globus Toolkit*. <http://www.globus.org/>, January 2007.
- [HAN 99] HEGERING, H.-G., S. ABECK and B. NEUMAIR: *Integrated Management of Networked Systems – Concepts, Architectures and their Operational Application*. Morgan Kaufmann Publishers, ISBN 1-55860-571-1, 1999.
- [Holl 95] HOLLINGSWORTH, DAVID: *The Workflow Reference Model*. Workflow Management Coalition Specification TC00-1003, Workflow Management Coalition, January 1995. Issue 1.1.
- [Holl 04] HOLLINGSWORTH, DAVID: *The workflow reference model – 10 years on*, pages 295–312. Future Strategies Inc., Lighthouse Point, FL, USA, 2004 edition, 2004.
- [Homm 05] HOMMEL, W.: *An Architecture for Privacy-Aware Inter-Domain Identity Management*. In *16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2005)*, Barcelona, Spain, October 2005. Springer.
- [ITIL 00] OFFICE OF GOVERNMENT COMMERCE (OGC) (editor): *Service Support*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2000.
- [ITIL 01] OFFICE OF GOVERNMENT COMMERCE (OGC) (editor): *Service Delivery*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2001.
- [ITIL 02a] OFFICE OF GOVERNMENT COMMERCE (OGC) (editor): *ICT Infrastructure Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2002.
- [ITIL 02b] OFFICE OF GOVERNMENT COMMERCE (OGC) (editor): *Application Management*. IT Infrastruc-

## BIBLIOGRAPHY

- ture Library (ITIL). The Stationary Office, Norwich, UK, 2002.
- [ITIL 03] OFFICE OF GOVERNMENT COMMERCE (OGC) (editor): *Software Asset Management*. IT Infrastructure Library (ITIL). The Stationary Office, Norwich, UK, 2003.
- [ITU M.3010] *Principles for a Telecommunications management network*. Recommendation M.3010, ITU, May 1996.
- [KeDa 05] KEMPTER, B. and V. DANCIU: *Generic policy conflict handling using a priori models*. In *Lecture Notes in Computer Science*, volume October 24–26, 2005 of *Ambient Networks: 16th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, DSOM 2005*, pages 84–96, Barcelona, Spain, October 2005. Springer-Verlag.
- [Kemp 04] KEMPTER, B.: *Konfliktbehandlung im policy-basierten Management mittels a priori Modellierung*. PhD thesis, Ludwig-Maximilians-Universität München, August 2004.
- [KFJ 03] KAGAL, LALANA, TIM FININ and ANUPAM JOSHI: *A Policy Language for A Pervasive Computing Environment*. In *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, June 2003.
- [KHN<sup>+</sup> 07] KARMARKAR, ANISH, MARC HADLEY, HENRIK FRYSTYK NIELSEN, NOAH MENDELSON, YVES LAFON, JEAN-JACQUES MOREAU and MARTIN GUDGIN: *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. W3C Recommendation, World Wide Web Consortium (W3C), April 2007. <http://www.w3.org/TR/2007/REC-soap12-part2-20070427/>.
- [Koch 96] KOCH, T.: *Automated Management of Distributed Systems*. PhD thesis, 1996.
- [KoLo 99] KOHLI, MADHUR and JORGE LOBO: *Policy Based Management of Telecommunication Networks*. In *Policy Workshop 1999*, Bristol, U.K., 1999. .

- [LaMi 07] LAFON, YVES and NILO MITRA: *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C Recommendation, World Wide Web Consortium (W3C), April 2007. <http://www.w3.org/TR/2007/REC-soap12-part0-20070427/>.
- [LBN 99] LOBO, JORGE, RANDEEP BHATIA and SHAMIM A. NAQVI: *A Policy Description Language*. In *AAAI/IAAI*, pages 291–298, 1999, [citeseer.ist.psu.edu/lobo99policy.html](http://citeseer.ist.psu.edu/lobo99policy.html) .
- [LuSl 99] LUPU, EMIL C. and MORRIS SLOMAN: *Conflicts in Policy-Based Distributed Systems Management*. IEEE Transactions on Software Engineering, 25(6):852–869, November 1999.
- [Marc 05] MARCU, P. G.: *Reference Installation of the Ponder Policy Toolkit*. Systementwicklungsprojekt, Ludwig–Maximilians–Universität München, April 2005.
- [MML<sup>+</sup> 07] MENDELSON, NOAH, JEAN-JACQUES MOREAU, YVES LAFON, MARC HADLEY, MARTIN GUDGIN, ANISH KARMARKAR and HENRIK FRYSTYK NIELSEN: *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C Recommendation, World Wide Web Consortium (W3C), April 2007. <http://www.w3.org/TR/2007/REC-soap12-part1-20070427/>.
- [OMG 02-01-01] *OMG-XML Metadata Interchange (XMI) Specification, v1.2*. OMG Specification formal/02-01-01, Object Management Group, January 2002, <ftp://ftp.omg.org/pub/docs/formal/02-01-01.pdf> .
- [OMG 04-03-12] *Common Object Request Broker Architecture: Core Specification*. Adopted specification, Object Management Group, March 2004.
- [Radi 02d] RADISIC, I.: *Using Policy-Based Concepts to Provide Service Oriented Accounting Management*. In STADLER, R. and M. ULEMA (editors): *Proceedings of the 8th International IFIP/IEEE Network Operations and Management Symposium (NOMS*

## BIBLIOGRAPHY

- 2002), pages 313–326, Florence, Italy, April 2002. IFIP/IEEE, IEEE Publishing.
- [Radi 03] RADISIC, I.: *Ein prozessorientierter, policy-basierter Ansatz für ein integriertes, dienstorientiertes Abrechnungsmanagement*. PhD thesis, Ludwig-Maximilians-Universität München, February 2003.
- [RFC 3060] MOORE, B., E. ELLESSON, J. STRASSNER and A. WESTERINEN: *RFC 3060: Policy Core Information Model – Version 1 Specification*. Request For Comments, IETF, February 2001, <ftp://ftp.isi.edu/in-notes/rfc3060.txt> .
- [RFC 3512] MACFADEN, M., D. PARTAIN, J. SAPERIA and W. TACKABURY: *RFC 3512: Configuring Networks and Devices with Simple Network Management Protocol (SNMP)*. Request For Comments, IETF, April 2003, <ftp://ftp.isi.edu/in-notes/rfc3512.txt> .
- [Sail 07] SAILER, MARTIN: *Konzeption einer Service-MIB – Analyse und Spezifikation dienstorientierter Managementinformationen*. Dissertation, Fakultät für Mathematik, Informatik und Statistik der Ludwig-Maximilians-Universität München, München, 2007.
- [Sche 99] SCHEER, A.-W.: *ARIS – Business Process Modeling*. Springer-Verlag, Berlin, Zweite edition, 1999.
- [ScTh 05] SCHEER, AUGUST-WILHELM and O. THOMAS: *Geschäftsprozessmodellierung mit der ereignisgesteuerten Prozesskette*. 34(8-9), 2005.
- [Stra 04] STRASSNER, JOHN: *Policy based network management*. Morgan Kaufmann Publishers, 2004.
- [UML2d] *UML 2.0 Diagram Interchange Specification*. OMG Adopted Specification ptc/03-09-01, Object Management Group, September 2003.
- [UML2i] *UML 2.0 Infrastructure Specification*. OMG Adopted Specification 03-09-15, Object Management Group, September 2003.

- [UML2o] *UML 2.0 OCL Specification*. Object Management Group Adopted Specification ptc/03-10-14, Object Management Group, October 2003.
- [UML2s] *UML 2.0 Superstructure Specification*. OMG Adopted Specification ptc/03-08-02, Object Management Group, August 2003.
- [UNI 07] *UNICORE - Uniform Interface to Computing Resources*. <http://www.unicore.org/>, January 2007.
- [Univ 95] UNIVERSITY, CORPORATE CARNEGIE MELLON: *The capability maturity model: guidelines for improving the software process*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [Whit ] WHITE, STEPHEN A.: *Process Modeling Notations and Workflow Patterns*. White paper.
- [Whit 04] WHITE, STEPHEN A.: *Business Process Modeling Notation (BPMN) Version 1.0*. Technical Report, Business Process Management Initiative, May 2004.
- [Wiel 03] WIELEMAKER, JAN: *An overview of the SWI-Prolog Programming Environment*. In MESNARD, FRED and ALEXANDER SEREBENIK (editors): *Proceedings of the 13th International Workshop on Logic Programming Environments*, pages 1–16, Heverlee, Belgium, December 2003. Katholieke Universiteit Leuven.
- [Wies 95] WIES, R.: *Policies in Integrated Network and Systems Management: Methodologies for the Definition, Transformation, and Application of Management Policies*. PhD thesis, Ludwig-Maximilians-Universität München, June 1995.
- [XACML] MOSES, TIM (ED.): *eXtensible Access Control Markup Language (XACML) Version 2.0*. OASIS Standard oasis-access.control-xacml-2.0-core-specos, OASIS, February 2005.
- [XMLS-0] FALLSIDE, D. C. (EDITOR): *XML Schema Part 0: Primer*. W3C Recommendation REC-xmlschema-

## BIBLIOGRAPHY

- 0-20010502, World Wide Web Consortium (W3C), May 2001, <http://www.w3.org/TR/xmlschema-0/> .
- [XMLS-1] THOMPSON, H. S., D. BEECH, M. MALONEY and N. (EDS.) MEHDELSON: *XML Schema Part 1: Structures*. W3C Recommendation REC-xmlschema-1-20010502, World Wide Web Consortium (W3C), May 2001, <http://www.w3.org/TR/xmlschema-1/> .
- [XMLS-2] BIRON, P. V. and A. (EDS.) MALHOTRA: *XML Schema Part 2: Datatypes*. W3C Recommendation REC-xmlschema-2-20010502, World Wide Web Consortium (W3C), May 2001, <http://www.w3.org/TR/xmlschema-2/> .
- [XPD 05] *Process Definition Interface – XML Process Definition Language*. Workflow Management Coalition Workflow Standard WFMC-TC-1025, Workflow Management Coalition, October 2005. Version 2.0.



# XML-based grammar of the ProPoliS language

This appendix contains the XML schema of the ProPoliS policy language. Though the language has been analysed in Chapter 4, a complete grammar was not given there. In the following, the schema elements are defined.

Listing 1: ProPoliS Language Schema

```
<?xml version="1.0" encoding="UTF-8"?>
2
<xsd:schema
4     xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="pdl"
6     xmlns:pdl="http://www.nm.informatik.uni-muenchen.de/pdl">
  <xsd:annotation>
8     <xsd:documentation xml:lang="de">
       Sprache zur Definition von Policies.
10      Vitalian A. Danciu, 2002
       Version 0.7
12    </xsd:documentation>
  </xsd:annotation>
14
16  <!-- ROOT ELEMENTS AND THEIR TYPES -->
  <xsd:element name="policySet" type="policySetType"/>
18  <xsd:element name="comment" type="xsd:string"/>
20
  <!-- The elements that MAY be defined standalone in document instances -->
  <xsd:complexType name="policySetType">
22    <xsd:sequence>
      <xsd:element name="policy" type="policyType"
24         minOccurs="1" maxOccurs="unbounded"/>
      <xsd:element name="roleDefinition" type="roleDefinitionType"
26         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="subject" type="entityContainer"
28         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="target" type="entityContainer"
30         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="event" type="eventType"
32         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="action" type="actionType"
34         minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element name="constraint" type="constraintType"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

## ProPolis Schema

```
36         minOccurs="0" maxOccurs="unbounded"/>
37     <xsd:element name="policyGroup" type="group"
38         minOccurs="0" maxOccurs="unbounded"/>
39     <xsd:element ref="comment" minOccurs="0"/>
40     </xsd:sequence>
41 </xsd:complexType>
42
43 <!-- END ROOT ELEMENTS AND THEIR TYPES -->
44
45
46 <!-- BASE TYPES -->
47
48 <!-- Abstract base type for all nodes that are referable by ID.
49     These MAY be library items. -->
50 <xsd:complexType name="referable" abstract="true">
51     <xsd:sequence>
52         <xsd:element name="id" type="xsd:ID"/>
53         <xsd:element ref="comment" />
54         <xsd:attribute name="libraryItem" type="xsd:boolean"
55             use="optional" default="false"/>
56     </xsd:sequence>
57 </xsd:complexType>
58
59 <!-- Referable items can be assigned to one or more processes.
60     A list of valid process names is embedded into the type definition. -->
61 <xsd:complexType name="processAssignable" abstract="true">
62     <xsd:complexContent>
63         <xsd:extension base="pdl:referable">
64             <xsd:sequence>
65                 <xsd:element name="relatedProcess">
66                     <xsd:simpleType>
67                         <xsd:union>
68
69                             <!-- A list of processes that apply.
70                                 Process names MUST NOT contain whitespace. -->
71                             <xsd:simpleType>
72                                 <xsd:list itemType="xsd:String">
73                                     <xsd:restriction base="xsd:string">
74                                         <xsd:enumeration value="accounting"/>
75                                         <xsd:enumeration value="charging"/>
76                                         <xsd:enumeration value="billing"/>
77                                         <xsd:enumeration value="change"/>
78                                         <xsd:enumeration value="deployMeters"
79                                             />
80                                         <xsd:enumeration value="
81                                             pushmanagement"/>
82                                         <!-- TODO: complete list -->
83                                     </xsd:restriction>
84                                 </xsd:list>
85                             </xsd:simpleType>
86
87                             <!-- MEANING: this item is related to all processes -->
88                             <xsd:simpleType>
89                                 <xsd:restriction base="xsd:string">
90                                     <xsd:enumeration value="ALL"/>
91                                 </xsd:restriction>
92                             </xsd:simpleType>
```

```

92         <xsd:restriction base="xsd:string">
93             <xsd:enumeration value="OTHER"/>
94         </xsd:restriction>
95     </xsd:simpleType>
96 </xsd:union>
97 </xsd:simpleType>
98 </xsd:element>
99 </xsd:sequence>
100 </xsd:extension>
101 </xsd:complexContent>
102 </xsd:complexType>
103
104
105
106 <xsd:complexType name="abstractValueType" abstract="true">
107     <xsd:attribute name="type">
108         <xsd:simpleType>
109             <xsd:restriction base="xsd:token">
110                 <xsd:enumeration value="float"/>
111                 <xsd:enumeration value="integer"/>
112                 <xsd:enumeration value="string"/>
113                 <xsd:enumeration value="boolean"/>
114                 <xsd:enumeration value="dateTime"/>
115             </xsd:restriction>
116         </xsd:simpleType>
117     </xsd:attribute>
118 </xsd:complexType>
119
120 <!-- END BASE TYPES -->
121
122
123
124 <!-- STANDALONE COMPONENT TYPES -->
125
126 <!-- A group is a list of references to items of the same type.
127      Groups themselves CANNOT be referenced.
128
129      There is one generic group defined for grouping all
130      sorts of elements like policies, targets, events etc.
131      Beware: this implies that the consistency of the group
132      content CANNOT be assured by the parser, i. e. a targetSet
133      containig a group consisting of actions and events
134      DOES NOT violate this schema !!
135      Such anomalies MUST be checked by the interpreter. -->
136 <xsd:complexType name="group">
137     <xsd:sequence>
138         <xsd:element name="groupItems" type="xsd:IDREFS"/>
139         <xsd:attribute name="itemsType" use="required">
140             <xsd:simpleType>
141                 <xsd:restriction base="xsd:string">
142                     <xsd:enumeration value="target"/>
143                     <xsd:enumeration value="event"/>
144                     <xsd:enumeration value="action"/>
145                     <xsd:enumeration value="policy"/>
146                 </xsd:restriction>
147             </xsd:simpleType>

```

## ProPolis Schema

```
148     </xsd:attribute>
149   </xsd:sequence>
150 </xsd:complexType>
151
152 <!-- This complexType is defined for the SUBJECT and
153 TARGET elements that have many similarities -->
154 <xsd:complexType name="entityContainer">
155   <xsd:sequence>
156     <xsd:choice>
157       <xsd:sequence>
158         <xsd:element name="domain" type="domainType"
159           "/>
160         <xsd:element name="entity" type="xsd:string"
161           />
162       </xsd:sequence>
163       <xsd:element name="role" type="xsd:IDREF"/>
164     </xsd:choice>
165   </xsd:sequence>
166 </xsd:complexType>
167
168 <!-- The domain type is just a string containing an absolute
169 or relative path e.g. "/a/bxx/cx" or "axxx/bx/cxx".
170 It is encapsulated in a named type of its own to allow for extensibility
171 . -->
172 <xsd:simpleType name="domainType">
173   <xsd:restriction base="xsd:token">
174     <xsd:pattern value="/(?([0-9a-zA-Z]){1,})"/>
175   </xsd:restriction>
176 </xsd:simpleType>
177
178 <!--
179 In EBNF:
180 pattern ::= [ '/' ] { ('0'|'1'|'..'9'|'a'|'b'|'..'z'|'A'|'B'|'..'Z') '/' }
181 -->
182 </xsd:restriction>
183 </xsd:simpleType>
184
185 <!-- Roles are to be defined separately using this type.
186 When actually used, an existing role MUST be referenced
187 using an element of the type roleType -->
188 <xsd:complexType name="roleDefinitionType">
189   <xsd:complexContent>
190     <xsd:extension base="pdl:referable">
191       <xsd:sequence>
192         <xsd:element name="name" type="xsd:token"/>
193         <xsd:element name="description" type="xsd:string"/>
194       </xsd:sequence>
195     </xsd:extension>
196   </xsd:complexContent>
197 </xsd:complexType>
198
199 <!-- When assigning a role to an item, the role MUST already
200 be defined so it can be assigned by using its ID-reference.
201 One or more roles can be assigned to the same item. -->
202 <xsd:simpleType name="roleType">
203   <xsd:restriction base="xsd:IDREFS"/>
204 </xsd:simpleType>
```

```

202 <!-- Targets can be single ones or reference to a group of targets. -->
<xsd:complexType name="targetSetType">
204   <xsd:sequence>
       <xsd:choice>
206         <xsd:element name="target" type="entityContainer"/>
         <xsd:element name="targetRef" type="xsd:IDREF"/>
208         <xsd:element name="targetGroup" type="group" />
       </xsd:choice>
210   </xsd:sequence>
</xsd:complexType>
212
<!-- Events can be single ones or a reference to a group of events -->
214 <xsd:complexType name="eventSetType">
   <xsd:sequence>
216     <xsd:choice>
         <xsd:element name="event" type="eventType"/>
218         <xsd:element name="eventRef" type="xsd:IDREF"/>
         <xsd:element name="eventGroup" type="group"/>
220     </xsd:choice>
     </xsd:sequence>
222 </xsd:complexType>
224
<!-- A single event with a name -->
<xsd:complexType name="eventType">
226 <xsd:complexContent>
   <xsd:extension base="pdl:processAssignable">
228     <xsd:sequence>
         <xsd:element name="eventName" type="xsd:Name"/>
230     </xsd:sequence>
   </xsd:extension>
232 </xsd:complexContent>
</xsd:complexType>
234
<!-- An action set contains one action or a group of actions. -->
236 <xsd:complexType name="actionSetType">
   <xsd:sequence>
238     <xsd:choice>
         <xsd:element name="action" type="actionType"/>
240         <xsd:element name="actionRef" type="xsd:IDREF"/>
         <xsd:element name="actionGroup" type="group"/>
242     </xsd:choice>
   </xsd:sequence>
244 </xsd:complexType>
246
<!-- An action consists of a generic action to be executed
and an optional error action to be executed if the genericAction fails.
248 Alternatively, an event can be generated and propagated -->
<xsd:complexType name="actionType">
250 <xsd:complexContent>
   <xsd:extension base="pdl:processAssignable">
252     <xsd:sequence>
         <xsd:choice>
254           <xsd:sequence>
               <xsd:element name="default" type="genericActionType"
               />
256           <xsd:element name="error" type="genericActionType"
               minOccurs="0"/>

```

## ProPolis Schema

```
258         </xsd:sequence>
        </xsd:choice>
260     </xsd:sequence>
    </xsd:extension>
262 </xsd:complexContent>
</xsd:complexType>
264

266 <!-- END STANDALONE COMPONENT TYPES -->

268 <!-- BUILDING BLOCKS FOR STANDALONE COMPONENTS -->

270
<xsd:complexType name="binaryLogicalOperator">
272     <xsd:sequence>
        <xsd:element name="constraint" type="constraintType"
274             maxOccurs="unbounded"/>
        </xsd:sequence>
276 </xsd:complexType>

278 <!-- A constraint set contains either a single condition or
    an expression in either conjunctive or disjunctive normal form.
280 -->
<xsd:complexType name="constraintSetType">
282     <xsd:sequence>
        <xsd:choice>
284             <xsd:element name="constraintCNF">
286                 <xsd:complexType>
                    <xsd:sequence>
288                         <xsd:element name="or" type="
                            binaryLogicalOperator"
                            maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
290             </xsd:element>
292             <xsd:element name="constraintDNF">
294                 <xsd:complexType>
                    <xsd:sequence>
296                         <xsd:element name="and" type="
                            binaryLogicalOperator"
                            maxOccurs="unbounded"/>
                    </xsd:sequence>
                </xsd:complexType>
300             </xsd:element>
302             <xsd:element name="constraint" type="constraintType" />
304             <xsd:element name="constraintRef" type="xsd:IDREF"/>
        </xsd:choice>
306     </xsd:sequence>
</xsd:complexType>
308

<!-- A constraint is an expression that can be a unary or binary predicate
    . -->
310 <xsd:complexType name="constraintType">
    <xsd:choice>
```

```

312     <xsd:element name="equal" type="binaryPredicate"/>
313     <xsd:element name="smaller" type="binaryPredicate"/>
314     <xsd:element name="greater" type="binaryPredicate"/>
315     <xsd:element name="greaterEqual" type="binaryPredicate"/>
316     <xsd:element name="smallerEqual" type="binaryPredicate"/>
317     <xsd:element name="predicate" type="valueType"/>
318     <xsd:element name="negatedPredicate" type="valueType"/>
    </xsd:choice>
320 </xsd:complexType>

322 <xsd:complexType name="binaryPredicate">
    <xsd:sequence>
324     <xsd:element name="value" type="valueType"
        minOccurs="2" maxOccurs="2"/>
    </xsd:sequence>
326 </xsd:complexType>

328 <xsd:complexType name="genericActionType">
330     <xsd:sequence>
    <xsd:choice>
332     <xsd:element name="invoke" type="invocation"/>
333     <xsd:element name="generateEvent">
334         <xsd:complexType>
            <xsd:sequence>
336                 <xsd:element name="eventName" type="
                    xsd:string"/>
                    <xsd:element name="parameterSet" type
                        ="parameterSetType"
                            minOccurs="0"/>
            </xsd:sequence>
338         </xsd:complexType>
        </xsd:choice>
340     </xsd:sequence>
    </xsd:element>
342     <!--</xsd:sequence-->
    </xsd:choice>
344 </xsd:sequence>
</xsd:complexType>
346

348 <!-- A method invocation consists of an optional object,
349 a method name and an optional parameterSet.
350 The object can either be a Name or a <subject/> element;
351 if it is omitted, the targets of the policy are used as object. -->
352 <xsd:complexType name="invocation">
    <xsd:sequence>
354     <xsd:choice>
        <xsd:element name="object" type="xsd:Name" minOccurs
            ="0"/>
356     <xsd:element name="object">
        <xsd:complexType>
358             <element name="subject" empty="true"
                minOccurs="0" />
        </xsd:complexType>
    </xsd:choice>
360     </xsd:element>
    <xsd:element name="method" type="xsd:Name"/>
362     <xsd:element name="parameterSet" type="parameterSetType"
        minOccurs="0"/>
    </xsd:choice>

```

## ProPolis Schema

```
364         </xsd:sequence>
</xsd:complexType>
366
<!-- A parameter set is a set of name-value pairs. -->
368 <xsd:complexType name = "parameterSetType">
  <xsd:element name = "parameter">
370    <xsd:complexType>
      <xsd:element name="paramName" type = "xsd:token"/>
372      <xsd:element name="value" type="valueType"/>
    </xsd:complexType>
374  </xsd:element>
</xsd:complexType>
376
<!-- A value is one of float, int, string, boolean or ISO dateTime.
378 It can be either
  - a constant/literal, or
380 - a value retrieved from an attribute of an object, or
  - a return value of a method invocation
382 Arrays of constants are also values.
-->
384 <xsd:complexType name="valueType">
  <xsd:complexContent>
386    <xsd:extension base="abstractValueType">
      <xsd:choice>
388        <xsd:element name="literal">
          <xsd:simpleType>
390            <xsd:union memberTypes="xsd:float,xsd:integer,xsd:string,
              xsd:boolean,xsd:dateTime"/>
          </xsd:simpleType>
392        </xsd:element>
          <xsd:element name="array" type="arrayType"/>
394          <xsd:element name="functionValue" type="genericActionType"
            />
          <xsd:element name="attributeValue">
396            <xsd:complexType>
              <xsd:sequence>
398                <xsd:element name="object" type="xsd:
                  Name"/>
400                <xsd:element name="attribute" type="
                  xsd:Name"/>
              </xsd:sequence>
402            </xsd:complexType>
          </xsd:element>
396        </xsd:choice>
      </xsd:extension>
406 </xsd:complexContent>
</xsd:complexType>
408
<!-- An array is a list of items of the same type.
410 A string array is a whitespace delimited list of strings. -->
<xsd:complexType name="arrayType">
412 <xsd:sequence>
  <xsd:element name="arrayContent">
414    <xsd:simpleType>
      <xsd:union>
416        <xsd:simpleType><xsd:list itemType="xsd:float"/></xsd:
```



```

simpleType>
<xsd:simpleType><xsd:list itemType="xsd:integer"/></xsd:
418 simpleType>
<xsd:simpleType><xsd:list itemType="xsd:string"/></xsd:
simpleType>
<xsd:simpleType><xsd:list itemType="xsd:boolean"/></xsd:
420 simpleType>
<xsd:simpleType><xsd:list itemType="xsd:dateTime"/></xsd:
simpleType>
<xsd:simpleType>
422   <xsd:restriction base="xsd:string">
       <xsd:enumeration value="null"/>
424   </xsd:restriction>
   </xsd:simpleType>
426 </xsd:union>
</xsd:simpleType>
428 </xsd:element>
</xsd:sequence>
430
</xsd:complexType>
432
<!-- END BUILDING BLOCKS FOR STANDALONE COMPONENTS -->
434
436
<!-- POLICY -->
438
440 <xsd:complexType name="policyType">
  <xsd:complexContent>
442   <xsd:extension base="pdl:processAssignable">
     <xsd:sequence>
444       <xsd:element name="policyDomain" type="domainType" minOccurs="0"
          />
       <xsd:element name="subject" type="entityContainer" minOccurs="0"
          />
446       <xsd:element name="targetSet" type="targetSetType"/>
       <xsd:element name="eventSet" type="eventSetType"/>
448       <xsd:element name="constraintSet" type="constraintSetType"
          minOccurs="0"/>
       <xsd:element name="actionSet" type="actionSetType"/>
450       <xsd:element name="policyDescriptor">
         <xsd:complexType>
452           <xsd:sequence>
             <xsd:element name="createdBy" type="xsd:token" minOccurs="
454               0"/>
             <xsd:element name="dateOfCreation" type="xsd:date"
               minOccurs="0"/>
             <xsd:element name="lastModified" type="xsd:dateTime"
               minOccurs="0"/>
456             <xsd:element name="lastModifiedBy" type="xsd:token"
               minOccurs="0"/>
             <xsd:element name="expires" type="xsd:dateTime" minOccurs="8"
               0"/>
           </xsd:sequence>
         </xsd:complexType>
460       </xsd:element>

```

## ProPolis Schema

```
462     </xsd:sequence>
463   <xsd:attribute name="isEnabled" type="xsd:boolean"
464     use="optional" default="true"/>
465   <xsd:attribute name="priority" use="optional" default="0">
466     <xsd:simpleType>
467       <xsd:restriction base="xsd:integer">
468         <xsd:minInclusive value="0"/>
469         <xsd:maxInclusive value="99"/>
470       </xsd:restriction>
471     </xsd:simpleType>
472   </xsd:attribute>
473   <xsd:attribute name="version" type="xsd:float"
474     use="optional" default="1.0"/>
475 </xsd:extension>
476 </xsd:complexContent>
477 </xsd:complexType>
478 </xsd:schema>
```

---

# Grammar of the Service Information Specification Language (SISL)

The necessity of a formalism to specify information items within the data flow of a process has been discussed in Chapter 5. The SISL has been identified as a candidate that fulfils the requirements imposed by the message-based information transport employed in distributed, policy-based process execution. However, SISL has only been outlined in that context. This annex provides the specification of its XML-based grammar.

Listing 2: SISL Schema

```

1 <?xml version="1.0" encoding="ISO-8859-1"?>
2 <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
3   <xs:element name="resource" type="resource"/>
4   <xs:complexType name="resource">
5     <xs:sequence>
6       <xs:element name="source" type="xs:string"/>
7       <xs:element name="identifier" type="identifier"/>
8       <xs:element name="description" type="description"/>
9       <xs:element name="sourceAttrib" type="sourceAttrib"
10        />
11     </xs:sequence>
12   </xs:complexType>
13   <xs:element name="function" type="function"/>
14   <xs:complexType name="function">
15     <xs:sequence>
16       <xs:element name="method" type="xs:string"/>
17       <xs:element name="return" type="xs:string"/>
18       <xs:element name="identifier" type="identifier"/>
19       <xs:element name="description" type="description"/>
20       <xs:element name="parameters" type="parameters"/>
21     </xs:sequence>
22   </xs:complexType>
23   <xs:element name="identifier" type="identifier"/>
24   <xs:complexType name="identifier">
25     <xs:sequence>
26       <xs:element name="id" type="xs:string"/>
27     </xs:sequence>
28   </xs:complexType>
29   <xs:element name="sourceAttrib" type="sourceAttrib"/>
30   <xs:complexType name="sourceAttrib">
31     <xs:sequence>
```

```

32         <xs:element name="interval" type="xs:float"/>
33         <xs:element name="return" type="xs:string"/>
34         <xs:element name="identifier" type="identifier"/>
35     </xs:sequence>
36 </xs:complexType>
37 <xs:element name="parameters" type="parameters"/>
38 <xs:complexType name="parameters">
39     <xs:sequence>
40         <xs:element name="valueset" type="valueset"/>
41     </xs:sequence>
42 </xs:complexType>
43 <xs:element name="valueset" type="valueset"/>
44 <xs:complexType name="valueset">
45     <xs:sequence>
46         <xs:element name="resourceRef" type="resourceRef"/>
47         <xs:element name="functionRef" type="functionRef"/>
48     </xs:sequence>
49 </xs:complexType>
50 <xs:element name="resourceRef" type="resourceRef"/>
51 <xs:complexType name="resourceRef">
52     <xs:sequence>
53         <xs:element name="id" type="xs:string"/>
54     </xs:sequence>
55 </xs:complexType>
56 <xs:element name="functionRef" type="functionRef"/>
57 <xs:complexType name="functionRef">
58     <xs:sequence>
59         <xs:element name="id" type="xs:string"/>
60     </xs:sequence>
61 </xs:complexType>
62 <xs:element name="condition" type="condition"/>
63 <xs:complexType name="condition">
64     <xs:sequence>
65         <xs:element name="identifier" type="identifier"/>
66         <xs:element name="description" type="description"/>
67         <xs:element name="boolExpression" type="
68             boolExpression"/>
69     </xs:sequence>
70 </xs:complexType>
71 <xs:element name="declaration" type="declaration"/>
72 <xs:complexType name="declaration">
73     <xs:sequence>
74         <xs:element name="valueset" type="valueset"/>
75     </xs:sequence>
76 </xs:complexType>
77 <xs:element name="boolExpression" type="boolExpression"/>
78 <xs:complexType name="boolExpression">
79     <xs:sequence>
80         <xs:element name="expression" type="xs:string"/>
81     </xs:sequence>
82 </xs:complexType>
83 <xs:element name="aggregation" type="aggregation"/>
84 <xs:complexType name="aggregation">
85     <xs:sequence>
86         <xs:element name="resource" type="resource"/>
87         <xs:element name="function" type="function"/>
88         <xs:element name="description" type="description"/>

```

```

      <xs:element name="notification" type="notification"
        />
88     <xs:element name="identifiant" type="identifiant"/>
      </xs:sequence>
90   </xs:complexType>
  <xs:element name="description" type="description"/>
92  <xs:complexType name="description">
    <xs:sequence>
94      <xs:element name="author" type="xs:string"/>
96      <xs:element name="date" type="xs:string"/>
96      <xs:element name="text" type="xs:string"/>
    </xs:sequence>
98  </xs:complexType>
  <xs:element name="notification" type="notification"/>
100 <xs:complexType name="notification">
  <xs:sequence>
102      <xs:element name="condition" type="condition"/>
104      <xs:element name="declaration" type="declaration"/>
104      <xs:element name="description" type="description"/>
    </xs:sequence>
106 </xs:complexType>
</xs:schema>

```

---



# Index

- A**
- activity diagram ..... 73
  - agent ..... 16
    - in customer domain ..... 30
    - mail transfer ..... 141
    - management ..... 28, 29
    - mobile ..... 265
    - performance management . 29
    - SNMP ..... 118
  - Application Service Provider ... 26
  - architecture
    - components ..... 207, 227
    - interoperation ..... 215
  - ASP ..... 26
  - Availability Management ..... 67
- B**
- BEIDTF ..... 72
  - BPML ..... 72
  - BPML ..... 72
  - BPMN ..... 72
  - Business Modeling & Integration Domain Task Force ..... 72
  - business process ..... 9
  - Business Process Execution Language for Web Services .... 72
  - Business Process Management Initiative ..... 72
  - Business Process Modeling Language 72
  - Business Process Modeling Notation 72
- C**
- CA ..... 256
  - Capability Maturity Model .... 77
  - Capability Maturity Model Integration ..... 77
  - Capacity Management ..... 67
  - Certification Authority ..... 256
  - Change Management ..... 65
  - CI ..... 43
  - CIM Object Manager ..... 231
  - CIMOM ..... 231
  - CMDB ..... 43, 64, 69
  - CMM ..... 77
  - CMMI ..... 77
  - Configuration Item ..... 43
  - configuration item ..... 66
  - Configuration Management .... 66
  - Configuration Management Database 43, 64, 69
- D**
- Definitive Hardware Store ..... 65
  - Definitive Software Library ..... 65
  - DHS ..... 65
  - DII ..... 242
  - DSL ..... 65
  - Dynamic Invocation Interface . 242
- E**
- ebXML ..... 71
  - Electronic Business using eXtensible Markup Language .. 71
  - emergency change ..... 65
  - Enhanced Telecom Operations Map 15
  - eTOM ..... 15, 70
  - Extended Telecom Operations Map 70
  - eXtensible Access Control Modeling Language ..... 87
- F**
- Financial Management for IT Services ..... 68

## INDEX

Forward Schedule of Change ... 45  
FSC ..... 45

### G

generating system 96, 152, 161, 163

### I

IDS ..... 40  
IDS Scheer ..... 75  
Incident Management ..... 63  
Intrusion Detection System .... 40  
IT Infrastructure Library ..... 15  
IT management process ..... 10  
IT Service Capability Maturity Model  
78  
IT Service Continuity Management  
68  
IT Service Management ..... 7  
ITIL ..... 15, 63  
    process formalisation ..... 69  
    service delivery ..... 66  
    service support ..... 63  
ITSCMM ..... 78  
ITSM ..... 7

### K

Key Performance Indicator ..... 64  
known error ..... 64  
KPI ..... 64

### L

LDAP ..... 125, 208  
Lightweight Directory Access Pro-  
tocol .....  
208

### M

mail transfer agent ..... 141  
management  
    autonomic ..... 267  
    discipline ..... 11  
    self- ..... 267  
maturity model ..... 77  
mobile agent ..... 265  
MTA ..... 141

### N

New Generation Operations Systems  
    and Software ..... 70  
NGOSS ..... 70

### O

OASIS ..... 71  
Object Management Group .... 72  
Object Request Broker ..... 240  
OMG ..... 72  
ORB ..... 240  
Organization for the Advancement  
    of Structured Information  
    Standards ..... 71  
outsourcing ..... 26

### P

pattern ..... 78, 173  
    basic pattern ..... 143  
    catalogue ..... 143  
    condition pattern ..... 144  
    conjunctive ..... 147  
    disjunctive ..... 146  
    mixed ..... 148  
    synchronisation pattern .. 149  
    variants ..... 143  
PbM ..... 13, 80  
PCIM ..... 84  
PDP ..... 83  
PEP ..... 84  
PIR ..... 45, 65  
PKI ..... 34, 256  
policy  
    architecture ..... 84, 209  
    conflict ..... 82  
    hierarchy ..... 81  
    language ..... 85  
    refinement ..... 82  
    standardisation ..... 84  
    type ..... 81  
Policy Core Information Model. 84  
Policy Decision Point ..... 83  
Policy Execution Point ..... 84  
Policy-based Management .. 13, 62  
Post Implementation Review 45, 65  
privacy ..... 264  
Problem Management ..... 64  
process  
    change ..... 205  
    data flow ..... 181  
    detail metric ..... 264  
    formalism .... 13, 61, 71, 106  
    framework ..... 15, 61  
    implementation ..... 202  
    language ..... 61, 106



life-cycle ..... 11, 200  
 maturity ..... 77  
 operational ..... 63  
 oriented ..... 14  
 pattern ..... 61, 78  
 refinement ..... 203  
 support tool ..... 10, 266  
 profile ..... 73  
 ProPoliS ..... 238  
 Public Key Infrastructure . 34, 256

**R**

Real Organisation ..... 33  
 Release Management ..... 65  
 Remote Method Invocation ... 124  
 Request for Change ..... 45, 65  
 RfC ..... 45, 65  
 RMI ..... 124  
 RO ..... 33  
 rollback plan ..... 65

**S**

SAMM ..... 78  
 SAP ..... 28  
 security ..... 264  
 Security Management ..... 68  
 Service Access Point ..... 28  
 service desk ..... 63  
   virtual ..... 64  
 Service Information/Data ..... 70  
 Service Level Management .... 67  
 Service Oriented Architecture .. 19  
 Service Oriented Monitoring Archi-  
   tecture ..... 226  
 service request ..... 64  
 Service Level Agreement ..... 245  
 SID ..... 70  
 Simple Object Access Protocol 243  
 Single Point Of Contact ..... 26  
 SLA ..... 66, 245  
 SLM ..... 67  
 SMONA ..... 226  
 SNMP ..... 116, 123, 183, 243  
   agent ..... 118  
 SOA ..... 19  
 SOAP ..... 243  
 SPOC ..... 26, 63  
 substitution rule ..... 172  
 System Administration Maturity Model

78

**T**

tool  
   interface ..... 12  
   OSS ..... 266  
   process support ..... 10  
   service management ..... 12  
   system & network management  
     12  
 translation pattern ..... 143

**U**

UC ..... 29  
 UML ..... 72  
   activity diagram ..... 73  
   profile ..... 73  
 UN/CEFACT ..... 71  
 Underpinning Contract ..... 29  
 Unified Modeling Language .... 72  
 United Nations Centre for Trade Fa-  
   cilitation and Electronic  
   Business ..... 71

**V**

variants ..... 143  
 Virtual Organisation ..... 33  
 VO ..... 33

**W**

Web Service ..... 72  
 Web Service Definition Language ..  
   72  
 WfMC ..... 73  
 workaround ..... 64  
 workflow  
   language ..... *see*  
   process formalism  
 Workflow Management Coalition ..  
   73  
 WS ..... 72  
 WS-BPEL, BPEL4WS ..... 72  
 WSDL ..... 72

**X**

XACML ..... 87  
 XMI ..... 73  
 XML Model Interchange ..... 73  
 XML Process Definition Language  
   74  
 XPDL ..... 74