

Master Thesis

October 25, 2013

Modeling and Execution

of Operational Security Policies

Niklaus Meyer

of Illnau, Switzerland (04-862-967)

supervised by

Prof. Dr. Martin Glinz

Dr. Norbert Seyff

Dr. Jochen Küster

Dr. Hagen Völzer



**University of
Zurich**^{UZH}

Department of Informatics



Master Thesis

Modeling and Execution

of Operational Security Policies

Niklaus Meyer



University of
Zurich^{UZH}

Department of Informatics



Master Thesis

Author: Niklaus Meyer, meyer.niklaus@gmail.com

Project period: 01. May 2013 - 31. October 2013

Department of Informatics, University of Zurich

Disclaimer. *This thesis template is provided by the s.e.a.l. group (<http://seal.ifi.uzh.ch>) of the University of Zurich, Department of Informatics (ifi) as a service to ifi students on an “as-is, as-available” basis for (master/bachelor/diploma) thesis layouting purposes only. We assume no responsibility for any errors or omissions in the template and make no commitment to update the template on a regular basis.*

ifi students may download and use the thesis template for personal and non-commercial use only, without altering or removing any copyright or other notice from the template. For the proper use of the template, the Frutiger Condensed font of the University of Zurich has to be installed.

Thesis template © 2009, s.e.a.l – University of Zurich, Switzerland

Acknowledgements

I would like to thank all the people who were involved in this thesis. First of all, I would like to thank Prof. Dr. Martin Glinz, Dr. Hagen Völzer and Dr. Jochen Küster for giving me the opportunity to write this thesis at the IBM Research Lab and the Requirements Engineering Research Group of the University of Zurich.

Especially, I would like to thank Dr. Norbert Seyff and my advisors at IBM for their dedicated support and valuable feedback. Thanks for all the constructive discussions and inputs which allowed me to improve this thesis. Last but not least, I would like to thank Dr. Axel Tanner, Maria Dubovitskaya and Michael Osborne who supported me with their security knowledge and assisted during the validation.

Abstract

Information systems often have to fulfill requirements regarding their security. Many of these requirements cannot be addressed using technical measures alone. Organizations therefore rely on security policies which regulate how people work with their system and define the processes that are required to operate them. In practice, these policies are too often not followed correctly, which jeopardizes the security and reliability of the systems.

This thesis proposes a modeling language based on the UML and BPMN standards. The language is used by security experts to specify security objects, such as cryptographic keys and the processes that are required to manage them. The formal language aims to improve the quality of the policy documents which, in practice, are written predominantly in natural language. The thesis presents the policy support system, a workflow engine which enacts the models. This system assists users during the execution of a policy and thereby reduces the risk of human errors. In addition, it manages the lifecycle of security objects and notifies users about important events.

A validation of the language and the policy support system showed promising results. The interviewed security experts agree that the models enhance the quality of the policy documents and that the support system reduces the risk of human errors. A long-term study would however be required to evaluate whether our approach can reduce the number of security incidents in practice.

Zusammenfassung

Heutige Informationssysteme müssen oft verschiedene Anforderungen bezüglich ihrer Sicherheit erfüllen. Viele dieser Anforderungen können nicht durch technische Mittel alleine abgedeckt werden. Organisationen setzen daher auf Sicherheitsrichtlinien um festzulegen wie Personen mit den Systemen zu arbeiten haben. In der Praxis werden diese Richtlinien jedoch oft nicht befolgt, was zur Folge hat, dass die Sicherheit oder Zuverlässigkeit der Systeme gefährdet ist.

Diese Masterarbeit schlägt eine Modellierungssprache basierend auf dem UML und BPMN Standard vor. Sicherheitsexperten verwenden diese Sprache um Sicherheitsobjekte wie, zum Beispiel kryptographische Schlüssel und die dazugehörigen Prozesse zu spezifizieren. Der Einsatz einer formalen Sprache soll die Qualität heutiger Sicherheitsrichtlinien, welche in natürlicher Sprache verfasst sind, verbessern. Die Arbeit enthält das Policy Support System, welches in der Lage ist die Modelle auszuführen. Das System unterstützt Benutzer während der Ausführung der Sicherheitsrichtlinien und reduziert das Risiko von menschlichen Fehlern. Zusätzlich verwaltet das System den Lebenszyklus der Sicherheitsobjekte und meldet wichtige Ereignisse an die Benutzer.

Eine Validation der Sprache und des Systems zeigte vielversprechende Resultate; Sicherheitsexperten bestätigen, dass die Modelle die Qualität einer Sicherheitsrichtlinie verbessern und dass menschliche Fehler mit dem Policy Support System reduziert werden können. Eine Langzeitstudie ist jedoch notwendig um festzustellen ob sich diese Erkenntnisse auch auf die Praxis anwenden lassen.

Contents

1	Introduction	1
1.1	Motivation and Problem Statement	2
1.2	Thesis Objectives	3
1.3	Thesis Structure	4
2	Background and Related Work	5
2.1	Security Foundations	5
2.1.1	Data Encryption and Digital Certificates	5
2.1.2	Information Security Policy	7
2.2	Process and Information Modeling Foundations	8
2.2.1	Business Process Management	8
2.2.2	Information Modeling	10
2.2.3	Multi-View Modeling	11
2.3	Modeling Security Policies	12
2.4	Integrating Object Lifecycle and Workflow	13
2.4.1	Foundations of Artifact-Centric Process Modeling	13
2.4.2	Other Approaches to Model Data in Processes	14
2.4.3	Artifact-Centric Business Process Modeling	17
2.4.4	Benefits and Limitations of the Approaches	21
3	Requirements for an Integrated Modeling Framework	25
3.1	Usage Scenarios	25
3.1.1	Scenario 1: Manage Security Objects	26
3.1.2	Scenario 2: Execute Key Management Processes	27
3.2	Modeling Language	28
3.2.1	Modeling Security Objects	28
3.2.2	Using Security Objects in a Process	31
3.2.3	Interaction between Security Objects and Processes	32
3.3	Design and Execution Methodology	32
4	The Integrated Modeling Framework	33
4.1	Security Object Modeling Language (SOML)	33
4.1.1	Security Object Model	33
4.1.2	Key Management Processes	42
4.1.3	Execution Semantics Rationale	47
4.1.4	Requirements Covered by the Language	51
4.2	Designing and Enacting the Security Policy	52

4.2.1	Designing the Security Policy	52
4.2.2	Enacting the Security Policy	56
4.2.3	Monitoring and Controlling the Security Policy Execution	58
5	Implementation of the Integrated Modeling Framework	59
5.1	The Activiti BPM Platform	59
5.2	Use Cases of the Policy Support System	60
5.3	Architecture of the Policy Support System	61
5.3.1	Sub-Components of the Policy Support System	63
5.3.2	Process Engine API	64
5.4	The Policy Explorer	65
6	Validation of the Integrated Modeling Framework	67
6.1	Method of Validation	67
6.2	Case Study: The Tachograph System	68
6.3	Designing the Tachograph Security Policy	69
6.4	Enacting the Tachograph Security Policy	72
6.5	Discussion	73
6.5.1	Limitations of the Integrated Modeling Framework	75
6.5.2	Threats to the Validity of the Validation	76
7	Conclusion	77
7.1	Summary of Contributions	77
7.2	Future Work and Research	79
A	Security Object Modeling Language Requirements	81
A.1	Security Object Information Model	81
A.2	Security Object Lifecycle Model	84
A.3	Using Security Objects in a Process	86
A.4	Interaction between Security Objects and Processes	88
A.5	Initial Modeling Example	90
B	Specification and Implementation Details	95
B.1	Data Modeling Grammar	95
B.2	Process/Object Engine Database Schema	96
C	Validation	99
C.1	Interview Questionnaire	99
D	Content on the Delivered CD-ROM	103
	Glossary	105
	Acronyms	107

List of Figures

1.1	Different Aspects of Information Security [MvR10]	1
1.2	Example: Simplified Key Lifecycle According to the NIST [BBB ⁺ 12]	2
2.1	Symmetric and Asymmetric Cryptography	5
2.2	Example: Smart Card & Terminal	6
2.3	Example: Public-Key Infrastructure (PKI)	7
2.4	Policy Framework & CA Policy Documents [MD01]	7
2.5	BPM Lifecycle [DLRMR13]	9
2.6	Multi-View Modeling. [Wah09]	11
2.7	Integrating Object Lifecycle and Workflow: Timeline	13
2.8	Proclets and Interaction Graph	15
2.9	Proclets and Interaction Graph: Example	15
2.10	BPMN Data Modeling Elements	16
2.11	Configuration Based Release Processes (Corepro) Models and Configuration	17
2.12	Integrated Process and Object Life Cycle Modeling	18
2.13	WS-BPEL extended with BPEL4Data	19
2.14	BPMN extended with BPM4Data	20
2.15	Information, Lifecycle and Process Models	21
3.1	Combined and Separated Information Models	29
3.2	Key Lifecycle (As Recommended by the NIST [BBB ⁺ 12])	30
3.3	Relation between Lifecycle and Process Models	30
4.1	Security Object Modeling Language (SOML) Language Components	33
4.2	The Security Object Model	35
4.3	Security Object Modeling Language UML Profile	36
4.4	Meta-Model: Information Model	37
4.5	Meta-Model: Lifecycle Model	39
4.6	Meta Model: State Behavior	41
4.7	SOML Data Modeling Elements	42
4.8	BPMN Data Flow	43
4.9	Activiti Process Variable Scope	43
4.10	Data Object Collections and Multi-Instance Activities	44
4.11	Using the Business Process Model and Notation (BPMN) Error Boundary Event to Resolve an Inconsistent Change	45
4.12	Optimistic Offline Lock and Rollback	47
4.13	Obtaining an Explicit Lock in a Process Model (<i>Not Implemented</i>)	48
4.14	Implicit Control Flow (L) vs. Normal Control Flow (R)	50
4.15	BPM Process Identification and Design [DLRMR13]	52
4.16	Process Architecture [DLRMR13] Applied to Security Policies	53
4.17	Object/Function Matrix Evolving into a Policy Landscape	54
4.18	Iterative Design Process of Security Objects and Key Management Processes	55
4.19	Generic BPMS Architecture [DLRMR13] Mapped to the Policy Support System	57
4.20	Example: Security Object and Process History Data	58
5.1	Overview of the Activiti BPM Platform [Act]	59
5.2	Use Case Diagram of the Policy Support System	60
5.3	Architecture Overview (Standard Activiti Components in Gray)	62
5.4	Service Extension Introduced by the Policy Support System	64

5.5	View & Manage Security Policy Definitions	66
5.6	View Security Object Instances	66
6.1	Tachograph System	68
6.2	Tachograph Certificates	69
6.3	Tachograph Policy Landscape (Excerpt)	70
6.4	Create Key and Certificate Process Abstract Model	71
6.5	Transformation into an Executable Model	71
6.6	Task Assistance by the Policy Support System	72
7.1	Reduce Risks with Task Automation and Process Governance	79
A.1	Initial and External Triggers	85
A.2	Accepted State(s)	86
A.3	Produced State and Error Handling	87
A.4	Intermediate Conditional Event	89
A.5	Sequence Diagram: Send/Receive Signals	89
A.6	Information Model	90
A.7	MSCA Key–Pair & Certificate Behavior	90
A.8	Key Generation Process Overview	91
A.9	Key Generation Process Detail (1/2)	92
A.10	Key Generation Process Detail (2/2)	92
A.11	Key Compromise Process Detail	93
A.12	Key End of Life (Simplified)	93
B.1	Process/Object–Engine Database Schema	97

List of Tables

2.1	Summary of the Underlying Concepts	23
2.2	Legend for Table 2.3	23
2.3	Summary of Supported Information and Process Model Features.	24
3.1	Stakeholders of the Framework	25
3.2	Security Object Information Model Requirements	29
3.3	Security Object Lifecycle Model Requirements	31
3.4	Using Security Objects in a Process	31
3.5	Interaction between Security Objects and Processes	32
4.1	Supported Types of the Policy Support System	37
4.2	Different State Change Semantics	49
4.3	Traceability between Language Parts and Requirements	51
6.1	Security Object Candidates of the Tachograph Policy (Excerpt)	70
7.1	SOML Compared to the Approaches Shown in Table 2.3.	78
B.1	Process/Object Database Fields	97
C.1	Roles Used in the Questionnaire	99

List of Listings

4.1	Example: Access to a Security Object Instance in a Groovy Script	41
4.2	Example: Object Creation (<i>Formal Syntax is Defined in Section B.1</i>)	44
4.3	Example: Different Queries (<i>Formal Syntax is Defined in Section B.1</i>)	44
4.4	Example: State Change (<i>Formal Syntax is Defined in Section B.1</i>)	45
4.5	Example: Setting Attributes (<i>Formal Syntax is Defined in Section B.1</i>)	45
4.6	Example: Different Conditional Expressions (<i>See JUEL Language Specification</i>) . .	46
4.7	Example: Conditional Start Events (<i>Formal Syntax is Defined in Section B.1</i>) . . .	46
5.1	Example: Extension Mechanism of the Activiti Process Engine	62
5.2	Example: Process/Object Engine Embedded in a JUnit Test	65
B.1	SOML Grammar (1/3: Commands and Events)	95
B.2	SOML Grammar (2/3: Types)	95
B.3	SOML Grammar (3/3: Conditions)	96

Introduction

The emergence of new interaction paradigms between individuals, corporations and government organizations has changed the way information systems are used. In the past, an information system has mainly served a few users within an isolated environment. Today's e-Business and e-Government systems are increasingly interconnected with the rest of the world. They are connected to other systems that reside beyond organizational boundaries [Wie92, Erl08] and provide services to customers and partners [Alt01]. Therefore they are no longer operated in a fully controlled environment and an organization cannot assume that all participants act in the organizations best interest. The interaction with other systems and individuals resulted in a ubiquitous use of information systems but increased the concerns about their security [GL02].

Information systems often need to fulfill different constraints and quality requirements [Gli07] related to the security and robustness of the system and its data. Fulfilling these security requirements became a critical success factor for the acceptance of any system [BP83, LIC03]. Information security is concerned with protecting these systems from misuse and is often achieved using technical solutions, such as cryptography or firewalls [SS08, MVOV10]. However, it has been acknowledged that security cannot be achieved by technical measures alone [And01, MvR10, SS08, BHW06].



Figure 1.1: Different Aspects of Information Security [MvR10]

In addition to technology, information security is also affected by the people who use the systems and work according to predefined processes. In order to adequately protect a system, all aspects of information security shown in Figure 1.1 must be considered:

Technology Technology provides the security practitioners with the tools and algorithms to achieve the desired security qualities such as confidentiality, integrity or availability. While the technology element contains exceptional capability for addressing security weaknesses, the other elements must not be ignored [MvR10].

People	Most information systems are used by different stakeholders. Their experience, intention, behavior and values influence how they interact with the system and affects the overall security. To understand people, one must study the fields of psychology and sociology [SS08].
Process	Processes, policies and operating procedures define how an organization and its employees want activities to be completed. They help the organization to achieve its goals, while complying with the required security practices.
Organization	The organization or context in which the information system resides has a major influence on its security. The organization governs the processes, defines the culture of the employees and affects technology choices by propagating an enterprise architecture. Ultimately it is up to the organization, to decide what and how many resources are spent on security.

The root cause for many security incidents in practice are processes which are not being followed and human errors [SS08]. This thesis addresses the problems observed in the operational processes of information systems that produce electronic identity documents based on smart cards. These systems rely on cryptography to protect identity documents against counterfeiting. Public–Key Infrastructures (PKIs) take a key role in the protection of the systems since they ensure that cryptographic keys have been issued by a trusted authority.

1.1 Motivation and Problem Statement

The security of identity documents, smart cards or PKIs often requires protection of security relevant components by the operator of the system. The security can, for example rely on a cryptographic key that is required to access the system just as a normal key is used to open the entry door of a house. The strongest door or best cryptographic algorithm does not help if an adversary has access to the key. Likewise the system is not secure if the process of creating the key and distributing it to the owner is flawed. The primary concern of an operator is therefore often not the technological aspect of information security, but the processes and operating instruction that are required to set up and operate the systems.

Security policies regulate processes and operating instructions. They define how users interact with the security relevant components of an information system. At the core, these documents describe how users handle security objects, such as cryptographic keys or certificates that are critical for the security. The policy instructs users how to create, manage and dispose the security objects without jeopardizing the security of the system. A security object typically traverses a non–trivial lifecycle as the object evolves. Figure 1.2 shows a possible lifecycle of a cryptographic key, a common security object covered by a policy. The lifecycle restricts the possible state changes which are valid for a key. A state change is initiated when a condition is fulfilled, for example, when a key has expired and has to be destroyed. The security policy contains detailed instructions on the steps that have to be executed to change the state of the object.

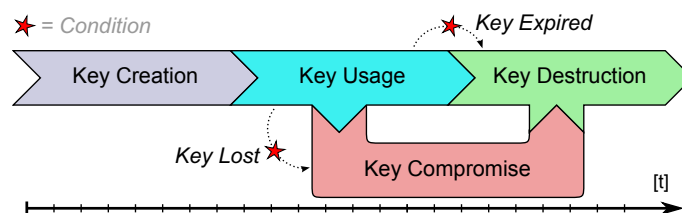


Figure 1.2: Example: Simplified Key Lifecycle According to the NIST [BBB⁺12]

Current policy documents make only little use of formal models to describe security objects and processes. Instead they rely on natural language to regulate how the system is to be used. Lifecycles and data of the security objects are often not explicitly documented in the security policy.

The usage of informal specification and omission of important parts leads to difficulties during the application of a security policy. Policy documents consisting of natural language and informal models can contain imprecise or ambiguous formulations. Although natural language is often used to define policies and specifications, it inherently contains ambiguities. A controlled form of natural language may improve the precision, but cannot completely eliminate all ambiguities [GZ05]. The problems are amplified by the fact that the policies are often written by a security expert and later used by an employee of the organization. Comprehensibility of the documents is however a crucial factor influencing the compliance with the policy [PSM07].

In practice, we could observe several incidents where users did not adhere to the security policy. The policy either suffered from ambiguities or users simply failed to follow the instructions. Current policy documents are suitable for an auditor but not for the actual user of the policy. Instructions that belong together are often scattered over multiple paragraphs which a user has to read and mentally combine. Execution of the processes is performed manually without any guidance that assists the user while he is performing his work. The absence of assistance during the application of the security policy increases the risk of human errors.

During the operational phase of the information system, the operator has to keep track of existing security objects. Without knowledge about the objects he cannot efficiently manage their lifecycle and guarantee the security of the system. Currently no generic solution exists that combines process execution and management of security objects. Instead, organizations use different spreadsheets, databases or other systems to track their security objects.

In summary, the current security policies based on natural language suffer from several problems that results in difficulties during the definition, validation and application of the policy:

- Informal documents are sometimes incomplete, ambiguous and difficult to understand. The policy is unintentionally violated because users misinterpret the documents.
- Current policies are only partly suited to guide users during the execution of the processes. Adhering to the written instructions is both cumbersome and prone to errors.
- Security objects are manually tracked and managed. As a result of errors, it is often unclear which security objects exist and in what state they are.

1.2 Thesis Objectives

The goal of the thesis is to address the problems identified in Section 1.1. The current security policies based on natural language are responsible for many security incidents which we could observe in practice. Our approach uses formal models known from Business Process Management and information modeling to specify and enact the operational processes of a security policy. These processes are a special form of business processes. Instead of creating a product or service, the processes of a policy regulate how security objects are created and managed. Using a business processes modeling language to specify operational processes of a security policies can therefore be considered as an appropriate choice.

Widely adopted business process modeling languages focus on activities and the sequence in which these activities are executed. These modeling languages are not suited to define the security objects and their lifecycles. Different approaches that combine lifecycle, information and business process models have been proposed [VDABEW01, Wah09, NKM⁺10, Mül09] in research literature but not yet applied in the domain of information security.

We propose a new framework, called the Integrated Modeling Framework for Operational Security Policies, to model the processes of security policies. The contributions of our work are:

An Integrated Modeling Language for Operational Security Policies

The framework uses formal process and information models to specify the operational parts of security policies. We propose the Security Object Modeling Language to model both, the processes and security objects that occur in the policy. The modeling language extends the UML and BPMN standards with new constructs that are required to handle security objects. We present a notation and the corresponding execution semantics to model the following aspects of a security policy in an integrated model:

- The processes which are required to create and manage security objects.
- The static information and dynamic lifecycle of security objects.

A Method to Design and Enact Operational Security Policies

We present an adapted form of the BPM lifecycle which includes security objects in the design and execution process. A business analysts can apply the methodology to specify process and security object models for the operational part of a security policy. We propose a Policy Support System which enacts the models. The Policy Support System assists users with the application of the policy. It guides the user during the execution of processes and helps him with the management of the security objects. The framework is completed by a prototype implementation of the Policy Support System based on an existing process engine. The system supports the execution of the models and allows monitoring and management of security object instances on the same level as process instances.

An internal evaluation of the methodology and execution engine validates the Integrated Modeling Framework for Operational Security Policies based on a real-life case study.

1.3 Thesis Structure

The remainder of this thesis is structured as follows: In Chapter 2, we present the foundations of security, BPM and modeling, on which the thesis builds upon. Existing approaches to combine process and lifecycle models are presented in detail and evaluated for their applicability to security policies. Chapter 3 presents the requirements of the Integrated Modeling Framework for Operational Security Policies. These requirements are implemented by the SOML language and methodology we propose in Chapter 4. In Chapter 5, we present a prototype of the Policy Support System. We have validated the methodology and Policy Support System based on a case study. The results of the validation are presented in Chapter 6. Chapter 7 concludes the thesis and provides ideas for further work to extend and improve our framework.

Background and Related Work

Chapter 2 introduces the foundations of information security (2.1), Business Process Management (BPM) (2.2.1) and information modeling (2.2.2) that serve as basis for the thesis. Widespread modeling languages lack the capabilities that are required to model and enact both, the processes and data of a security policy. Multi-view modeling (2.2.3) is an option to combine BPM and information modeling to formally specify processes and data as they occur in a security policy. Section 2.4 introduces existing work that combines business process and information modeling.

2.1 Security Foundations

Section 2.1 introduces the necessary background of cryptography and security. Only a brief overview of the relevant topics is presented, since the focus of this thesis is not on the technology used in information security.

2.1.1 Data Encryption and Digital Certificates

Information systems and their data often need to be protected from potential misuse. Information security therefore uses technology, defines processes and guides people to achieve different security objectives such as confidentiality, integrity and authentication [MVOV10]. Data encryption is one of the cryptographic tools that can be used to achieve the objectives of information security. Figure 2.1 shows the two main encryption algorithm categories. Symmetric algorithms use the same key to decrypt and encrypt the data. Exchange of this key between the sender and receiver of a message often poses a problem in practice. Asymmetric algorithms solves this problem with the introduction of a key-pair containing a Public-Key (PK) and Private (Secret)-Key (SK). Data that has been encrypted with the Public-Key can only be decrypted with the corresponding Secret-Key. The Public-Key can be disclosed and safely transmitted over an insecure channel.

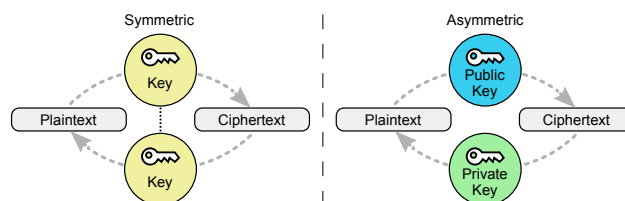


Figure 2.1: Symmetric and Asymmetric Cryptography

Asymmetric cryptography also serves as basis for the creation of (digital) certificates. A certificate is an electronic document that is used to bind the public-key to an identity. Certificates are typically issued by a trusted authority and are used to identify the owner of a public-key.

The fundamental goal of certificates, symmetric and asymmetric cryptography is to address the different objectives of information security. These objectives can be grouped into the four categories: confidentiality, data integrity, authentication and non-repudiation [MVOV10].

Figure 2.2 shows an example of a system that involves smart cards and a trusted terminal. The cards and terminal communicate over an insecure channel where cryptography is used to secure the information system:

1. When a smart card is inserted into the terminal it transmits its PK (and certificate) to the terminal.
2. The terminal uses the PK of the card to encrypt the data that it sends to the card.
3. The encrypted data is transferred over the insecure communication channel and decrypted with the SK of the card.

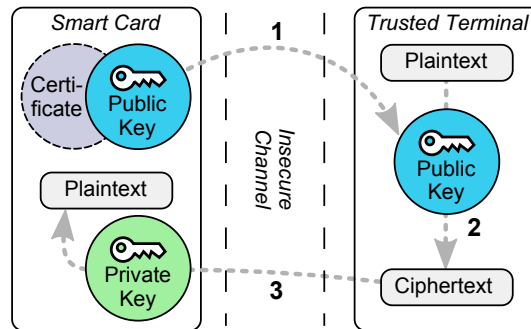


Figure 2.2: Example: Smart Card & Terminal

The example highlights how the application of cryptography is used achieve some of the fundamental objectives of information security:

Confidentiality	Data that is transferred between the smart card and the terminal is encrypted. The data protected from an adversary that is eavesdropping the communication between the card and terminal.
Data Integrity	All data that is sent between the terminal and smart card is encrypted. Therefore it is not possible to alter the data as it passes the communication channel. An attempt to modify the data can be detected during the decryption.
Authentication	The smart card could transmit a digital certificate instead of its public-key alone. The terminal can then use the certificate to check whether a trusted 3 rd party has identified the smart card and signed its Public-Key.
Non-Repudiation	Non-repudiation prevents an entity from denying previous commitments or actions. If the smart card is used for payment, the card owner could later deny that he has committed a purchase. The terminal could therefore also use cryptography to prove that a payment has been authorized by the card.

Public-Key Infrastructures (PKIs)

In the previous example a certificate has been used to verify the identity of the smart cards. The terminal controlled whether the public-key that has been provided by the card is bound to the identity of the card owner. The terminal however had to trust that the creator of the certificate has correctly identified the owner of the card. In a trivial setup, only a single certificate creator exists, whom every user of the system trusts. In practice such a configuration would have difficulties to scale since a single issuer has to create certificates for all participant of the system. Public-Key Infrastructures (PKIs) therefore use a system of Certification Authorities (CAs) to distribute the certificate creation and management over multiple entities [Shi00]. Figure 2.3 shows an application scenario, where a CA hierarchy is used to create certificates for the smart cards.

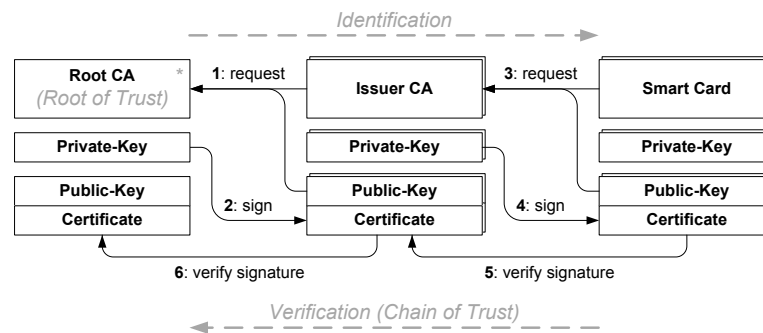


Figure 2.3: Example: Public-Key Infrastructure (PKI)

The PKI is used in two different phases. During the identification phase, the CAs verify the identity of smart cards and subsidiary CAs. Every participant of the system (Root CA, Issuer CAs and Smart cards) creates a key-pair. The public-key of this pair is sent to the superior CA in a certificate request. The CAs verify the certificate request and sign the public-key of the requester with their own private-key. The CAs thereby guarantee that the public-key of the request belongs to the identity of the requester. (Step 1-4 of Figure 2.3)

In the second phase, the certificates are used to verify the validity of a smart card. A terminal would now want to know whether the card has been issued by a trusted party. The terminal therefore analyzes the certificates of the card, issuer CA and root CA. The root CA thereby acts as root of trust, meaning that the terminal trusts the root CA. A card is considered as valid, if the certificates form a chain of trust that ends at the certificate of the root CA. (Step 5-6 of Figure 2.3)

2.1.2 Information Security Policy

An (information) security policy is a formal, brief, and high-level statement or plan that embraces an organization's general beliefs, goals, objectives, and acceptable procedures for a specified subject area [MD01]. The policy is often supported by a set of standard actions or rules that are required to comply with the policy. Optional guidelines may be used to provide best-practices for the implementation of the policy. Information security policies are applied by companies to regulate the usage of their IT infrastructure. Likewise larger information systems may use information security policies to regulate their application.

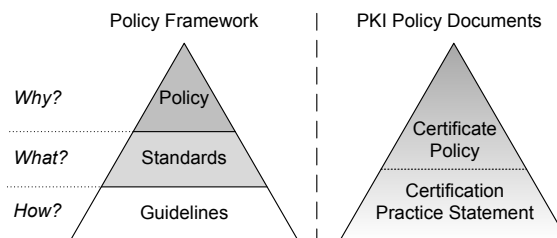


Figure 2.4: Policy Framework & CA Policy Documents [MD01]

Figure 2.4 shows the policy framework and its application to the certification authority in a public-key infrastructure. Certification authorities often employ a Certificate Policy (CP) and

a Certification Practice Statement (CPS) to regulate how certificates are issued [CFS⁺03]. The certificate policy contains rules and assumptions that are valid for the certificates issued by the CA. This document is often made public. A end-users that verifies a certificate can examine the policy and decide whether he wants to trust the CA. The Certification Practice Statement is derived from the Certificate Policy and describes the practices that a certification authority employs in issuing and managing its certificates. It defines the processes and standard operating procedures that specify how tasks have to be performed by the employees of the CA. The purpose of the CP is to define the desired results of the Certification Authority. The CPS then covers the concrete standards and practices which are applied to fulfill the requirements of the CP. The practice statement is therefore usually more detailed and procedurally oriented than the policy [Shi00].

2.2 Process and Information Modeling Foundations

Modeling languages and methodologies have been widely adopted and used in modern software engineering. The application of models concern all phases of the software engineering process. From requirements engineering [SSV98,NKF94] up to the actual implementation and execution [OMG03], models are used to improve or simplify the development process.

2.2.1 Business Process Management

Business Process Management is the discipline of defining and overseeing how work is performed within an organization [DLRMR13]. The goal of BPM is to achieve consistent outcomes of processes, align them to customer needs and continuously take advantage of opportunities to improve existent processes. Different definitions of what a business processes is, have been proposed. Generally a process consists of activities (or tasks) which are executed in a given sequence to produce an output. According to Davenport a business process is:

... a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. ... [Dav93]

A core concept of BPM is the continuous improvement of existing processes. Therefore the lifecycle not only covers the initial analysis and design phase of a process, but also includes monitoring and process optimization.

Workflow & Process Modeling

Modeling languages can be used to model and represent the processes and standard operation procedures used by an organization. Some of the most popular and widely used languages for process modeling are: Event-Driven Process Chains (EPCs) [KN92], Unified Modeling Language (UML) Activity Diagrams [OMG11e] and Business Process Model and Notation (BPMN) [OMG11a]. These models provide constructs to capture the activities, control-flow, data and other aspects that are required to define a business process. An overview of the capabilities of the different languages has been presented in the workflow-patterns¹. Modern languages, such as BPMN [OMG11a], Yet Another Workflow Language (YAWL) [vdAtH05,Fou12] or WS-Business Process Execution Language (WS-BPEL) [OAS07] take a further step and also define detailed execution semantics of

¹<http://www.workflowpatterns.com>

the models. Workflow and process engines [Act, IBM] support the execution of the business process models. These systems can automate tasks and assist users during the execution of a process.

BPMN is an Object Management Group (OMG) standard for business process modeling using models and graphical representations. BPMN claims to be suited for stakeholders originating both, from the technical and business side. The language includes a notation that is intuitive to business users but is also able to model complex process semantics. Since the notation is understandable by all stakeholders it allows communication between the business analysts and the developers who are responsible to implement the processes. The BPMN standard defines a model interchange format [OMG11c] that is used to store and exchange models between different modeling tools and process engines. Many process engines either support native execution of BPMN models [Act] or require a transformation to other formats such as WS-BPEL [OAS07].

Most modeling languages, including BPMN are activity-centric and provide only limited support for modeling the data that in conjunction with a process. The need to represent data and its use has been identified and included in different modeling languages. A set of data-relevant workflow patterns have been identified [RTHEvdA05]. Existing process engines however only provide limited support for the advanced patterns, which define interactions between processes and data. Many engines either do not support the patterns or require additional code or workarounds to accomplish the desired results [RTHEvdA05].

BPM Lifecycle

Business Process Management involves additional topics beyond the scope of process modeling. First of all, processes that are executed in an organization need to be identified and discovered. Once a process has been modeled and is being executed an organization might want to monitor and improve it. Business process lifecycle management covers the lifecycle phases of a process. Different BPM lifecycles have been proposed in the literature [DLRMR13, ZM04]. The lifecycles are often inspired from the concept of continuous improvement used in process and software engineering [TD92]. Figure 2.5 shows one of the proposed BPM lifecycles.

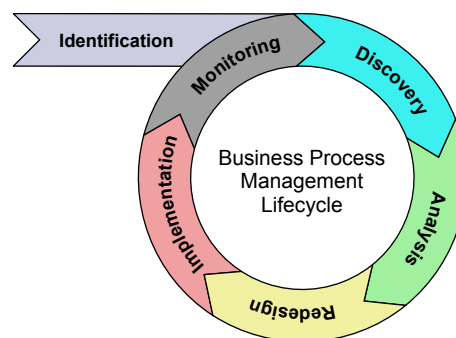


Figure 2.5: BPM Lifecycle [DLRMR13]

Identification	Process identification tries to identify all relevant processes of an organization. The result of the identification phase is a process architecture that provides an overview of all relevant processes and their relations.
Discovery	The current state of the identified processes is modeled in the discovery phase. Based on the current practices and operating procedures a business analyst creates the “ <i>as-is</i> ” process models.
Analysis	The discovery phase yields a model of the current state that can be analyzed to

	discover potential problems or bottlenecks. Process analysis can, for example be performed with the help of process simulation.
Redesign	Process redesign covers the optimization of the existing processes based on the insight gained in the analysis phase. The result of the redesign phase is a “ <i>to-be</i> ” process model that specifies how the processes have to be executed in future.
Implementation	During the implementation phase, the process is enacted by implementing organizational changes or using process automation. Organizational changes imply reorganizing the way employees work while process automation includes the use of IT infrastructure to support or automate processes. To support the execution of a process with IT, the previously created process models have to be transformed into executable form. Generic workflow engines or Business Process Management Systems (BPMSs) simplify the implementation of processes by supporting the execution of process models. Compared to simple workflow engines which only support execution of process models, the BPMSs suites often provide support for the other phases of the BPM lifecycle.
Monitoring	Processes that have been implemented and are being executed are actively managed and monitored. Key Performance Indicators (KPIs), such as process execution time or error rate, are collected in real-time and stored. The collected data is used for operational management and to gain further knowledge of the processes. The BPM lifecycle is repeated based on this additional knowledge and results in continuous improvement of the processes.

2.2.2 Information Modeling

Information or data modeling is concerned with the formal representations of entities (or objects). The emergence of relational databases triggered the development of Entity Relationship Models (ERMs) [Che76] to specify the increasingly complex data structures that reside in the databases. Object-oriented software development picked up the ideas of information modeling and included them into the software development process [RBP⁺91]. In the recent years UML [OMG11e] became the de-facto standard to model software systems. It provides a variety of models to capture the static and dynamic behavior of data. Information modeling focuses on the static aspect of the data. A generic definition of the content of an information model has been given by Kilov and Ross [KR94]: An entity (or object) consists of operations that can be performed on the entity, properties and associations to other entities. Domains (or data-types) specify the type of properties and restrict the possible values. The five elementary association types: Dependency, Reference, Composition, Sub-typing and Symmetric Relationship define relations between entities.

Lifecycle Modeling

In addition to the static definition of operations, properties and relations, entities may also have a dynamic behavior. This dynamic behavior defines how the object evolves over its lifetime. As the object evolves it steps through different states which are connected by transitions. These transitions restrict the possible state changes that can be performed on an object. A cryptographic key, for example is created, activated, used and destroyed. Using the key without activating is not possible. A direct transition from created to used is therefore not allowed.

Object lifecycle models are mostly represented using languages based on either Finite State machines (FSMs) (or automaton) [Bla,Moo56], statecharts [Har87] or petri nets [KS91,SS02]. Finite state machines represent an object lifecycle with a finite set of states that model the lifecycle phases of the object. These states are connected by transitions which are taken when a trigger is received. A finite state machine is defined by the quintuple: $[(Q, \Sigma, \delta, q_0, F)]$ where:

- Q Is a finite set of states in which the object resides over its lifetime.
 Σ Is a finite set of triggers that can fire a transition.
 δ Is a function ($\delta : Q \times \Sigma \rightarrow Q$) that defines how the occurrence of a trigger affects state.
 q_0 Is the initial state. ($q_0 \in Q$)
 F Is a set of final states, which mark the end of the objects lifetime. ($F \subseteq Q$)

A lifecycle modeled based on a FSM is purely sequential and does not contain any parallel constructs. Statecharts extend the semantics with hierarchical states and concurrency. UML state machine models [OMG11e] are based on statecharts and introduce additional features.

2.2.3 Multi-View Modeling

Using a single model to represent a complete system is often not possible or has several disadvantages. A single model showing every aspect of a system might fail scale as the complexity of the systems grows. In addition, the stakeholders of a system might want to use different models, notations or methodologies to capture their domain knowledge. Multi-view modeling, multi-perspective modeling and viewpoints have therefore been developed and applied to software engineering, requirement elicitation and process modeling [Kru95, SSV98, NKF94]. Instead of using a single model, these approaches apply specific models, notations and methodologies to define the different aspects of a system. Each view and its models can thus be tailored to the domain and its stakeholders. Figure 2.6 shows the application of multi-view modeling to the domain of security policies.

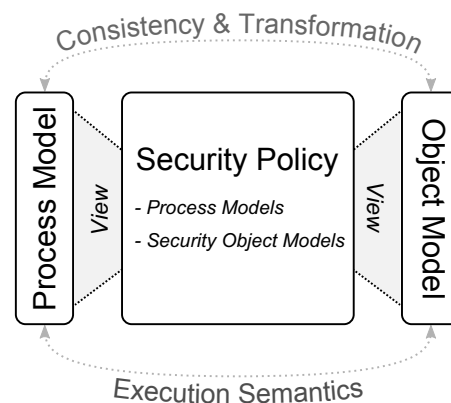


Figure 2.6: Multi-View Modeling. [Wah09]

A security policy defines processes that create, modify or delete different security objects. Using two different models for processes and security objects, allows the usage of most suited modeling languages to specify the different aspects of the policy. Multi-view modeling however introduces a set of new problems, both during design- and run-time of the models. Multiple models covering different aspects of the same system often overlap or complement each other. It is therefore crucial that the models are consistent among one and another. Different methods to detect (and resolve) inconsistencies between models have been proposed [Wah09, VDS05, K us04]. Model transformation [KWB03] can be used to reduce the risk of inconsistency or to derive parts of a model from each other [Wah09]. Other approaches use an integrated model presenting multiple aspect of a system and avoid consistency issues between models [GBJ02]. Using independent modeling languages that were not designed to be executed together, also yields the question about their execution semantics when they are used in conjunction.

2.3 Modeling Security Policies

Models and formal notations have been applied to the specification and development of secure systems for quite some time [BDL03, LBD02]. Many of these approaches are used to specify how the information system itself fulfills certain requirements of a security policy. They only cover the technological aspect of information security and neglect processes and people that are required to operate the systems. Other approaches that are concerned with the processes and people only cover a narrow part of a security policy, such as rule based access control or authorization [DDLS01, SL02]. In the domain of PKIs, identity documents and smart cards, much of the security however relies on the correct execution of procedures that are required to set up and operate the information system. Modeling languages to define the processes that covered by a security policy are currently missing. These key management processes are crucial for the secure operation of the system:

Definition (Key Management Process)

A key management process is a sequence of activities that are required to execute security relevant work in compliance with the policy documents. Typical characteristics of these processes are:

- Processes often involve human-intensive activities. In some cases automation would be possible but the policy requires that the activity is supervised by multiple users in person.
- Some of the processes are only executed during an initialization phase or must be repeated infrequently. Nevertheless, it is crucial that these processes are executed as defined by the policy. Subsequent operation relies on the assumption that the initial processes have been executed correctly. An example for such an initialization process is the generation and storage of the key-pairs used by a certification authority.

The purpose of the key management processes is to create and manage the security objects that are required for the secure operation of the information system. Most key management processes result in the creation or manipulation of security objects:

Definition (Security Object & Security Object Instance)

A security object is an abstract object that is relevant in the context of security. It contains information in the form of attributes and relations with other security objects. The object has a dynamic behavior that defines how it evolves over its lifetime. Common security object are:

- | | |
|--------------------------------|--|
| Cryptographic Material: | Symmetric/Asymmetric Keys, Certificates, ... |
| Physical Credentials: | Key Cards, Keys, Security Tokens, ... |
| Secured Environments: | Hardware Security Module (HSM), Safes, ... |

A security object instance is a concrete instance of a security object.

Existing security object instances must be tracked and managed once a policy is applied in an operational environment. The tasks that are required to handle the security object instances is summarized in security object lifecycle management:

Definition (Security Object Lifecycle Management)

Security object lifecycle management covers the management of security object instances during the operational phase of the information system. Lifecycle management includes the following tasks that are required to keep control over the existing security object instances:

- Controlled creation, manipulation and destruction of security objects.
- Maintaining current data and state of security objects.
- Notifications or actions based on state and data changes of security objects.

2.4 Integrating Object Lifecycle and Workflow

Current business process modeling approaches are mainly concerned with designing the activities that are required to achieve a business goal. Process models focus on the control flow and empathize the sequencing of tasks. Modeling data, handled by the processes, is often omitted or only treated in the scope of independent activities. Artifact-centric business process modeling [NC03, BGH⁺07] acknowledges that modeling data and its behavior is sometimes as important, as the control flow.

Multiple authors [VDABEW01, Wah09, NKM⁺10, Mül09, EQST13] have since then investigated how behavior and information specification of data can be included in the business process modeling methodology. The approaches generally model the lifecycle of non-trivial data objects. Lifecycle and information specification of the data is then integrated in the process definition and execution.

In this section we present existing approaches to define an artifact-centric business process or similar modeling frameworks. The common goal of the approaches is, to combine data and process modeling. Figure 2.7 shows an overview of the existing approaches and how they influenced each other. All approaches use concepts from the early work in information [Che76], lifecycle [Bla, Moo56] and process modeling [KN92]. Within the field, the work on artifact-centric business processes laid the foundation for further research [Wah09, NKM⁺10, EQST13]. Besides these approaches different researchers [VDABEW01, Mül09] investigated how data and process modeling can be combined.

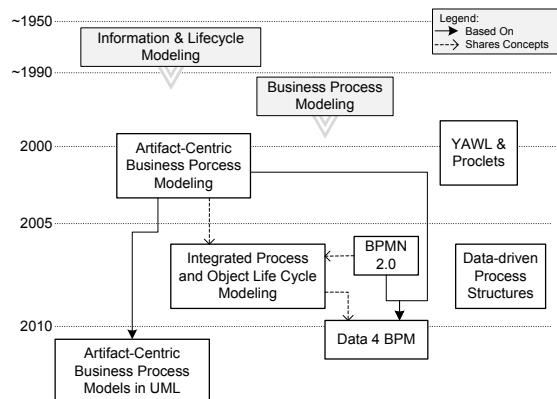


Figure 2.7: Integrating Object Lifecycle and Workflow: Timeline

NOTE: Different terms are used in the literature to describe a data element that has a non-trivial dynamic behavior. For simplicity the term “object” is used for the remainder of this chapter. The original terms include: (Business-) Artifact, Record, Entity, Object, Data and Case.

2.4.1 Foundations of Artifact-Centric Process Modeling

Artifact-centric business process modeling emerged from the need to better represent data in business processes models. The initial work was heavily influence by business-transformation or business-process-integration engagement conducted by IBM [NC03, BGH⁺07]. Two main reasons for the shift from an activity-centric to a data-centric design process where:

- Artifact-centric process modeling methodologies can start with modeling the work items which a business deals with. Modeling key objects, that are relevant for revenue instead of the processes has been identified as more amenable to business people. Other objects and

processes are then discovered based on the key objects. Process and object specification is iteratively refined until an enterprise-wide view of the business has been built.

- Including the lifecycle of objects in the process model, allows flexible customization of the overall processes. Coordination between processes can now be changed by adjusting the lifecycle of the object without touching the underlying processes.

Initial object models only defined the information model of an object and no dynamic behavior [NC03]. Processes therefore could manipulate the state of an object without any restriction. Later object definitions also included the lifecycle of an object and prevented the process from requesting an invalid state change [BGH⁺07].

Combining lifecycle and process models also introduces drawbacks. Synchronization between object lifecycle and process can result in various problems such as “dead-ends” or “redundant attributes”. First methods to check consistency between the models were therefore developed [BGH⁺07]. The initial approaches focused primarily on the design phase of business process management. Enactment of the process models was already in discussion but not yet implemented.

2.4.2 Other Approaches to Model Data in Processes

YAWL and Proclets

Independent of the artifact-centric business process movement, the need for data modeling has been recognized by other researchers. The Yet Another Workflow Language (YAWL) [Fou12, vdAtH05] supports modeling and enactment of data objects. An object does however not specify any dynamic behavior and therefore also contains no lifecycle model. YAWL instead introduces the inter-workflow service Proclets [VDABEW01, Fou12] to compose processes. Proclets are described as “lightweight interacting processes” [Fou12]. They are used to divide entangled processes into simpler fragments and synchronize their execution. Proclets are not directly related to data objects. Nevertheless they are relevant, as they introduce entities which define an interaction graph that is similar to a lifecycle. Proclets can communicate with each other using interaction points. The process specific communication protocol, that is used between interaction points is defined in entities. An entity is an object that exists in conjunction with a Proclet. An entity typically represents a (data) object can therefore be compared to the objects used by the other approaches.

Implementation: YAWL defines static data objects which are created and manipulated by the process. A data object is either stored internally or located on an external system. An activity that uses data, defines input and output parameters which read or manipulate the object. The object is passed between activities using a variable in a shared space. Explicit data-passing between activities is not supported.

Proclets are lightweight processes that communicate with each other to complete a larger business goal. Figure 2.9 shows an order process where the order and construction of the product is handled by two independent Proclets that communicate with each other. Proclets define interaction points on which messages are exchanged. An interaction point can also be used to define dependencies between two tasks. Figure 2.9 shows an example, where the creation of the *Procure* Proclet needs to be followed by the *Ship* task.

Entities are used to specify the interaction protocol between interaction points. The entity instance exists in conjunction with the Proclet instances and represents the case that combines the Proclet instances. An entity defines an interaction graph that consists of interaction nodes and arcs. Each node maps to an interaction point in a Proclet. An arc references a communication between two interaction points. Figure 2.8 shows the constructs for Proclets and the interaction graph. The interaction arc contains state information that describes the current state of the referenced

communication. This state is updated whenever a communication occurs. The interaction graph can thus be used to check whether the Procleets communicated with each other as intended.

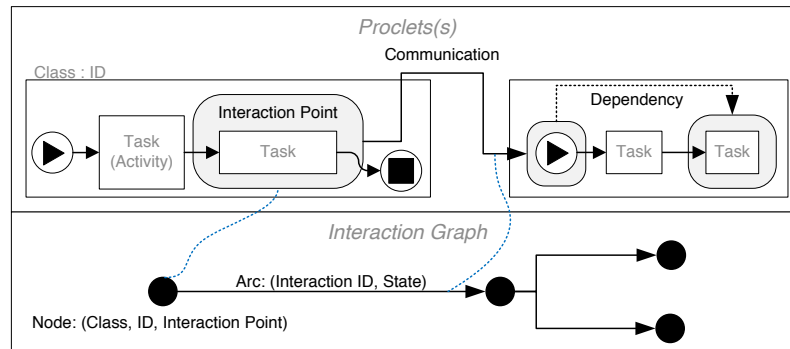


Figure 2.8: Procleets and Interaction Graph

Figure 2.9 shows an example of a process execution which is supervised by an interaction graph. In the first phase, the state of the *Procure* communication is *unproduced* since a message has not yet been sent. A message is produced, sent to the Procleet and consumed as the process continues and reaches the *decide* task. Therefore the state of the interaction arc is updated to *consumed*. The interaction graph can thus be used to ensure that the communication between the *Order* and *Procure* Procleet has been completed in the correct sequence.

Representation: YAWL relies on XML-based standards to handle data. Data is stored in XML documents, queried with XPath [W3C12a] and manipulated using XQuery [W3C10]. The information model of a YAWL data object is defined using a predefined or user provided XML schema [W3C12b]. Entities contain an interaction graph which is represented as directed graph. Node and edge names are used to reference to the corresponding Procleet class and instance.

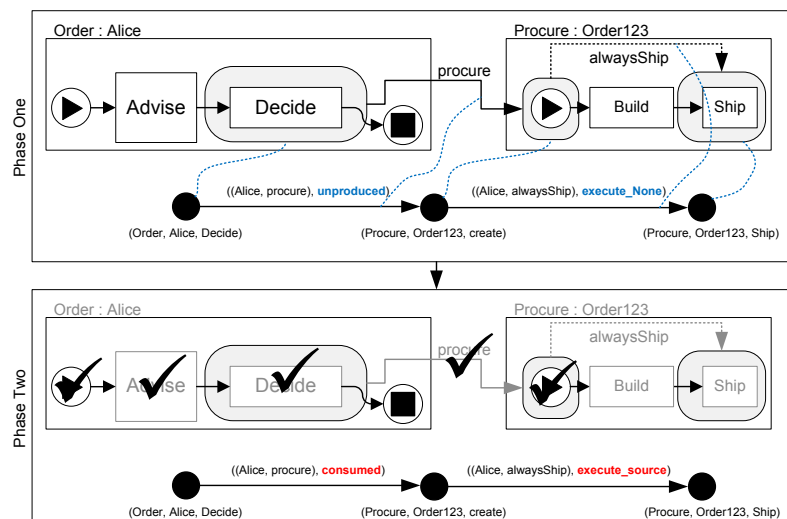


Figure 2.9: Procleets and Interaction Graph: Example

BPMN Standard

The BPMN 2.0 standard [OMG11a] is able to represent data that is created, manipulated and used during the execution of a process. It only defines data modeling on an abstract level. Concrete models and languages for describing or querying data are not part of the standard. Execution of the models is therefore not possible by default. 3rd Party vendors are expected to provide implementations for data modeling. The standard recommends XML Schema [W3C12b] as information model and XPath [W3C12a] to query the data. The recommended data models do however not mention dynamic behavior of the objects.

Implementation: The standard primarily uses *Data Objects* to model physical or information objects. *Data Objects* are used as input or output of other process elements. They include states, which can be accessed using the `<objectName[State]>` notation. *Data Associations* are used to model the data-flow within processes and to pass object references between the activities. The lifetime of an object is bound to the process instance in which it is contained. Object instances are created as the process starts and disposed when it terminates. *Data Stores* can be used to model persistence of *Data Objects*. They allow the designer to model persistence of objects beyond the scope of a process. Specification on how such a storage container is accessed or queried is however not included in the standard.

Representation: BPMN does not define a concrete models for objects and therefore also no notation. BPMN processes use the notation shown in Figure 2.10 to model interaction with the data.

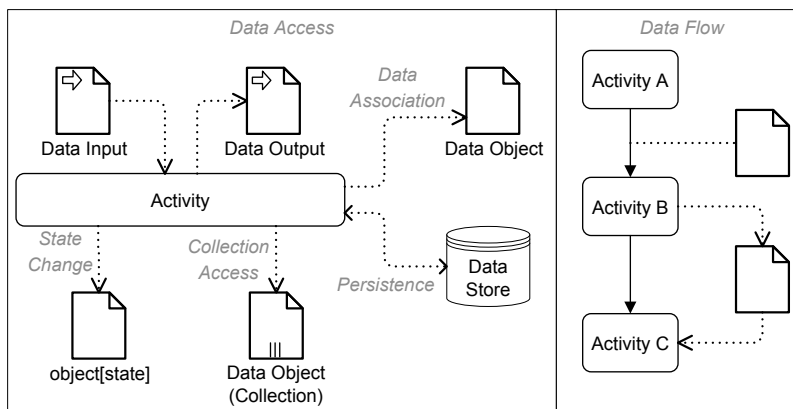


Figure 2.10: BPMN Data Modeling Elements

Data-driven Process Structures

Configuration Based Release Processes (Corepro) [Mül09] use data modeling to facilitate coordination and configuration of different processes. In application domains, such as car development, a variety of processes have to be executed to develop, test and release the final product. Coordination between the dependent processes becomes complex, as a product's process structure may involve thousands of processes. In Corepro the dependencies between processes are not specified in the process models. Instead they are captured on the level of product-components. Dependencies between product components and their lifecycles are defined in a data model. Process structures are then automatically derived from the data models.

Implementation: The information model only contains the relations between sub-components but no information about their data. A directed graph with the relations between the objects is used to model the data structure. Finite state machines [Bla] are used to specify the lifecycle of each object. The Corepro object lifecycle thereby uses a slightly adapted version of a FSM:

- Each transition between two states can be assigned to a process. A process instance is created whenever the transition is taken. A transition completes when the process instance terminates.
- A single state can have multiple outgoing transitions that use the same process. The return value of the process instance is used to determine which transition is taken.
- Objects that share a relation in the information model may also specify dependencies between their states. The object lifecycle structure is used to model dependencies between the states of different objects.

Process structures are derived based on the information, lifecycle and lifecycle structure model. A concrete instance of the data structure is pre-configured before the actual generation begins. Processes instances are only used to perform a single transition and do not change the object. They receive the object and state as input and produce a result that may be consumed by the lifecycle. Figure 2.11 shows an example of the different Corepro models for the development process of a car. Product/Data structure and object lifecycle models define the information and behavior of the car's parts. The process configuration rules are then used to map transitions of the lifecycle to a process.

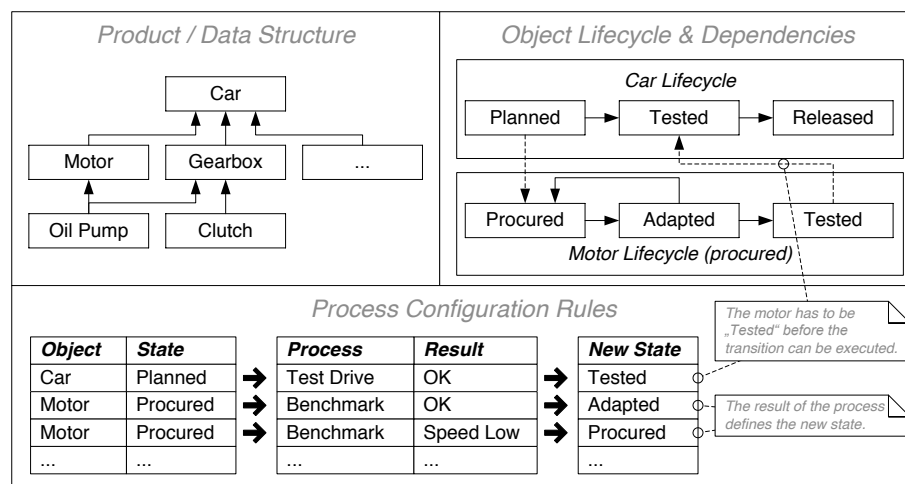


Figure 2.11: Corepro Models and Configuration

Representation: Information, lifecycle and lifecycle structure models are represented as directed graphs. A configuration defines the relations between lifecycle transitions and processes.

2.4.3 Artifact-Centric Business Process Modeling

Integrated Process and Object Life Cycle Modeling

The framework for Integrated Process and Object Life Cycle Modeling [Wah09] introduces a generic model for work/control-flow, data-flow and lifecycle models. Object lifecycles and processes are modeled and combined based on these models. The framework is used to ensure consistency between the models. Therefore, it defines methods for inconsistency detection and resolution.

Model transformations are used to support the design process of the lifecycle and process model. Execution of the models is currently not supported.

Implementation: The framework builds up on the foundations of the UML activity and state machine model. It enhances their expressiveness and clarifies the execution semantics when they are not fully specified. The framework uses the same concepts for data modeling as UML and BPMN: Objects are created by the process and routed through the process using the data-flow constructs. An object instance is thereby exclusively held by a single process instance. The framework formalizes the following modeling constructs, which allow a process to interact with an object:

Repository Data Flow

Objects can be stored in a repository that is accessible to the whole process. The process uses this repository to store and load object references. Objects transferred using the repository are always passed by reference. (Figure 2.12 [a & c])

Route Data Flow

Alternatively a process may use routed data flow to transfer an object. The routed data flow is defined on a transition edge between two process activities. Objects transferred using the route flow are always passed by value (i.e. copied). (Figure 2.12 [b & d])

State Condition (Accepted States)

An activity that is working with an object can define valid states which the object might have as it enters the activity. An activity “reads” an object, if it only contains accepted states. It “updates” the object if it also changes its state. (Figure 2.12 [c & d])

State Change (Produced States)

An activity can define the state of an object as the activity completes. Objects are “created” by activities that define a state change without having read the object upon entering the activity. Additional constraints on the possible state changes can be defined based on the state of the object upon entering the activity. (Figure 2.12 [c & d])

Lifecycle models are defined as finite state machines and represented in UML state machines. The model only captures the dynamic behavior of the object and nothing about its data.

Representation: The models for processes and lifecycles are represented using UML activity and state machine diagrams. Figure 2.12 shows the lifecycle model and the constructs used to access an object from within a process.

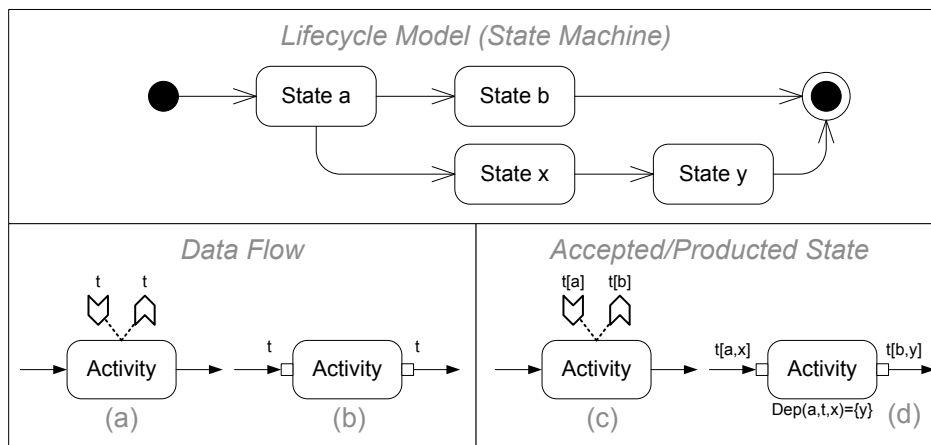


Figure 2.12: Integrated Process and Object Life Cycle Modeling

Data 4 BPM (BEDL)

IBM Data 4 BPM [NKM⁺10,NPK⁺11] defines the Business Entity Definition Language (BEDL) to model information and behavior of objects. BEDL models are loaded and executed by a separate software component which acts as repository and management service for the object instances. The process engine interact with this component using a predefined interface. Processes are either defined using WS-BPEL or BPMN [NPK⁺11,NKW12].

Implementation: Objects are modeled in XML using the BEDL dialect. A BEDL model of an object consists of four main components:

- Information** The information model specifies the data of an object as attribute/value pairs. The Data-type of an attribute is defined in an XML schema.
- Lifecycle** Lifecycle models are specified as finite state machines. The model does not contain activities that occur in a state or as a part of a state transition. Detailed behavior of the object is specified in process models which go along with a BEDL specification. The BEDL model only shows the behavior of an object from a macro-perspective.
- Notifications** A process engine that interacts with the BEDL-component can subscribe to notifications. The model supports definition of notification events on state and data changes. The BEDL-component monitors changes to the object and notifies the subscribers.
- Access Policy** BEDL models contain access policies for create, read, update, delete and execute state change (CRUDE) operations. User can be restricted from performing CRUDE operations based on the object's state or an XPath [W3C12a] expression. Access policies are also used to define transition guards regardless of the users role.

BEDL models are loaded and executed in a separate software component. This component acts as repository for the object instances and allows the processes to CRUDE object and subscribe to notifications. A reference to an instance must be obtained before an object can be used by a process. The repository therefore provides a query interface to search for objects. The repository allows multiple processes to use the same object instance simultaneously. Processes must therefore request a lock on the object to avoid synchronization issues. Business processes defined in WS-BPEL [OAS07] use the BPEL4Data extension to interact with the objects located on the repository. The extension captures the intent of a process to manipulate the object's data or state.

Representation: BEDL and BPEL4Data are XML dialects with graphical representation. Figure 2.13 shows how BPEL4Data constructs are visualized in a process model. The first process of the example creates an instance of the *Object A* which is later retrieved (by querying the repository) and changed by the second process.

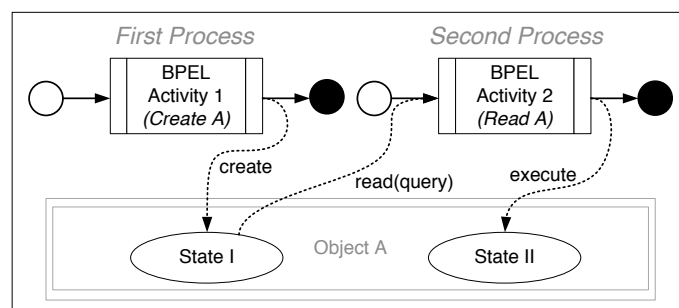


Figure 2.13: WS-BPEL extended with BPEL4Data

BPM 4 Data also introduces a modeling notation for BPMN models based on the data modeling capabilities of the BPMN standard. The BPMN notation uses the same semantics of produced and accepted states as the Framework for Integrated Process and Object Life Cycle Modeling [Wah09]. The BPMN elements *Data Object* and *Data Association* are used to model access to objects and reference passing between activities. An additional BPMN pool contains the lifecycle of the object. Figure 2.14 shows the same example as in Figure 2.13 using BPMN. The model is however only used to model state related actions and is currently not executable.

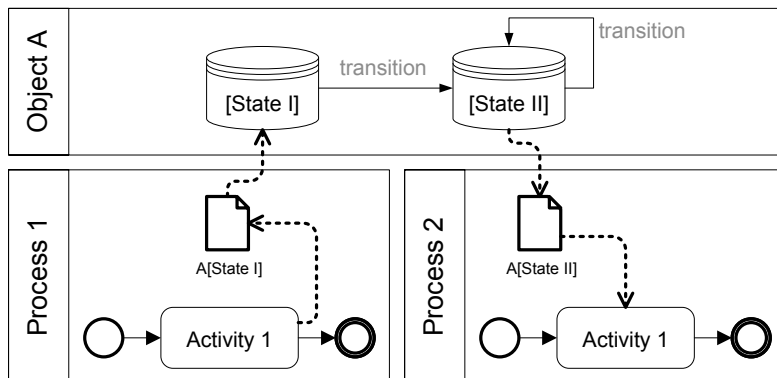


Figure 2.14: BPMN extended with BPM4Data

Artifact-Centric Business Process Models in UML

The UML standard [OMG11e] defines models for describing both, the static and dynamic behavior of a system and its data. UML class and state machine diagrams are widely used to model data in the context of software engineering. UML activity diagrams can be used to model the high-level behavior of a system. The authors of the Artifact-Centric Business Process Models in UML [EQST13] apply the UML models to define data and processes. Business processes are modeled using UML activity diagrams, which interact with objects that are specified in class and state machine diagrams. The Object Constraint Language (OCL) [OMG12] is used to add constraints and behavior to the models. UML activity diagrams are however only partly suited to model a business processes. An evaluation [RvdATHW06] revealed several limitations:

- Activity diagrams only provide limited support for the organizational aspect of a process.
- Interaction with the environment in which the process is executed can not be captured.

Implementation: UML class models are used as information model. An object can consist of attributes and relations to other objects. The object is annotated with OCL constraints to restrict its content. The same object can change its characteristics as it resides in different states. Inheritance is used to model the characteristics of an object as it resides in a certain state.

UML state machines are used to model the dynamic behavior of an object. The state machine consist of states which are connected by transitions. A transition is triggered when the corresponding event is received. The model supports two kinds of events:

External events An external event always references a process that belongs to the transition. Their execution is explicitly requested by the customer of the business process.

Internal events Internal events correspond to conditions stated over the content of the object and cause the execution of services without requiring any customer intervention.

All operations that are performed on an object are defined as services. A service is a unit of work that is relevant in the context of a process. It changes the state and data of the object. Services are implemented using the object constraint language and called by the lifecycle and process models.

UML activity models are used to define the detailed behavior of external events. Each external event in the state machine has an associated process in the form of an activity model. The process defines in which sequence the services of the objects are to be executed. As a process is started it receives a reference to the corresponding object instance. The process may also call services of related objects. Swimlanes are used to indicate on which object type a service has to be called.

Representation: The authors use the standard UML notation and models to represent objects and processes. Figure 2.15 shows the information, lifecycle and process model of an *Object A*. The lifecycle model contains an external event that references *Process I*. Services are written in OCL and have no graphical representation.

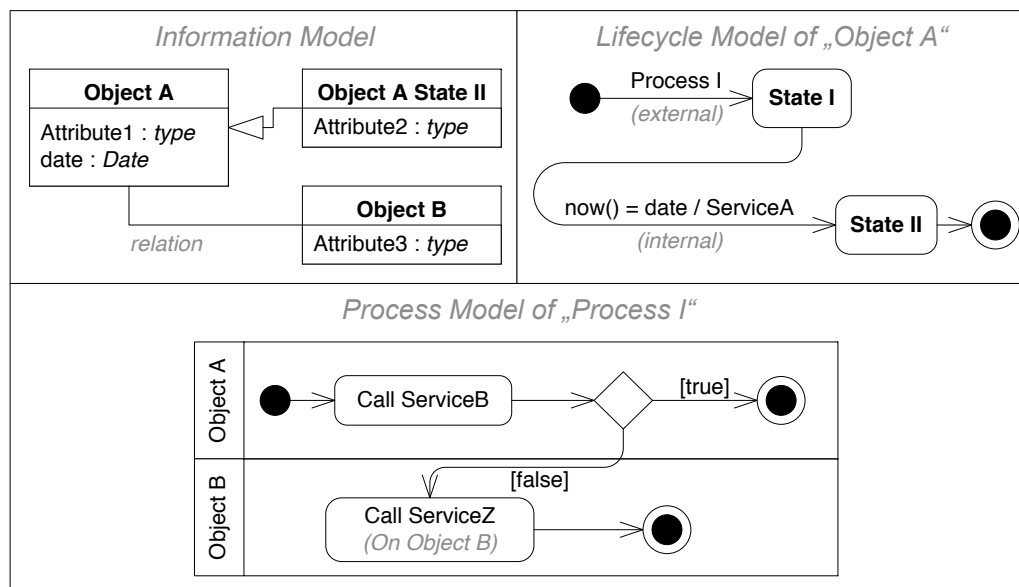


Figure 2.15: Information, Lifecycle and Process Models

2.4.4 Benefits and Limitations of the Approaches

Section 2.4 summarized various approaches, that combine business process and information modeling. The approaches originate from the field of business process modeling and use information modeling to achieve different goals. Information models are used to synchronize and control process executions or to model the data that is used by the process. Some of the approaches do not support execution of the models or do not define concrete execution semantics. Besides their primary goals and application domains, the main differences between the approaches are:

Relation Between Process and Lifecycle

Process and lifecycle instances are both active elements that can influence each other. To specify their execution semantics, one must first define how they are related to each other. The process and lifecycle model can either be equal elements or one is subordinated to the other. Most approaches either use the process or lifecycle instance as the controlling authority. In

this setup, the controlling authority is actively executed and thereby controls the subordinated element. At the expense of additional complexity of the models, it is also possible to allow both elements to take an active role.

Modeling Notation and Representation

Different model types and representations are used to model objects, processes and their interaction. Depending on whether the models are executable in a software system, they include a more or less detailed specification.

Information Model

An information model that defines attributes and relations of the objects is independent of the objects behavior. Some of the approaches therefore focus solely on the dynamic behavior of the object and do not consider data. In these cases, processes only interact with the state of the objects and do not model access to their data.

Table 2.1 summaries the core concepts of the different approaches and identifies the relation between the process and object models.

	<i>Concept Summary</i>	<i>Relation Between Data and Process</i>
<i>YAWL</i>	Procllets are lightweight, interacting processes which define the interaction protocol between processes. Entities are the case/objects that hold the different Procllet instances together. Data modeling itself does not support dynamic behavior.	Independent process interaction and information models: Data has no dynamic behavior and is only used by the process. Procllet-entities define and control the interaction between Procllets.
<i>BPMN</i>	The standard defines concepts for data modeling on an abstract level. 3 rd party vendors are expected to provide an implementation.	Multi-view modeling: The information model shows parts of the process from a different perspective. The process is the controlling authority and object instances are bound to the scope of a single process.
<i>Lifecycle and process modeling</i>	The framework is used to validate consistency between lifecycle and process models. It provides inconsistency detection and defines model transformation to convert between lifecycle and process model.	Multi-view modeling: The information model shows parts of the process from a different perspective. The process is the controlling authority and object instances are bound to the scope of a single process.
<i>Corepro</i>	Data models only acts as instrument to derive process structures. Dependencies between processes is removed and represented in the data model.	Dependency modeling: Object lifecycles, relations and dependencies define the execution order of processes. A process instance is only responsible for a single transition in the lifecycle. The process itself has no influence on the object.

Continued on next page

Table 2.1 – Continued from previous page

	<i>Concept Summary</i>	<i>Relation Between Data and Process</i>
<i>Data 4 BPM</i>	BEDL supports modeling information and behavior of objects. These models are loaded and executed by the BEDL engine. WS-BPEL or BPMN is used to define processes which interact with the BEDL engine and the data.	Macro–micro modeling: Lifecycle model presents the behavior of the system from a high–level perspective. Processes contain the detailed behavior and work with one or more objects. An object instance can be used by multiple processes simultaneously.
<i>Artifact–centric models in UML</i>	The models are based on the UML and OCL standards. Information modeling uses class and state machine diagrams. Processes are defined in UML activity models.	Macro–micro modeling: Lifecycle model presents the behavior of the system from a high–level perspective. Processes contain the detailed behavior and work with an object and its related objects.

Table 2.1: Summary of the Underlying Concepts

Depending on the background of the work, different languages and notations are used to model the processes and objects. The modeling frameworks however largely agree with each other on the modeling languages and notations used to represent the object’s data and behavior:

Lifecycle Model	The approaches used lifecycle models based on the finite state machines, statecharts and petri nets as introduced in Section 2.2.2. Most of them only support lifecycles based on purely sequential finite state machines.
Information Model	When an information model is used to model the data of the object it is either based on XML schema or UML class models.

Table 2.3 summarizes the features that are supported by the different approaches. Only the features that are relevant to model the data and processes of security policies are presented. Table 2.2 shows the notation that is used to present the features.

<i>Related Work Identifier</i>		<i>Feature Support</i>	
YAWL	YAWL & Proclets	–	Parent construct is not supported
BPMN	BPMN Standard	(✓)	Partly Supported
LC&P	Lifecycle and Process Modeling	✓	Supported
Corepro	Corepro – Data–driven Process Structures	?	Undefined
D4B	Data 4 BPM, BEDL	(n/a)	Used in a different context
UML	Artifact–Centric Business Process Models in UML	n/a	Not Supported

Table 2.2: Legend for Table 2.3

	<i>YAWL</i>	<i>BPMN</i>	<i>LC³P</i>	<i>Corepro</i>	<i>D4B</i>	<i>UML</i>
<i>Information Model</i>	✓	n/a	n/a	(n/a)	✓	✓
<i>Technology</i>	XSD	–	–	–	XSD	UML
<i>Attributes</i>	✓	–	–	–	✓	✓
<i>Associations</i>	✓	–	–	✓	(✓)	✓
<i>Integrity Constraints</i>	n/a	–	–	–	✓	✓
<i>Sub-Typing</i>	n/a	–	–	–	n/a	✓
<i>Lifecycle Model</i>	(n/a)	n/a	✓	✓	✓	✓
<i>Technology</i>	Entities	–	UML	FSM	FSM	UML
<i>Branches</i>	–	–	n/a	✓	✓	✓
<i>Dependencies Between Lifecycles</i>	–	–	n/a	✓	n/a	n/a
<i>Integrity Constraints</i>	–	–	n/a	n/a	✓	✓
<i>Process Model</i>	✓	✓	✓	?	✓	✓
<i>Technology</i>	YAWL	BPMN	UML Activity	–	BPEL, BPMN	UML Activity
<i>Data Flow</i>	Global Data	✓	✓	–	Global Data	Global Data
<i>State Read/Write</i>	–	✓	✓	–	✓	✓
<i>Data Read/Write</i>	–	n/a	–	–	✓	✓
<i>Concurrent Access by Multiple Processes*</i>	–	n/a	n/a	–	✓	n/a
<i>Retrieve Existing Data</i>	–	(n/a)	n/a	–	✓	n/a
<i>Start Process Based on Data Notifications</i>	–	(n/a)	n/a	–	n/a	✓
<i>Continue Process Based on Data Notifications</i>	–	(n/a)	n/a	–	✓	n/a
<i>Access Permissions</i>	n/a	n/a	n/a	n/a	✓	n/a
<i>Data Execution Semantics</i>	✓	n/a	n/a	✓	✓	✓

Table 2.3: Summary of Supported Information and Process Model Features.

Currently available modeling languages do not provide the capabilities to specify the security objects and key management processes that are contained in a security policy. The approaches do either not define execution semantics for data and processes, or have other limitations that makes them unsuitable to model the behavior and management of security objects. The Integrated Modeling Framework for Operational Security Policies adopts concepts from the existing approaches and provides a modeling framework that allows the formal specification of security object and key management processes.

Requirements for an Integrated Modeling Framework

Chapter 3 specifies the requirements of the Integrated Modeling Framework for Operational Security Policies. Section 3.1 introduces the two key usage scenarios of the Policy Support System from the perspective of the end-user. Requirements for the Security Object Modeling Language (SOML) are derived based on these scenarios and the input of security and modeling domain experts from the IBM Business Integration Technologies and Security research groups¹.

Table 3.1 lists the stakeholders of the modeling language. These roles are used to identify who is most likely to make use of a SOML language feature.

<i>Stakeholder</i>	<i>Description</i>
Business Analyst	The business analyst is responsible to model key management processes and security objects. In his role he possesses domain knowledge and is familiar with the modeling language and methodology.
Security Analyst	The security analyst is responsible to annotate the models with security requirements. The security analyst is also responsible to audit a process that has been created by the business analyst.
Developer	The developer is responsible to implement the processes. He is familiar with the modeling and programming language. The developer transforms the processes defined by the analyst into executable artifacts.
Process User	A user that executes a process and interacts with the support system.

Table 3.1: Stakeholders of the Framework

3.1 Usage Scenarios

The Policy Support System is used to execute the models of the Integrated Modeling Framework for Operational Security Policies. The following two main usage scenarios describe, how the system assists end-users with their daily tasks.

¹<http://www.zurich.ibm.com/csc/bit/> & <http://www.zurich.ibm.com/csc/security/>

3.1.1 Scenario 1: Manage Security Objects

The Policy Support System conducts the management of the security objects that are specified in the models of a security policy. A model contains security object blueprints, from which concrete instances are derived at runtime. The security object blueprints specify the information and behavior of the objects. The system supports the end-user with the management of security object instances and allows him to:

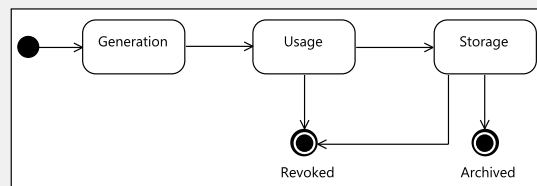
- View existing and archived security object instances, their data and state.
- Extract history data about the evolution of the security object instance.

The system persists data and state of the security object instances either internally or delegates the task to an external system. It ensures that:

- Instances are well formed. To be well formed the data of the instance must be of the correct type and fulfill the constraints as specified in the model.
- Confidential objects instances, their data and state are hidden from unauthorized users.

Example (Manage a Certificate)

A certificate is a crucial security object which an organization has to create and manage during its lifetime. The certificate is created by an authorized user and is valid until it expires or is revoked. Without a support system, the organization would have to track the state and data of the certificate in an additional system or database. The lifecycle of such a certificate can be represented with a state machine:



The following examples show how the Policy Support System ensures that the certificate instance remains valid as it is processed. Additionally, possible consequences that could occur if no system was in place are described:

Constraints	<i>A certificate must have an expiration of type date and the certificate must expire within N years after its creation. Failure to comply with the constraint would indicate that the certificate does not have an expiry date which would violate most security policies.</i>
Permissions	<i>The creator of a certificate can only be viewed by users that have an administration role. Exposing the name to all employees could impose privacy concerns.</i>
State Change	<i>A certificate can only evolve as defined by its lifecycle model. State changes that were not foreseen in the lifecycle model could violate the security policy.</i>
Relationships	<i>The certificate has a relation to the public-key security object which it contains. The certificate must be revoked if the corresponding key-pair changes its state to compromised. A certificate can no longer be trusted if an adversary knows the associated private-key.</i>

Lifecycle management is an important task that has to be completed regardless of whether there is a Policy Support System in place or not. The system facilitates the task by providing a generic solution for the management of all security objects. It automatically supports lifecycle management based on the behavior and information models of the security objects. The system helps to prevent common mistakes that could occur when lifecycle management is done manually:

- The system provides a global solution to access the data and state of security objects. Without such a system, users often rely on different databases and spreadsheets to track their security objects. The generic support system has several advantages over these handcrafted solutions:
 - The same lifecycle and information models are used in the policy documents and by the system. Consistency between the documentation and execution is therefore guaranteed.
 - A single system reduces the chance of redundant data and invalid relationships. The system can ensure the integrity of the relationships since it maintains all security objects.
 - State and data of the instances are always up to date since the Policy Support System combines lifecycle management and process execution.
- Information model and constraints exactly define the expected content of a security object. The system can ensure that only valid data is stored.

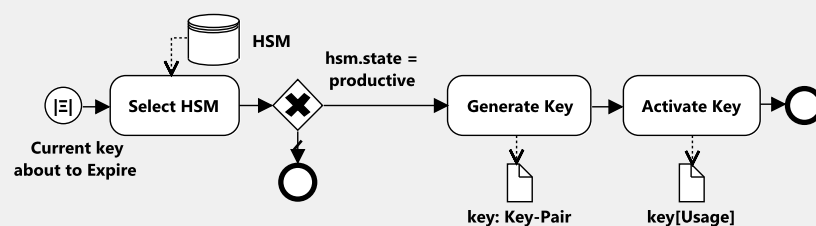
3.1.2 Scenario 2: Execute Key Management Processes

The Policy Support System is capable of executing standard Business Process Model and Notation (BPMN) process models. Additionally, it is aware of the information and behavior of security objects. The system integrates security object and process instances and synchronizes their execution. A process typically works with one or more security object and depends on other objects that exist in the system. The process instance interacts with security object instances in different ways:

- New security object instances are created by the process.
- A process can search for existing security object instances.
- A process can read data or state of a security object instance. The information is used by the process as an input for a choice or like a normal process variable.
- A process can change the data and trigger state transitions in the lifecycle.
- Conditional events stated on the security object can be used to react to events in a security object. A process is either started or continued when such a conditional event occurs.

Example (Key-Pair Generation)

Key-Pair Generation is a fairly complex process that involves different security objects. The process spans over a longer phase of the Key-Pair lifecycle and covers multiple transitions. A simplified version of the process looks as follows:



The following steps are executed to create a new instance of a Key-Pair security object:

1. A new key has to be generate when the currently used Key-Pair is about to expire. The process is therefore automatically started based on the data of the current Key-Pair.
2. The process instance requires a valid Hardware Security Module (HSM) to create the Key-Pair. Therefore it searches a HSM instance and checks whether it is in production mode.
3. The process now creates a new instance of a Key-Pair security object.
4. Once the Key-Pair has been created it has to be activated before it can be used. The activity "Activate Key" performs the required tasks and modifies the state of the security object. The state of the object is changed to "Usage" once the activity completes.

The security architecture of the system relies on the correct execution of the processes defined in the policy. The Policy Support System adequately assists users during their daily work. The system can either automate tasks or support the user if an activity has to be executed manually.

3.2 Modeling Language

Section 3.2 specifies the requirements of the modeling language for security objects (Further called object(s)) and key management processes (Further called process(es)). The requirements for the modeling language are divided into three categories: Security objects are defined in a security object model (Section 3.2.1) and used by key management processes (Section 3.2.2). These processes interact with the security object and used them to synchronize their execution (Section 3.2.3). Examples from a real security policy are used to show which modeling situations make use of non-trivial language features. A complete example process taken from a security policy used in practice is presented in more detail in Section A.5.

Language features are given a priority. The following priorities are used to classify the requirements:

HIGH	<i>The features is required to model the processes of a security policy.</i>
MEDIUM	<i>The feature is optional but can be used to cover additional application scenarios.</i>
LOW	<i>The feature would be nice to have, but is not mandatory.</i>
-	<i>The requirement has been identified but there are no plans to implement the feature in an initial version. The feature is either not directly related to the lifecycle of a security object or is not required for a proof-of-concept.</i>

3.2.1 Modeling Security Objects

Current security policies often contain only superficial specification about the security objects that occur in the policy. Information and behavior of the objects is sometimes not explicitly stated. Analysts either rely on existing standards or implicitly assume that the information and behavior of a security object is well known. Both approaches lead to various problems in practice:

- Often only a small part of a standard is actually used within a policy. Referencing the standard forces the end-user to identify the relevant parts in the standard document.
- Behavior and information defined in a standard sometimes have to be adapted to the specific needs of an information system.
- Relying on implicit knowledge bears tow risks: First of all, end-users might not have the same background as the analyst and therefore have a different understanding about a security object. A policy document is often used over a long period. Implicit knowledge about a security object can however be subject to change over the lifetime of the policy.

The Integrated Modeling Framework for Operational Security Policies encourages explicit specification of important security objects that are used within the policy. Two model types are used to describe the information and behavior of a security object.

Information Model

The information model specifies the properties and relationships of the security objects. The model ensures that the object, which are managed by the system are valid and consistent.

Lifecycle Model

The lifecycle model specifies the dynamic behavior of the security object during its lifetime. It defines the states that a security object may adopt during its evolution. The model defines all allowed transitions that can cause a change of the object's state.

Security Object Information Model

A security object information model is used to model attributes and relations of the objects. The model contains all physical and logical objects that have been identified as important in the context of the domain. The analyst can decide to use independent models to separate objects that are not related to each other. Figure 3.1 shows how the same domain can either be modeled in one combined or in multiple separated models.

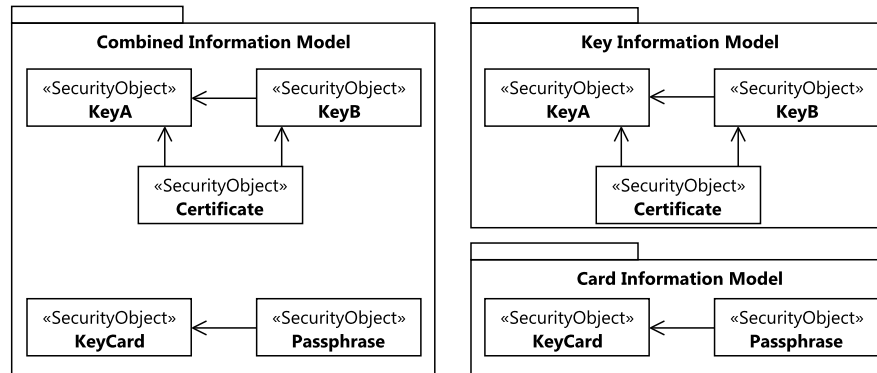


Figure 3.1: Combined and Separated Information Models

Table 3.2 lists all requirements that are related to the security object information model. A detailed description of each requirement can be found in Section A.1.

<i>Description</i>	<i>Primary Role</i>	<i>Priority</i>
Model an object (IM 01)	Business Analyst	High
Define permissions to view object instances (IM 02)	Security Analyst	Low
Model inheritance between two objects (IM 03)	Business Analyst	–
Define the asset classification of an object (IM 04)	Security Analyst	–
Model data properties of an object (IM 10)	Business Analyst	Medium
Define permissions to read a property (IM 11)	Security Analyst	Low
Mark a property as unique identifier (IM 12)	Business Analyst	Low
Model relations between objects (IM 20)	Business Analyst	Medium
Define constraints on data (IM 30)	Business Analyst	–
Define the execution environment of the object (IM 40)	Developer	–

Table 3.2: Security Object Information Model Requirements

Security Object Lifecycle Model

The security object lifecycle model is used to model the dynamic behavior of an object. Each object has exactly one lifecycle model that contains all stages that the object can assume during its evolution. Figure 3.2 shows the security object lifecycle model on the example of a key.

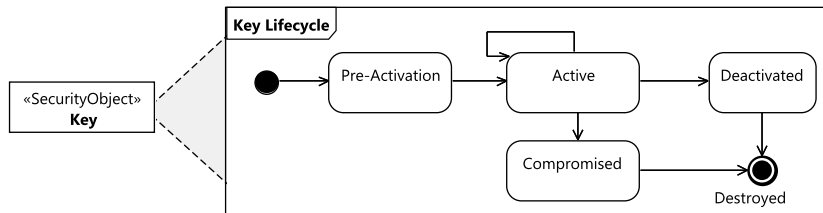


Figure 3.2: Key Lifecycle (As Recommended by the NIST [BBB⁺12])

The lifecycle model presents the objects from a macro perspective. It only contains the fundamental behavior and evolution stages of an object. Key management processes are then used to model the detailed tasks, that are required to manage the object, from a micro perspective. A process can be only responsible for a single evolution step in the objects lifecycle or also cover a broader phase of its lifetime. Figure 3.3 shows how lifecycle and process models are used on a different abstraction level. Process *A* and *B* both cover a single phase in the lifecycle of the object. They are only responsible to transfer the object from one into the other state and do not know the holistic behavior of the object. The processes therefore only show parts of it lifetime from a micro perspective. In contrast, process *C* covers a larger phase of the objects lifecycle and knows parts of the object's behavior.

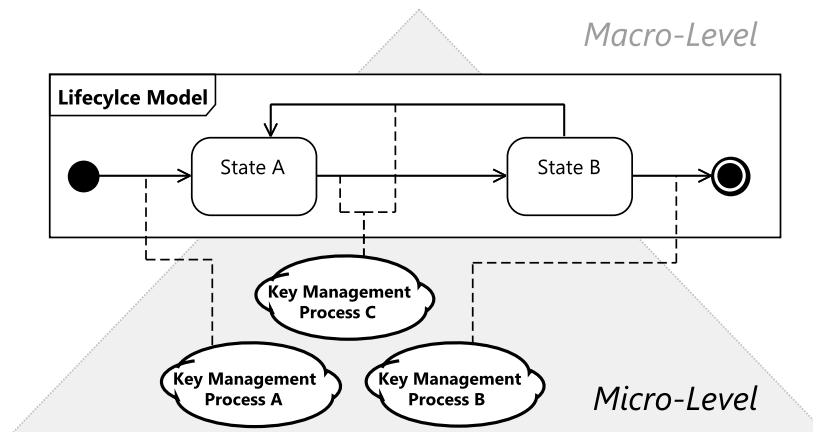


Figure 3.3: Relation between Lifecycle and Process Models

Table 3.3 lists all requirements that are related to the security object lifecycle model. A detailed description of each requirement can be found in Section A.2.

<i>Description</i>	<i>Primary Role</i>	<i>Priority</i>
Model the behavior of an object (LC 01)	Business Analyst	High
Share behavior between objects (LC 02)	Business Analyst	–
Define entry and exit actions of a state (LC 10)	Developer	Medium
Use composite states (LC 11)	Business Analyst	Low
Define constraints on states (LC 12)	Business Analyst	–
Define transition triggers (LC 20)	Business Analyst	High
Model choices (LC 30)	Business Analyst	–
Define transition conditions (LC 31)	Business Analyst	–

Table 3.3: Security Object Lifecycle Model Requirements

3.2.2 Using Security Objects in a Process

Current security policy documents mainly rely on natural language and informal models to specify processes and standard operating procedures. Section 1.1 identified different problems that arise from these practices. The Integrated Modeling Framework for Operational Security Policies encourages the use of BPMN models to specify the operating procedures covered by the policy.

Many processes defined in a policy are working with one or more security object instance. Process models therefore may use Create, Read, Update and Delete (CRUD) [Mar83] operations to interact with security object instances. All features supported by the original process modeling language (BPMN) remain unaffected and are not further specified. From the perspective of a process, the security objects can be seen as normal variables that support additional features.

Table 3.4 lists all requirements that are related to the usage of security objects from within a key management process. A detailed description of each requirement can be found in Section A.3.

<i>Description</i>	<i>Role</i>	<i>Priority</i>
Create new object instances (P 01)	Business Analyst	High
Delete existing object instances (P 02)	Business Analyst	–
Get data and state of an object instance (P 03)	Developer	High
Set data of an object instance (P 04)	Developer	High
Model accepted states of an activity (P 10)	Business Analyst	–
Model produced state of an activity (P 11)	Business Analyst	High
Retrieve existing object instances (P 20)	Developer	Medium
Work on collections of object instances (P 30)	Business Analyst	Low
Annotate the process with security zones (P 40)	Security Analyst	–

Table 3.4: Using Security Objects in a Process

3.2.3 Interaction between Security Objects and Processes

Key management process and security object instances can interact with each other. The modeling language defines the execution semantics of these interactions. An interaction between an object and a process is always defined and controlled by the process model. The process interacts with a security object in two different ways:

- An activity of a process can trigger a transition in the lifecycle of an object.
- A process can define event conditions on the data and state of security objects. Events are generated and consumed by the process when the condition becomes true. Processes use this construct to initiate their execution or continue a running process instance based on the data and state of security object.

Table 3.5 list all requirements that are related to the interaction between key management processes and security objects. A detailed description of each requirement can be found in Section A.4.

<i>Description</i>	<i>Role</i>	<i>Priority</i>
As an action of a change in the object:		
Trigger a conditional start event (I 01)	Business Analyst	High
Trigger a conditional intermediate event (I 02)	Business Analyst	Medium
As an action from within a process:		
Change the state of an object instance (I 10)	Business Analyst	High

Table 3.5: Interaction between Security Objects and Processes

3.3 Design and Execution Methodology

The Integrated Modeling Framework for Operational Security Policies includes the identification, design and execution of security objects in the activities of the BPM lifecycle. Security objects are no longer hidden deep inside the process implementation, but are modeled as first-class citizens. The modeling methodology integrates security objects and processes in the identification, discovery, analysis and re-design activities of the development process.

Both, the process and security object models are enacted to support the execution of processes and manage the security objects. Security object and process models are therefore transformed into executable artifacts and deployed to the Policy Support System. This system assists users during the execution of the processes and manages the lifecycle of the security object instances.

Finally, the Policy Support System allows monitoring and controlling of process and security object instances. The system provides users with an interface to interact with the instances. It captures historic data which can be used to optimize or audit the execution of the security policy.

The Integrated Modeling Framework

Chapter 4 introduces the modeling notation and execution semantics of the Security Object Modeling Language (SOML). Important design considerations regarding the execution semantics of the models are discussed in detail in Section 4.1.3. Section 4.2 elaborates how security policies are designed, implemented and enacted based on the BPM lifecycle. In particular, Section 4.2.2 covers how SOML security object and process design models are transformed into an executable artifact that can be executed by the Policy Support System.

4.1 Security Object Modeling Language (SOML)

We propose the Domain Specific Language (DSL) SOML, to model security objects and key managements process. The language consists of two independent parts to represent security objects and interact with them in a process. Figure 4.1 shows the two components. The security object model allows us the specification of security objects using the Unified Modeling Language (UML) class and state machine diagrams. Key management processes defined in BPMN are then used to specify how instances of these security objects are created and managed. Section 4.1.1 and 4.1.2 contain the specification of the two language parts used to model security objects and processes.



Figure 4.1: SOML Language Components

4.1.1 Security Object Model

The security object model represents a domain model [Fow03] of the security objects. As such it defines both, the data and behavior of the objects which can be any physical or logical asset that is affected by a security policy. These objects are then created, read, updated or deleted by key management processes. The Security Object Modeling Language defines a Domain Specific Language to create such a domain model by extending the UML class and state machine models.

Implementation Details of the Security Object Model

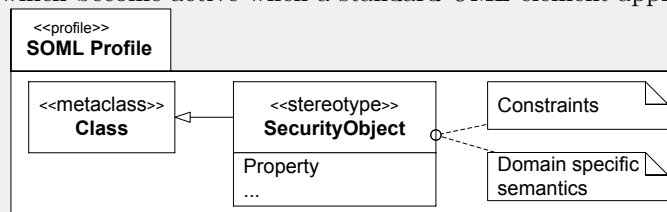
Different techniques to customize UML to a specific domain, such as security policies, are envisioned by the Object Management Group (OMG). The extension capabilities range from lightweight additions using UML keywords or profiles, up to the definition of a new meta-model based on the Meta Object Facility (MOF). A detailed overview of the different extension capabilities has been given by Bruck and Hussey [BH07]. We identified UML profiles as the preferred choice to create the DSL for the security object model. The usage of profiles has two advantages over a lightweight extension using keywords or the definition of a new meta-model:

- The application of UML profiles, as defined by the standard, allows the usage of existing UML tools to create a security object model. Defining a new meta-model would result in incompatibilities between existing tools and the DSL. The definition of a new meta-model would also require the creation of a dedicated editor for security object models. This restriction not only applies to the design-time, where the model is created, but also to the runtime. A profile allows the usage of the existing Eclipse MDT/UML2 libraries and tools to create and parse a model. A profile therefore significantly reduces the implementation effort.
- While a profile does not allow the addition of new elements to the UML meta-model, it may restrict the model by introducing additional constraints. These constraints are capable of restricting the standard UML models to the requirements of the security object model. Profiles can use the Object Constraint Language (OCL) to define the constraints. *NOTE: The current version (Juno) of the Eclipse MDT/UML2 library does not support evaluation of OCL constraints. Both, the editor used to create the models and the runtime are based on the library and cannot execute OCL. The SOML profile does therefore not implement concrete constraints to restrict the model. Failure to comply with the constraints enlisted in this specification will lead to a runtime-error when the model is parsed by the Policy Support System.*

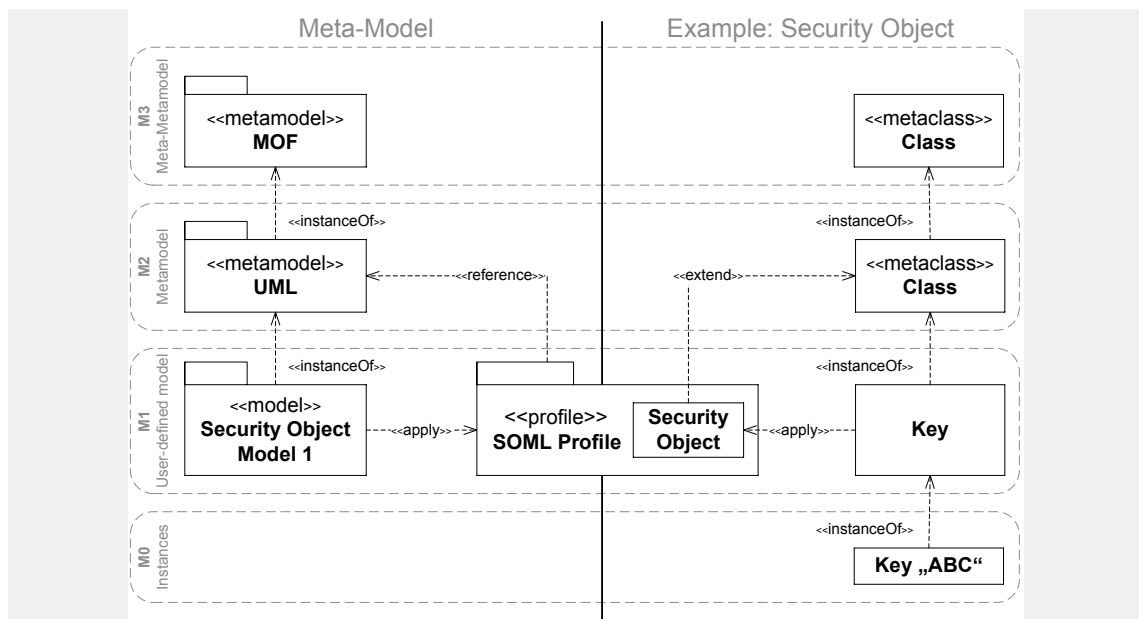
The Security Object Modeling Language defines an UML profile for the specification of security objects using UML class and state machine models. The profile specifies the execution semantics of a security object model that is processed by the Policy Support System.

Background Information (UML Profiles and Meta-Models)

In order to comprehend the SOML security object model, it is necessary to have a closer look at UML profiles and meta-models. The UML Profiles package contains mechanisms that allow meta-classes from existing meta-models to be extended. A profile contains stereotypes which can be applied to standard UML model elements. These stereotypes define additional constraints and extend the meta-class with new properties. Most important, a stereotype can define domain specific semantics which become active when a standard UML element applies the stereotype.



The security object model is an instance of a standard UML model that applies the SOML profile. Therefore all constraints and definitions of the standard UML model remain valid. The profile only introduces further restrictions and defines the concrete execution semantics of the model elements. A security object definition for example, is a normal UML **Class** which applies the **SecurityObject** stereotype of the SOML profile. It is represented on the *M1* level by a instantiating an UML class which resides on the *M2* level. At runtime concert instances of this security object are create on the *M0* level.



All meta-models which are shown in this specification are a subset of the UML *M2* specification. The profile does not introduce new model elements or change existing ones. All it does, is restricting the existing model to a minimal subset that is capable of covering the requirements presented in Chapter 3. Additional details on profiles and meta-models can be found in the OMG specifications [OMG11e,OMG11b].

Figure 4.2 shows the building blocks of a security object model. The model is contained in an UML Model which acts as container for the information and lifecycle model. The information model contains the security objects whose behavior is defined in their corresponding lifecycle model.

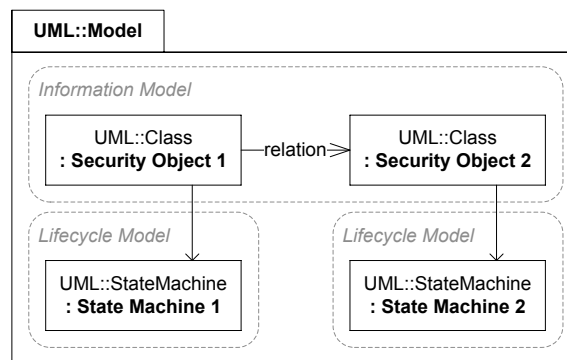


Figure 4.2: The Security Object Model

The remainder of this section defines the SOML UML profile and specifies the execution semantics of a model that applies the profile. Figure 4.3 shows an overview of the stereotypes that are defined by the SOML profile. Additional primitive types, which can be used in a user-defined model, are also included in the profile.

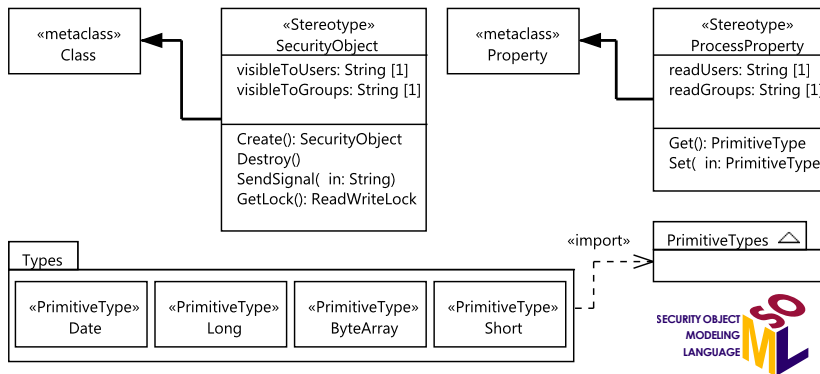


Figure 4.3: Security Object Modeling Language UML Profile

The following conventions are valid for all model elements of the specification:

- UML attributes of meta-classes that are not mentioned are ignored by the Policy Support System. Nevertheless all mandatory UML attributes must be specified to create a valid model.
- Information and lifecycle model use the notations from class and state machine diagrams.
- The following terms are used to describe attributes of a meta-class in the SOML profile:

- Label** The value of attribute is used as label of the meta-class that owns the attribute. A label is shown to the user when the corresponding meta-class is visualized.
- Identifier** The value of attribute is used as identifier of the meta-class.

Security Object Model Container

Metaclass: UML::Model	Profile Application: SOML
Metaclass Attributes:	
<i>Name</i>	<i>Type / Description</i>
name	String [0..1] <i>The identifier and label of the security object model. A unique name must be provided when multiple models are loaded by the Policy Support System. Elements of the model are accessed using the fully qualified name: <modelName>::<elementName></i>
Execution Semantics:	
UML models which apply the SOML profile are loaded and executed by the Policy Support System. The models must be serialized in XML Metadata Interchange (XMI) format as specified by the UML specification [OMG11e]. The model itself only acts as container and has no execution semantics. An arbitrary number of independent security object models can be loaded.	
Profile Constraints:	
<ul style="list-style-type: none"> • The model may only directly contain elements of type UML::Class and UML::Association. • All contained elements of type UML::Class must have stereotype SOML::SecurityObject or a sub-stereotype of SOML::SecurityObject. • The property name must be a valid Java identifier [GJSB05]. 	

Primitive Types

The SOML profile contains additional primitive types, which can be imported by a security object model. These types are formed from the UML `PrimitiveTypes` package [OMG11d] and the SOML `type` package. Only the types shown in Table 4.1 are supported by the Policy Support System.

<i>UML Type</i>	<i>Corresponding (Java) Type</i>
<code>UML::Boolean</code>	<code>Boolean</code>
<code>SOML::ByteArray</code>	<code>Byte []</code>
<code>SOML::Date</code>	<code>Date</code>
<code>UML::Real</code>	<code>Double</code>
<code>UML::Integer</code>	<code>Integer</code>
<code>SOML::Long</code>	<code>Long</code>
<code>SOML::Short</code>	<code>Short</code>
<code>UML::String</code>	<code>String</code>

Table 4.1: Supported Types of the Policy Support System

Information Model

The information model is contained in the security object model and uses an UML class diagram to specify the security objects, that are relevant in the context of the application domain. The model contains one or more security object(s). Figure 4.4 shows the meta-model of the information model.

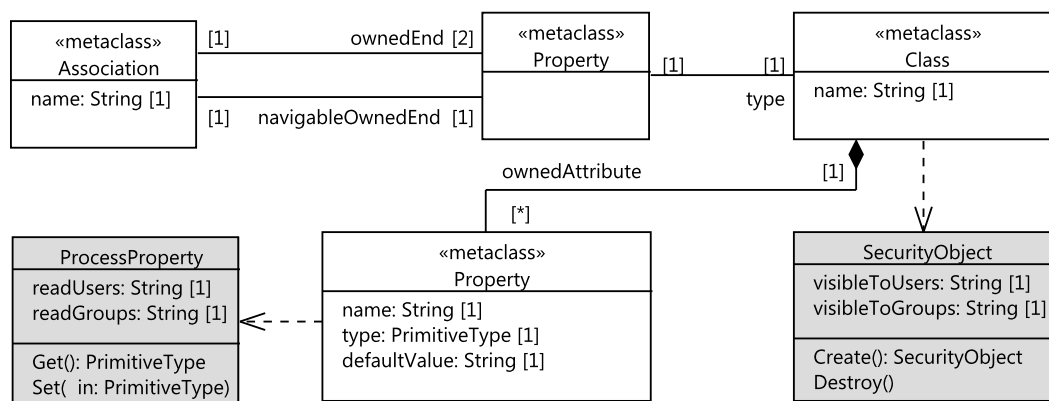
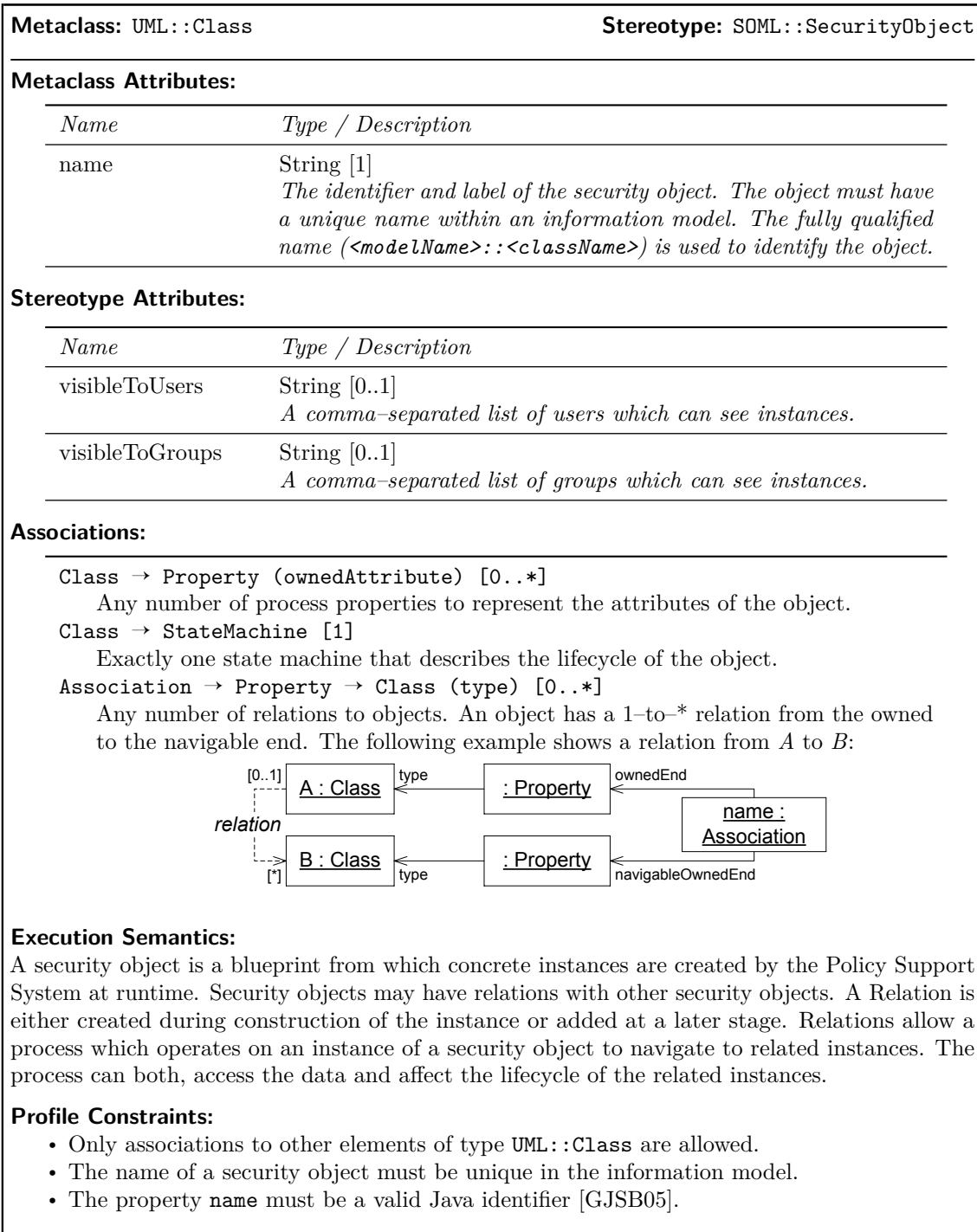
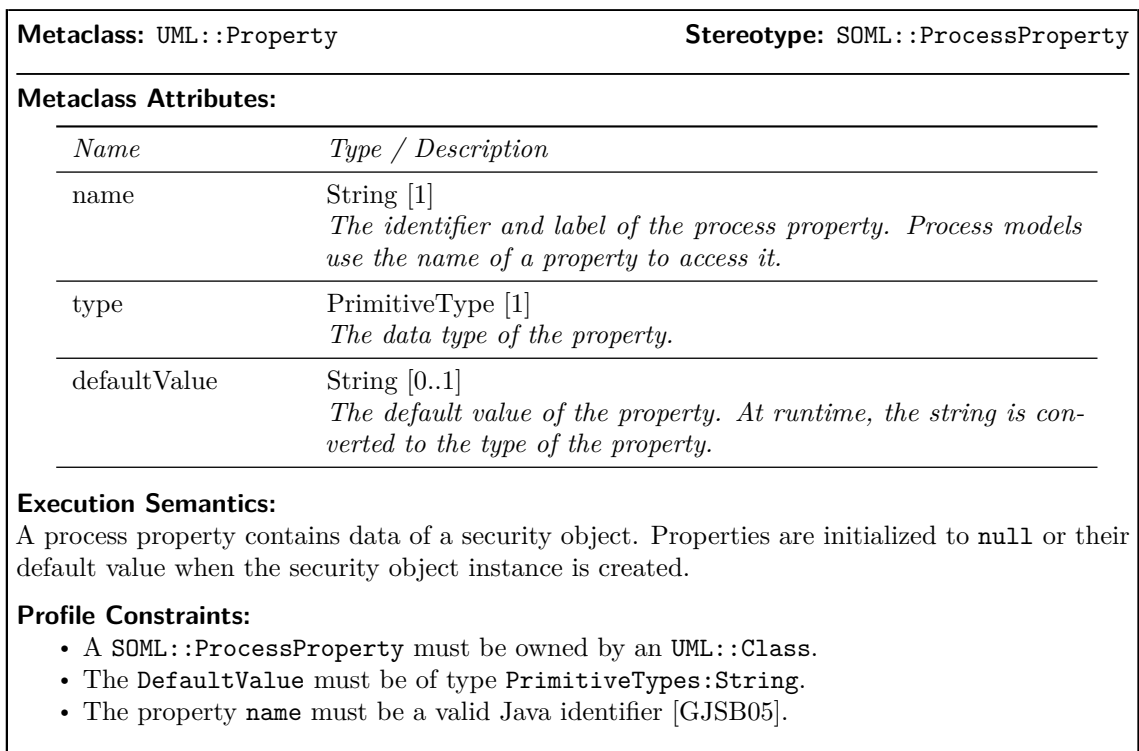


Figure 4.4: Meta-Model: Information Model

SecurityObject: A security object represents a physical or logical object that is relevant in the context of the domain. It consists of information (attributes), a lifecycle and relations to other objects. The object is defined using an UML `Class` that applies the `SecurityObject` stereotype.



ProcessProperty: A process property represents a primitive data attribute that is managed and stored by the Policy Support System and assigned to a security object instance. The process property supports the types defined in Section 4.1.1.



Lifecycle Model

Each security object exhibits a lifecycle that specifies the states and transitions it runs through during its evolution. The lifecycle is represented with an UML state machine which is owned by the security object. Execution semantics of UML state machines [OMG11e] and statecharts in general [VdB94] are increasingly complex the more language features are used. To simplify the definition of security object lifecycles, simple execution semantics based on the notion of a Finite State machine are used. Figure 4.5 shows the meta-model of the lifecycle model.

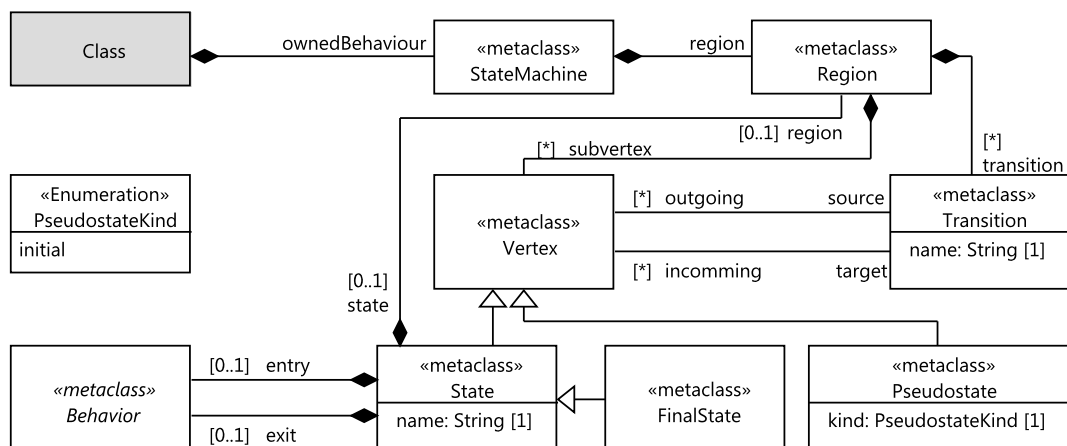


Figure 4.5: Meta-Model: Lifecycle Model

StateMachine: A subset of the UML state machine package is used to represent the lifecycle of a security object. In particular the use of parallel constructs (Multiple regions, forks, joins, ...) is prohibited. The purpose of the lifecycle model is to act as lightweight workflow that synchronizes different processes over the lifetime of the security object. Lifecycle models therefore only support few constructs and should remain simple. Detailed behavior and processing rules of a security object are typically modeled in the associated processes. A lifecycle state machine therefore contains exactly one region with the following sub-elements:

Metaclass: UML::State, UML::Pseudostate, UML::Finalstate		Stereotype: -
Metaclass Attributes:		
<i>Name</i>	<i>Type / Description</i>	
name	String [1] <i>The identifier and label of the state.</i>	
Execution Semantics:		
A state models a situation during which the security object is in a certain state of its lifecycle. While the object is in a state it is waiting for external events (Triggers) to occur. The state may specify an entry and exit action that is executed upon entering or leaving the state (See Section 4.1.1). Besides normal states there are two special cases of a state:		
<ul style="list-style-type: none"> • A final State is a special kind of state, signifying that the enclosing region is completed. If the enclosing region is directly contained in a state machine, then it means that the entire lifecycle is completed. • When a security object is created its lifecycle starts at the initial state. The transition connecting the initial to the first state is automatically executed upon creation of the object. 		
Profile Constraints:		
<ul style="list-style-type: none"> • A UML::State may have an entry/exit action of type UML::OpaqueBehaviour • The property name must be a valid Java identifier [GJSB05]. • A lifecycle model must contain exactly one initial state and at least one final state. 		

State Transitions: A state transition models a possible state changes between two states.

Metaclass: UML::State		Stereotype: -
Metaclass Attributes:		
<i>Name</i>	<i>Type / Description</i>	
name	String [1] <i>The label of the transition.</i>	
Execution Semantics:		
A transition models a possible change from a source into a target state. A state may have multiple outgoing and incoming transitions. A transition is triggered when a change state signal with the identifier of the target state is received. The name of a transition is only used as label.		

State Behavior: State behavior is used to model actions that are executed upon entering or leaving a state. Figure 4.6 shows the meta-model of state entry and exit actions.

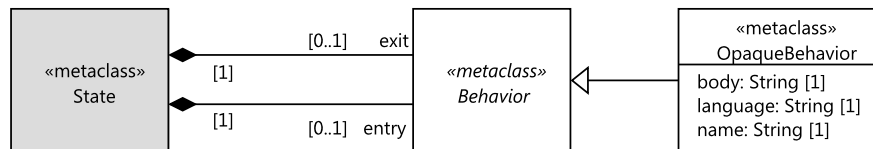


Figure 4.6: Meta Model: State Behavior

Metaclass: UML::OpaqueBehavior	Stereotype: -
Metaclass Attributes:	
<i>Name</i>	<i>Type / Description</i>
name	String [1] <i>The label of the behavior.</i>
language	String [1] <i>The programming language of the body.</i>
body	String [1] <i>The code to be executed at runtime.</i>
Execution Semantics:	
A state may specify OpaqueBehavior (s) that are executed by the engine when entering/leaving the state. They must define valid code (body) in the language specified by the language property. Return values of code running in an action are ignored. In the lifecycle diagram the property Name is displayed in place of the code.	

An entry or exit behavior can be defined using one of the programming languages: JUEL, Groovy or JavaScript. The scripts have access to the security object instance on which the state change is executed. Listing 4.1 shows how scripts access the security object.

```
// Entry and exit action scripts can use the "instance" variable.
// The variable is automatically bound to the executing instance.
instance.setAttribute("date", new Date()); // Assign the current date to the attribute.

// Likewise an Activiti process script can use "execution" to access the current process.
// execution.setVariable(...)
```

Listing 4.1: Example: Access to a Security Object Instance in a Groovy Script

A security object model used in the design phase can also make use of natural language to informally define entry and exit action. These actions must be converted into a valid programming language if the process is implemented.

4.1.2 Key Management Processes

Section 4.1.2 introduces the modeling constructs used to define interactions with security objects in a key management process. A key management process is developed in different phases. During an initial analysis and design phase, a business analyst creates a coarse model of the processes which is later refined. A developer transforms these design models into executable artifacts which are enacted by the Policy Support System. SOML models used during the design phase can make use of syntax abbreviations and notations that are not executable. Complex features, such as queries and conditional events, allow the usage of natural language to simplify the development process. The design model must therefore be transformed into an executable model before it can be enacted.

Key management processes are modeled using the standard BPMN notation and execution semantics. The difference between a BPMN and SOML process model is, the interaction with security object that is specified in a SOML model. SOML uses the BPMN data modeling elements to model interaction with security objects. Our language therefore defines semantics of the data modeling element for which the BPMN standard does not provide concrete execution semantics.

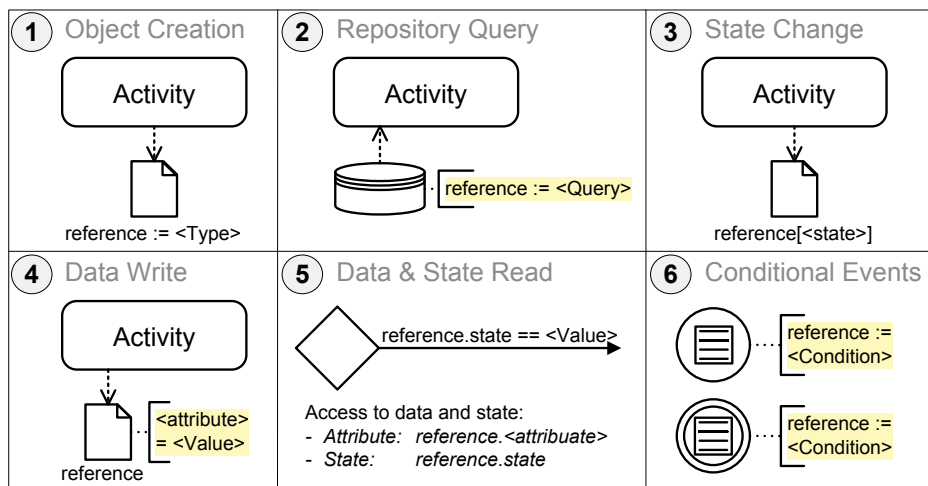


Figure 4.7: SOML Data Modeling Elements

Figure 4.7 shows an overview of all available modeling constructs to interact with security objects. Each construct is described separately in Section 4.1.2. First of all, the general concepts of data scope and data collections, which are valid for all elements are introduced. Many concepts are tightly coupled to the capabilities of the Policy Support System that is used to execute the processes. The execution semantics and capabilities of the Policy Support System have therefore been taken into account when defining the SOML data modeling elements. Security object instances and standard data are used the same way in an executable model. The Policy Support System which enacts the key management processes is based on the Activiti process engine. Execution semantics of the security objects instances therefore comply with the semantics of Activiti process variables.

Data Flow and Lifetime: The BPMN standard defines complex semantics to model data flow within a process or sub-process. Data objects can either be referenced using **Data Object References** or explicitly passed between tasks with **Data Associations** [OMG11a]. Many execution engines only allow global process variables and do not support sophisticated data flow patterns [RTHEvdA05]. Figure 4.8 shows the different possibilities to model data flow in BPMN:

- A is used by the 1st activity and subsequently referenced by a Data Object Reference.
- B is only used by the 2nd activity.
- C is passed from the 3rd to the 4th activity with a Data Association.

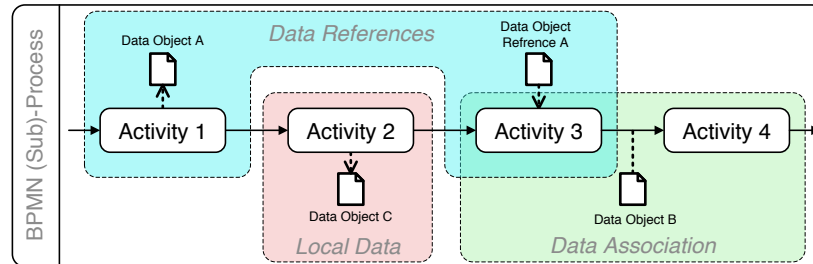


Figure 4.8: BPMN Data Flow

The Activiti engine only supports two types of process variables to store data. Global process variables are in the scope of a (sub-) process and can be accessed by all its elements. Local variables have the scope of an activity and are accessible only by the activity. Figure 4.9 shows how object A and C shown of Figure 4.8 are translated into process variables while object B is represented as local variable. To remain consistent with normal data, security object instances are treated as Activiti process variables. The use of Data Objects, Data Associations and Data Object References is not supported in an executable model.

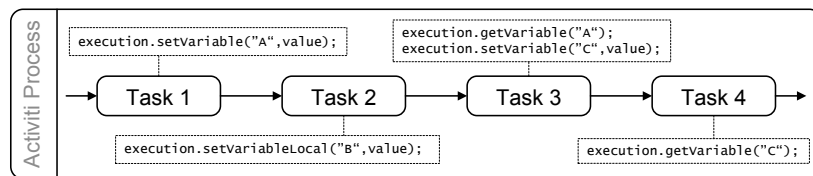


Figure 4.9: Activiti Process Variable Scope

The lifetime of a BPMN data object or Activiti process variable is bound to the lifetime of its parent process or sub-process. All data objects are deleted when the process instance gets disposed. At this point, the data is no longer available. Security object instances behave differently than normal process variables. They are independent of a process instance and are not bound to their lifetime. To be precise, a process variable never contains a security object instance but only a reference to the object. Once the process completes, this reference is disposed but the instance remains.

Data Object Collections: Simultaneous access to more than one instance of a data object is modeled using BPMN Data Object Collections. Designer may use the collections to indicate that multiple security object instances are affected by an action. Executable SOML models cannot make use of collections as the Activiti engine does not support them. Collections must either be transformed into custom code or multi-instance activities. Figure 4.10 shows a BPMN collection which indicates that the state change command is executed for all instances in the collection. In the executable model, this construct is transformed into a multi-instance activity that is executed for each instance in the collection.

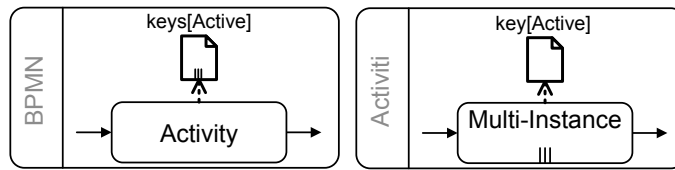


Figure 4.10: Data Object Collections and Multi-Instance Activities

Object Creation (1)

Security object instances are created by key management processes. The process models use a BPMN `DataObject` to model the creation of an instance. The name of the `DataObject` specifies which instance type is created and bound to a process variable:

```
// Create an object of type <policy::key> and bind it to <currentKey>.
currentKey := policy::key;
```

Listing 4.2: Example: Object Creation (Formal Syntax is Defined in Section B.1)

Upon completion of the activity to which the `DataObject` is attached, a new instance of the security object is created and assigned to the process variable. As the object is created, its attributes are initialized to their default values and the lifecycle starts at the initial state. The variable has implicitly the scope of the whole process.

Repository Query (2)

Key management processes either obtain a reference to a security object instance if they have created the instance or by querying the global object repository. Each object instance is stored in this repository from where it can be retrieved. Process models utilize BPMN `DataStores` to specify queries which search for security object instances. A query is evaluated when the activity to which the `DataStore` is attached completes. The results of the evaluation are zero or more security object instances which match the query and are bound to the process variable. Constructs for data collections must be used to work with the result if a queries returns multiple results. An `Annotation` attached to the `DataStore` contains the query in natural language or as a formal query:

```
// Natural language query. Must be transformed if implemented.
masterKey := "Key where attribute type is equal to Master_Key"

// A query searching for exactly one result.
// Searches for all keys which are of type <key> where the attribute
// <type> is equal to the string "Master_Key". The query expects exactly
// one result. Finding zero or more results is considered as an error.
masterKey := {instanceof policy::key && type == "Master_Key"}.selectOne();

// A query searching for one or more results.
// Searches for all keys which are of type <key> and in state <Active>
// The result is a set of security object instances. The process model must
// treat the result as collection.
activeKeys := {instanceof policy::key && inState(Active)}.selectMany();
```

Listing 4.3: Example: Different Queries (Formal Syntax is Defined in Section B.1)

State Change (3)

All state changes that occur in a security object instance are requested by key management processes. BPMN `DataObjects` are used to model changes to the state of a security object instance. A state change is requested upon completing the activity to which the `DataObject` is attached. The name of the `DataObject` is used to model a state change:

```
// Change the state of the security object instance to <Archived>.
// The process must have initialized the variable <masterKey> beforehand.
masterKey[Archived];
```

Listing 4.4: Example: State Change (*Formal Syntax is Defined in Section B.1*)

A state change request is executed by the security object in best-effort. A BPMN `Error` event is created, if a state change cannot be executed. This might occur if two processes simultaneously request inconsistent state changes. Using the event, processes may react to failed state changes.

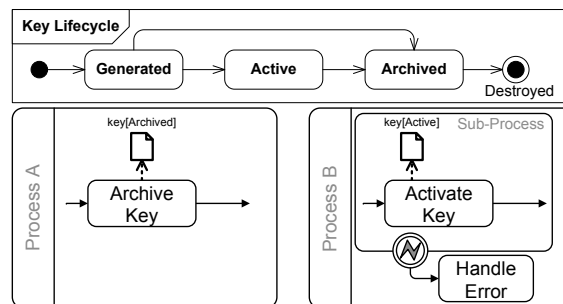


Figure 4.11: Using the BPMN Error Boundary Event to Resolve an Inconsistent Change

Figure 4.11 shows an example, where two processes refer to the same object instance. The transition to `Active` will fail, if process `A` is executed before process `B`. It is not necessarily the case that this situation is an error in the process models. The process `B` can therefore catch the error event and react to a failed state change. Design considerations behind the execution semantics of state changes are discussed in more detail in Section 4.1.3.

Data Write (4)

Process models can change the attributes of a security object instance. BPMN `DataObjects` and `Annotations` are used to model data write operations on object instances. The `DataObject` is annotated with a text that defines what data is written to the instance once the activity completes:

```
key.type = "Test";           // Set the value to a static literal.
key.size = SelectedSize;    // Set the value of another process variable.
key.hsm.hasKey = true;      // Set a nested attribute

// Set multiple related security object instances
// C* are process variables which point to security object instances.
key.certificates = {C1, C2, C3};
```

Listing 4.5: Example: Setting Attributes (*Formal Syntax is Defined in Section B.1*)

Data/State Read (5)

Process models can access the attributes and state of the security object instances to which the process holds a reference. The information can be used to model conditional **SequenceFlows** as defined by the BPMN standard. The Activiti engine therefore defines an expression language based on JUEL to implement expressions which evaluate to a boolean value:

```
// Comparison of an attribute with a literal.
// Flow is taken if <key> is a master key with a size greater than 1024.
key.type == "Master_Key" && key.size > 1024

// Flow is taken if the key is compromised or not located on a test HSM.
key.state == "compromised" || key.hsm.state != "Test"

// Comparison of an attribute with process variable <selectedType>.
key.type == selectedType

// Comparison of the current state with a literal.
key.state == "Active"
```

Listing 4.6: Example: Different Conditional Expressions (See JUEL Language Specification)

An executable process can also make use of the data in any activity that is capable of working with process variables. Security object instances can therefore be used in custom code or user activities like normal Activiti process variables.

Conditional Event (6)

BPMN conditional events can be used to start and continue a process instance. The BPMN standard does however not define an executable condition language. SOML allows the definition of events which state a condition about a security object type. The conditional event is triggered whenever the condition changes from false to true for a concrete instance of a security object type.

```
// Natural language condition. Must be transformed if implemented.
newKey := "A new key is created"

// Start process whenever an instance of a <key> is created.
// The process gets a reference in the process variable <newKey>.
newKey := policy::key.{};

// Start process whenever an instance of a <key> enters state <Active>.
// The event is triggered each time the key re-enters the state.
activeKey := policy::key.{inState(Active)};

// Start process when an existing key expires. ("now" = current data/time)
expiredKey := policy::key.{expiryDate <= now};

// Start process when an existing key expires in the next day.
// (86400000 = 1ms * 1000 * 60 * 60 * 24 = 1 day)
toExpire := policy::key.{expiryDate <= now + 86400000};
```

Listing 4.7: Example: Conditional Start Events (Formal Syntax is Defined in Section B.1)

4.1.3 Execution Semantics Rationale

Different execution semantics could be envisioned to enact SOML security object and key management process models. In Section 4.1.3 we discuss the design considerations of controversial language features in more detail.

Concurrent Access to Security Object Instances

Simultaneously running process instances can use SOML queries to retrieve the same security object instance. The execution semantics must therefore consider concurrent access to a security object instance. SOML has two possibilities to deal with concurrent access:

- Avoid concurrent access by restricting a security object instance to a single process.
- Deal with concurrency by using synchronization primitives such as locks.

Enforcing a strict association between a security object and process instance shifts the problem towards the execution semantics of queries. A query would either not be allowed to return an instance that is already in use, or would block until the instance is released. Both approaches would reduce the utility of queries and increase their complexity. The problem of concurrent access is not solved even if a security object is accessed only by a single process. BPMN supports a variety of constructs that allow parallel execution within a single process. SOML therefore allows concurrent access and uses synchronization to avoid inconsistent changes to security object instances.

Changes to security object instances from within processes are synchronized using an optimistic offline lock [Fow03]. Each data modeling construct introduced in Section 4.1.2 is thus atomic, meaning that it is either completely executed or has no effect. Changes to the security object instance and process are rolled back if an optimistic offline lock yields a conflict.

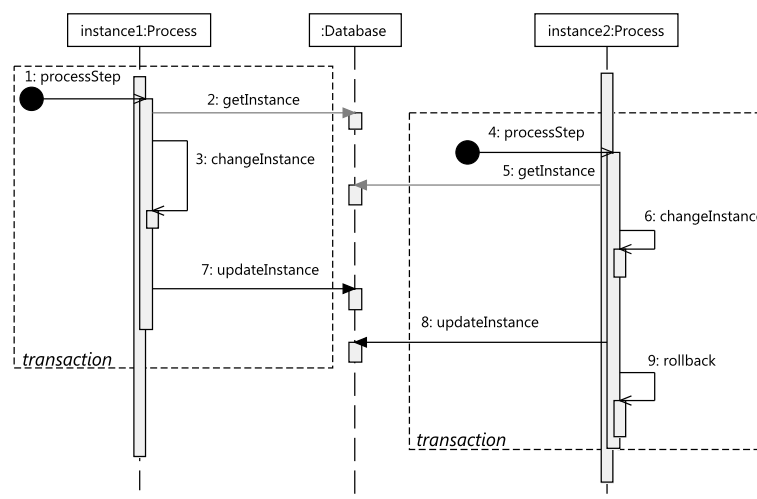


Figure 4.12: Optimistic Offline Lock and Rollback

Figure 4.12 shows an example where two processes simultaneously update the same security object instance. Both process instances execute a process step which could, for example, be triggered by completing a user form. The Activiti process engine uses transactions to execute the two process steps. A process step is triggered externally and executes the process model until the process is

blocked and waiting for external input. A change to the security object occurs as part of both process steps. The processes therefore obtain the current data of the security object instance from the database and change the object locally. Later the processes try to write the changes back to the database. The second process which updates the object will detect a conflict and initiate a rollback. The rollback causes all changes of this transaction, including the changes to the security object and process execution, to be discarded. The second process instance is therefore in the exact same state as before the process step was requested. The initiator of the process step is notified about the exception and can react to the conflict. (E.g.: Inform the user that the object has changed and he needs to resubmit his changes.)

The usage of optimistic offline locks has several advantages compared to other synchronization mechanisms. The main reason for choosing optimistic locks where:

- From the perspective of a designer it is desirable that changes to object instances are atomic and require no complex locking mechanisms. Optimistic locking allows atomic operations without the usage of explicit lock constructs in the models.
- Our initial modeling sessions showed, that security object instances are often changed by a single process at a given time. The chances of a write conflict is therefore low. Optimistic locking allows other processes to read while another process is changing the object.
- The Activiti process engine already uses optimistic offline locks to prevent concurrent execution of the same process instance on multiple nodes. Using the same lock types allows reuse of the existing infrastructure to roll back the complete transaction.

The second choice about locking is, whether to allow a process instance to exclusively lock a security object instance for a longer period. A process which owns a lock can expect that an object does not change. Figure 4.13 shows an exemplary use of an explicit lock request in a process model. The lock is used to ensure that the key cannot be changed by another process while the long-running task is running. Our analysis of an existing policy showed that security object instances are typically used by a single process at a point in time. Therefore we decided not to include explicit locks in the modeling language.

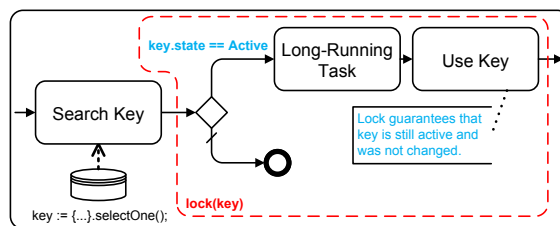


Figure 4.13: Obtaining an Explicit Lock in a Process Model (*Not Implemented*)

Alternatives for State Change Semantics

A transition between two states of the security objects lifecycle can be realized using different execution semantics. The execution semantics affect how state changes are incorporated into process models. We identified the following two key aspects of the state change operation:

Notion of the State Change Language Construct

Process models can either use a descriptive or prescriptive construct to specify the intent to change an object's state. The descriptive variant uses a state change requests which describes the intention to change to object's state. The prescriptive construct applies a pre-

and post-condition. The pre-condition dictates the accepted state(s) of an activity while the post-condition defines the state of the security object when the activity is completed.

Response to an Illegal State

Process models can request a transition to a state which is not reachable from the current state of the security object. The language must deal with illegal state change requests. It can either assume that the illegal request indicates an error or block the state change until the transition becomes valid. An illegal state change can also occur if a process model makes use of pre-conditions to define the accepted state(s) of an activity. The execution semantics must define a reaction, if an object is not in an accepted state when the activity starts. Likewise the execution semantics can treat this case as an error or wait until the object is put into a state that matches with the pre-condition.

Table 4.2 shows the combination of the two aspects and describes the resulting execution semantics.

		<i>Response to an Illegal State</i>	
		<i>Exception</i>	<i>Blocking</i>
<i>Notion of the State Change Construct</i>	<i>Descriptive</i>	<p>1: Descriptive-Exception The activity defines a produced state, which describes the activity's intent to change the state of the object. The state change is executed in best-effort manner and throws an exception, if it is illegal. Optionally a process model may catch the error and resolve an illegal state change.</p>	<p>2: Descriptive-Blocking The activity defines a produced state, which describes the activity's intent to change the state of the object. The state change blocks the execution of the process if the state transition is currently not possible. The state change is executed and the process continues once the transition becomes valid. Therefore another process must have changed the state of the object.</p>
	<i>Prescriptive</i>	<p>3: Prescriptive-Exception An activity defines an accepted state, in addition to the state that is produced when the activity completes. Upon entering the activity, the current state of the security object is compared to the accepted state and an exception is thrown if the object is not in the accepted state. The presence of the accepted state in the models provides additional information to the reader. It even allows the generation or validation of the lifecycle models based on the process models.</p>	<p>4: Prescriptive-Blocking An activity defines an accepted state, in addition to the state that is produced when the activity completes. The activity can only be started if the object instance is in the accepted state. The object instance is locked and can only be accessed by the activity. The modeler can thus be sure, that the object is still in the accepted state. Process models can be statically validated and illegal state changes prevented. During execution the process risks to reach a dead-lock when an accepted state never occurs.</p>

Table 4.2: Different State Change Semantics

The orthogonal choices can be combined into different execution semantics. The 1st and simplest semantic uses descriptive state change requests and treats any illegal state change as an error which must be handled by the user. This execution semantic is both intuitive to use and requires no additional modeling constructs. The other semantics introduce blocking or pre-conditions.

Blocking until a state change is possible or an accepted state is reached, removes the possibility of errors during the process execution. Blocking does however not solve the problem. Instead of throwing an exception, processes may now wait indefinitely for a state that will never occur. Blocking guarantees the correctness of state operations at the expense of liveness of processes.

The main advantage of introducing pre-conditions, which define accepted state(s) of an activity, lies in the additional information contained in the process models. The process models now contain all possible transitions between the states of a security object. The lifecycle model of a security object could therefore be derived or validated based on this information.

Conclusion: The 2nd and 3rd execution semantics presented in Table 4.2 only provide few benefits over the simple semantics relying on descriptive state changes and exceptions. The choice is therefore mainly between the two extreme variants, *Descriptive-Exception* and *Prescriptive-Blocking*. Implementation wise, *Prescriptive-Blocking* builds up on the simpler *Descriptive-Exception* semantics. Therefore, and since we could not identify the need for the complex *Prescriptive-Blocking* variant, we chose the execution semantics based on descriptive model constructs and exceptions.

The *Prescriptive-Blocking* variant still has some compelling properties compared to the other execution semantics. Once an activity is executed, its state changes are guaranteed to be completed. In comparison to the other approaches, data can also be used to model implicit control flow. Figure 4.14 shows an example that uses a data object to model implicit control flow. In the example, the *Task A* has to be executed before *Task D*. This constraint is modeled using the pre-condition $obj[S2]$ on the *Task D*. Modeling the same scenario only using normal control flow is difficult. The naive approach shown in Figure 4.14 does not have the same semantics. *Task b* cannot be executed before *Task c* anymore.

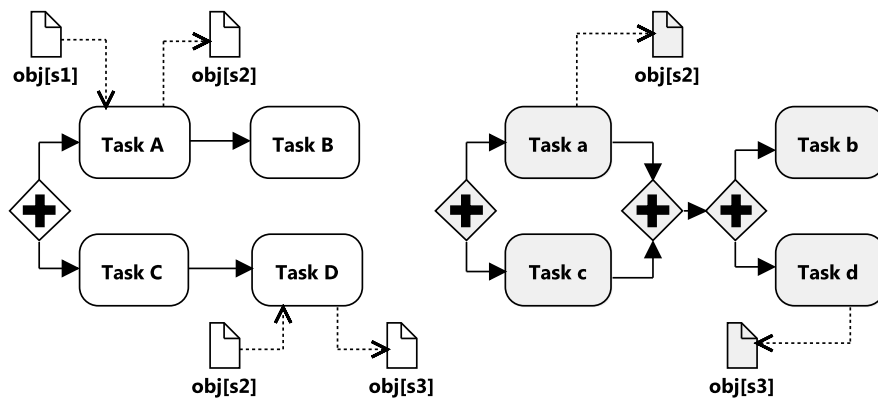


Figure 4.14: Implicit Control Flow (L) vs. Normal Control Flow (R)

4.1.4 Requirements Covered by the Language

Table 4.3 lists the requirements presented in Chapter 3 and their corresponding language feature of the Security Object Modeling Language. Some requirements, which we identified as non-critical for the proof-of-concept, are not implemented by the language. The contains the motivation to omit certain modeling features and reduce the implementation effort.

<i>Language Feature</i>	<i>Implements Requirement(s)</i>
Information Model	IM 01, IM 02, IM 10, IM 11, IM 12, IM 20
Lifecycle Model	LC 01, LC 10, LC 20
Data Modeling Elements	P 01, P 03, P 04, P 11, P 20, P 30, P 01, I 02, I 10

<i>Requirement(s)</i>	<i>Rationale for Omission</i>
IM 03, LC 02, LC 11, LC 30, LC 31	<i>These features would simplify the modeling process but can be replicated using existing constructs. Therefore they are not mandatory for a proof-of-concept.</i>
IM 30, LC 12	<i>The library that is used to parse the model does not support OCL constrains. Implementing them would greatly increase the implementation effort.</i>
IM 40	<i>Access to external system is beyond the scope of the initial prototype.</i>
IM 04, P 40	<i>The concept of security zones is beyond the scope of the initial prototype.</i>
P 02	<i>A security object instance is never deleted. It is locked once a final state has been reached.</i>
P 10	<i>The chosen execution semantics render the feature obsolete. See Section 4.1.3 for further explanations.</i>

Table 4.3: Traceability between Language Parts and Requirements

4.2 Designing and Enacting the Security Policy

Section 4.2 illustrates how security objects and their usage in key management processes fits into the BPM lifecycle. The Integrated Modeling Framework for Operational Security Policies proposes an adapted form of the BPM lifecycle and methodology introduced by Dumas [et al.] [DLRMR13].

4.2.1 Designing the Security Policy

Using formal models to design a security policy requires the specification of security objects and key management processes that occur in the policy. The Integrated Modeling Framework for Operational Security Policies uses the same techniques, known from BPM, to identify and design key management processes. The main difference between a business process and a key management process is the intended result of the process. The purpose of many business processes is, the production of a good or service for a customer. Instead, key management processes are used to create and manage security objects. Compared to a normal process, where the output is not formally modeled, security objects are subject to the same formalism as processes. The methodology must therefore combine the design of key management process and security object models. Figure 4.15 shows the steps that are required to create the models of a security policy.

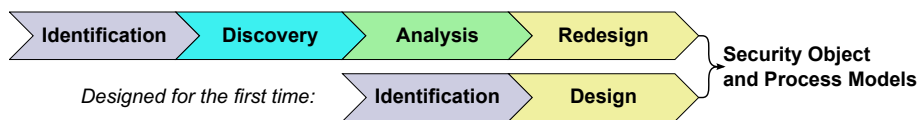


Figure 4.15: BPM Process Identification and Design [DLRMR13]

Key management processes and security objects that are relevant for a security policy must be identified before they can be modeled. Compared to a business, where many processes are already in place and have to be discovered, analyzed and redesigned, a policy is often created from scratch. The discovery, analysis and redesign phase can therefore be consolidated in a single design phase. The remainder of this section introduces the methodology used to identify and design security objects and key management processes based on the work of Dumas et al. [DLRMR13]. The Integrated Modeling Framework for Operational Security Policies currently only covers the creation of new security policies. Discovery, analysis and improvement of existing policies is not in the scope of the thesis. The same methods as used for business processes could also be applied to the design of existing policies. In particular, the following topics would be of importance for security policies:

- Discovery of ad-hoc processes that are executed, but not covered by the policy. These undocumented processes increase the risk of mistakes and pose a threat to business continuity if key personnel is leaving the organization.
- Analysis of processes to assess compliance of process execution with the policy documents.
- Analysis and reduction of errors and risks in existing processes. Mistakes in security processes often involve severe consequences. Compared to business process, one would therefore be more concerned to increase process-reliability and quality instead of performance.

Security Object and Key Management Process Identification

During the identification phase, the security objects and key management processes of a security policy are systematically elicited. The goal of this phase is to create a process architecture which gives an overview of the processes that occur in a policy. The processes which are implemented are selected based on this process architecture. Figure 4.16 shows how a process architecture can be modeled on different levels of details. In contrast to a normal modeling initiative, the specification of a policy also includes the definition of security objects.

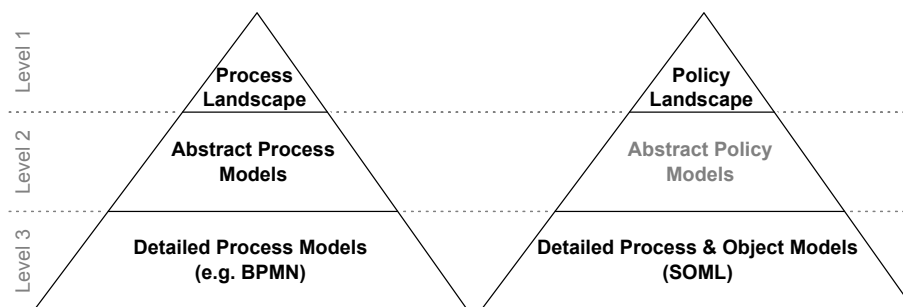


Figure 4.16: Process Architecture [DLRMR13] Applied to Security Policies

On the first level, a policy (or process) landscape provides an overview of the processes and security objects that occur in the policy. Each process and security object that occurs in the landscape is modeled in more detail in the 2nd and 3rd level of the process architecture. The policy landscape is used to decide, which processes and objects are modeled in more detail or implemented in the Policy Support System. On the 2nd level, abstract models are used to provide deeper insight on the processes and security objects. Alternative paths or exceptions are often not presented on this level of detail. These formal, but incomplete definitions could be applied to parts of the policy, which are not worthwhile to be modeled in full detail. The specification of the abstract models is not further covered by this thesis. Simply put, these models are the same as on the 3rd level but omit certain details. Key management processes and security objects that have been identified as eligible for the detailed design and implementation are modeled in full detail on the 3rd level.

Deriving a Process Architecture: The first step to create a process architecture is to identify the key management processes and security objects. The framework applies the same method to derive a policy landscape that is used to create a process landscape [DLRMR13]. The policy landscape includes security objects in addition to the key management processes. Our approach consist of the same four steps as the original method but uses security objects instead of business case types:

Identify Security Object

At first, the security objects that are affected by the security policy have to be identified. This step should be straight forward since an organization typically knows what assets it owns, creates and manages.

Identify Policy Functions

Once the security objects are known, one has to define the functions of the policy that can be applied to these security objects. Typical functions are related to the creation, management and destruction of the security objects. The functions of a policy are also often related to state changes in a security object. It is therefore beneficial to think about the lifecycle of the security object already at this point in time.

Construct an Object/Function Matrix

The object/function matrix represents which functions of the policy can be applied to a security object. A match in the matrix indicates that the function is applicable to the object.

Identify Key Management Processes

Using the guidelines of the original method [DLRMR13], the key management processes of a security policy are extracted from the object/function matrix. At this point, the policy landscape is completed and contains a list of processes and security objects. Additionally the object/function matrix shows, which security objects are affected by a process.

Figure 4.17 shows how a policy landscape is derived based on the security objects and the policy functions. Each cross in the matrix indicates that a policy function can be applied to the security object. Identifying the key management processes is performed based on the matrix. Process identification is not deterministic and a trade-off between two extremes, one which uses a single process for the complete policy and one which uses a dedicated process for every cross in the matrix.

		<i>Security Object Type</i>		
		Key	Certificate	HSM
<i>Policy Function</i>	Configure			① X
	Decommission	② X		X
	Create Object	③ X	X	
	Archive Object	X	X	
	Revoke Object		X	
	...			

¹ Process: Setup and Configure HSM
² Process: Remove Key from HSM
³ Process: Key and Certificate Management

Figure 4.17: Object/Function Matrix Evolving into a Policy Landscape

Prioritize Processes and Security Objects: Not every process that has been identified is worthwhile to be modeled in full detail or even implemented in the Policy Support System. Processes are therefore often prioritized based on different criteria, such as their contribution to strategic goals, error rates or their suitability for automation. Revenue generated by a process would be a typical criteria to prioritize a business process. Most processes in the domain of security of however generate no revenue [SS08]. The strategic goal of a policy is to reduce the risk of a security breach. One should therefore prioritize processes with high risks and where errors have a fatal impact.

Security Object and Key Management Process Design

Design of the process and security object models starts, once the security objects and key management process have been identified and selected for detailed modeling. Figure 4.18 shows the iterative design process that is applied by the Integrated Modeling Framework for Operational Security Policies. The design process consist of the main activities, the design of the security object models and the design of the corresponding key management processes.

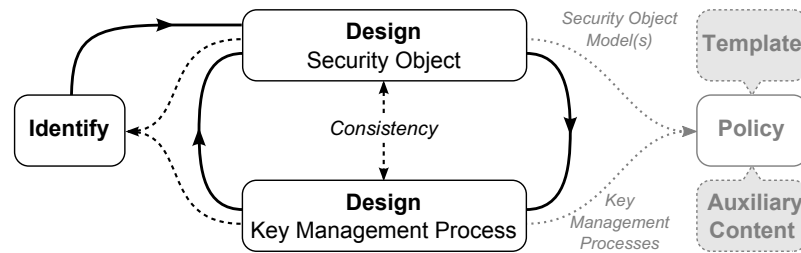


Figure 4.18: Iterative Design Process of Security Objects and Key Management Processes

One should start with the design of the SOML security object model, since it represents the policy from a higher abstraction level. The security object model also has no dependency on the process models. Creating the security object model is done using techniques known from information modeling [KR94] and object-oriented design [RBP⁺91]. Existing standards [BBB⁺12] and best-practice may help to identify attributes and behavior of the security objects. The detailed modeling of the relations and lifecycles can result in the identification of new objects and processes.

Key management processes are designed with BPMN and use the SOML data modeling elements to specify the interaction with security objects. Not surprisingly, the same techniques known from BPM [DLRMR13] can be applied to the design of key management processes. Beyond the normal process design process, one must ensure that security objects and key management processes are consistent and complete. Resolving these inconsistencies can result in changes to existing security objects and the identification of new objects and processes. Consistency and completeness implies that the models fulfill the following definition:

Definition (Consistency and Completeness)

Static consistency can be decided at design time and guarantees that the models are valid. A set of security object and key management process models satisfy *static consistency* if the models fulfill the following conditions:

- Processes are valid BPMN models which use the SOML data modeling constructs.
- Security object models are valid UML models which apply the SOML profile.
- Processes only use security objects that are defined in the security object models.
- Processes only access attributes of security objects that are defined in the security object models. The data types of the attributes used in the two models match.
- Processes only request state transitions that are defined in the lifecycle model of the corresponding security object.

Dynamic consistency can only be evaluated at runtime and requires that:

- State change requests must only occur in an order that is expected by the lifecycle of the security object. No uncaught state change error must leave the process instance.

The main purpose of the *completeness* criteria is to find unused objects, states and attributes. As such, the criteria are not mandatory and only serve as guideline to evaluate whether processes consider all aspects of the security objects. A model is complete if:

- Each security object type is created by at least one process.
- Each lifecycle state of a security object can be reached. The condition implies that each state transition can be triggered by at least one process.
- Each attribute of the security objects is accessed by at least one process.

Security object and key management process models are iteratively refined until the desired state has been achieved. The models are then included into the security policy, which is often created

based on a predefined document template. The policy document is enriched with auxiliary content that describes the processes and covers any topic that is not directly related to the security objects and processes. In particular, detailed instructions for the activities of the key management processes are added to the policy. These instructions are used later when a process is implemented in the Policy Support System.

4.2.2 Enacting the Security Policy

A security policy that has been modeled using the Security Object Modeling Language can be enacted by the Policy Support System. Before doing so, the models created in the design phase have to be transformed into an executable artifact and packaged in a deployment unit. The Policy Support System assists the user during the execution of key management processes and manages the lifecycle of security objects, once the SOML process and security object definitions have been deployed. Section 4.2.2 describes how the models of a security policy are implemented, deployed and executed by the Policy Support System.

Model Transformation into an Executable Artifact

Key management process and security object models created during the design phase must be transformed into an executable form before they can be enacted by the Policy Support System. Design models typically do not contain sufficient details to allow automated execution. They may for example lack the formal routing conditions on gateways. A developer therefore has to complete the models that have been created by the business analyst and defined implementation specific execution properties. The following four steps are required to transform the SOML design models:

Transform Security Object Models

A valid security object design model is already executable and does not have to be transformed any further. Only entry and exit actions defined using natural language must be implemented in a programming language that is supported by the Policy Support System.

Transform Key Management Processes

The BPMN design model must be transformed into an executable process model. The goals of this transformation is to identify process parts that are suited for automation and refine the process models to a level where it can be enacted by an engine. Typical refinements which are added in this step are error handling and formalized conditional expressions. These elements are often omitted in the design models to improve readability, but are required to automate the process in an engine. The same techniques as used for business processes are applied to transform the processes of a policy. Detailed instructions to transform processes into an executable model have been suggested by Dumas et al. [DLRMR13].

Transform SOML Data Modeling Elements

The data modeling elements used in the design models cannot be directly executed by the Policy Support System. The following steps are required to transform the data modeling elements presented in Section 4.1.2 into an executable format:

- BPMN data-flow and `DataCollections` are not supported by the process engine. Data-flow must be converted either into local task or process variables. Activities working with collections of data must use multi-instance activities [Rad12] or custom code.
- Conditions or queries written in natural languages must be formalized.
- `DataCollections` and `DataStores` must be transformed into BPMN `Annotations`.¹

¹The data element is replaced by an `Annotation` that contains the text (SOML command) of the data element. This transformation is required since the Activiti engine and designer do not yet support the BPMN data modeling elements. The transformation into annotations allows us to use Activiti while keeping the implementation effort reasonable. The transformation does not affect the semantics of the models at all.

Import Task Instructions

User activities of a key management process do not yet include any details that instruct the user how the activity has to be completed. This information has been added to the policy documents and is now imported back into the process models. The instructions of each task are collated from the relevant policy paragraphs and added to the user activity.

Deployment Unit

Security object and process models must be packed in a compatible deployment unit. Both process and security object definition can be deployed independently or together in a Business Archive (BAR) file. The following resources can be deployed to the Policy Support System:

- Independent key management process definitions stored in a file ending with `*.bpmn20.xml` or `*.bpmn20`. The definitions must be stored in the BPMN XMI format and may include visual information in the form of a BPMN Diagram Interchange (DI) section. The process engine extracts the DI section from the models and visualizes the processes.
- Independent security object definitions stored in a single security object model file ending with `*.sowl.uml` or `*.sowl`. The definitions must be stored in the UML XMI format.
- A static `png`, `jpg`, `gif` or `svg` image of a process definition. The image must be located in the same folder as the process definition and share the name of the process key and filename. (E.g. A process definition file `SubFolder/Def.bpmn20` containing a process with key `abc` will search for images named `SubFolder/Def.abc.*` or `SubFolder/Def.*`) The image can be displayed alternatively to the image generated from the DI information.
- A static `png` image of a security object lifecycle definition. The image must be located in the same folder as the security definition and share the name of the security object definition. (E.g. A security object model file `SubFolder/Def.sowl` containing a security object with key `abc` will search for images named `SubFolder/Def.abc.*`.)
- A BAR file ending with `*.bar` or `*.zip`. A BAR file contains a compressed folder structure that includes any number of resources, including images, process and security object models.

Process Assistance and Security Object Lifecycle Management

The Policy Support System enacts SOWL security policy models based on the same principles as a normal Business Process Management System (BPMS). Process and security object models are loaded into the system, where they are executed by an execution engine. This engine parses the models and enacts them according to the BPMN and SOWL language specifications. Figure 4.19 shows the components of the Policy Support System compared to a generic BPMS.

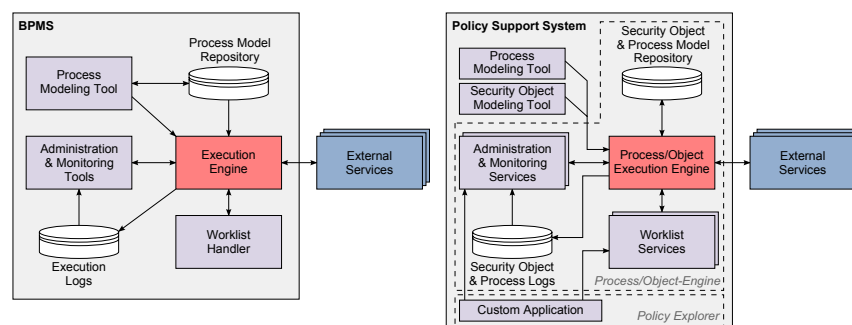


Figure 4.19: Generic BPMS Architecture [DLRMR13] Mapped to the Policy Support System

The difference* between a standard BPMS and the Policy Support System lies in the support for security objects. The process/object engine is not only capable of executing BPMN processes, but also supports SOML security object models. In addition to processes, the execution engine now also creates instances of security objects, executes their lifecycle and stores their data. The engine can also send notifications or execute actions based on state and data changes of security object instances. These functionalities together comprise the lifecycle management of security objects for which the Policy Support System is responsible. The detailed use cases and functionality of the system is presented in Chapter 5.

*The prototype implementation of the Policy Support System has several minor differences compared to a generic BPMS. These architectural changes are a result of the decision, to build the prototype based on the Activiti platform. The Activiti execution engine can be embedded into other applications. Services interfaces are provided by the engine instead of concrete worklist handlers, administration- and monitoring tools. Custom client applications use these interfaces to interact with the engine.

4.2.3 Monitoring and Controlling the Security Policy Execution

The Policy Support System collects security object and process execution logs during the execution of the models. The engine can store the following events and forward them to other applications:

- Creation and deletion of security object and process instances.
- Activation and completion of (user-) tasks.
- Changes to process variables and attributes of security objects.
- State changes in the lifecycle of security object.

Other application may read/receive these events and perform analysis on the data. The events include detailed information about the event occurrence and the user which caused it. Clients can use the data to perform real-time Business Activity Monitoring or use the historic data for Business Intelligence [GRC04]. Both methods can be applied to the following scenarios:

- The data can be used to audit the compliance with the policy both in real-time and ex post. The data may be used to reveal fraudulent patterns or convince an auditor, that a process has been correctly executed.
- Process execution data can be used to identify processes and tasks which are prone to errors or cause high costs. Based on this knowledge, processes of a policy can be improved.
- Knowledge about existing security object instances and prior task/process execution time can be used to forecasts further work performed by the users. Figure 4.20 shows an exemplary use case of the collected data. The history of keys and the duration of the process could be used to predict the number of key generations and the resulting effort for the year 2014.

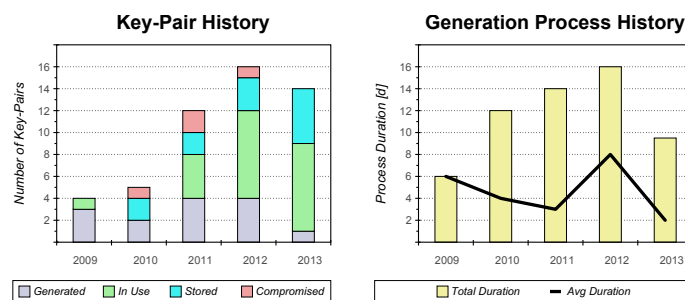


Figure 4.20: Example: Security Object and Process History Data

Implementation of the Integrated Modeling Framework

In Chapter 5, we present the prototype implementation of the Policy Support System. This system enacts the SOML models and performs the lifecycle management of security objects.

The Policy Support System is built on top of the Activiti BPM Platform. In Section 5.1, we give a short introduction of Activiti before presenting the use cases (Section 5.2) and architecture (Section 5.3) of the Policy Support System. In Section 5.4, we introduce the Policy Explorer which allows users to execute a security policy.

5.1 The Activiti BPM Platform

The Activiti BPM Platform is an open source BPM platform and includes a process engine. This engine provide native support for the execution of BPMN process models. The engine is built in Java and is designed to be lightweight, fast and extensible.

Users access the process engine either through the Activiti Explorer or a custom client application. The Activiti Explorer allows process participants to execute processes and provides an interface to manage the engine. The Explorer is a web-application which is built based on the Vaadin [G⁺11] framework. Alternatively, a custom interface can be built around the process engine. Such an application either accesses the engine trough the built-in Java interfaces or the REST interface.

The platform contains three process modeling tools which are targeted to different audiences. *Activiti Kickstart* provides a low-barrier entry to process modeling. The *Designer* and *Modeler* are full BPMN modeling tools. Figure 5.1 shows an overview of the core Activiti components.

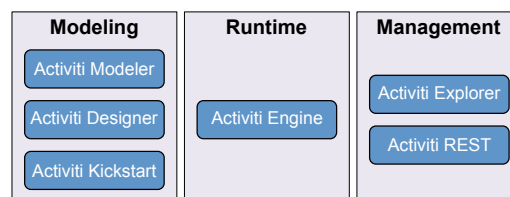


Figure 5.1: Overview of the Activiti BPM Platform [Act]

5.2 Use Cases of the Policy Support System

The Policy Support System consists of the Process/Object Engine and the Policy Explorer. These sub-systems support two different application scenarios. The Process/Object Engine can either be used independently or in conjunction with the Policy Explorer.

Executable SOML artifacts can be deployed in the Process/Object Engine. The engine enacts the models and manages the lifecycle of the security object instances. Clients, such as the Policy Explorer, use the public services to manage and control the Process/Object Engine. In the first application scenario, the engine is embedded in, and accessed by a 3rd party application. In this scenario, the use cases cover the usage of the services exposed by the engine.

The second application scenario covers the usage of the Policy Support System to enact a security policy. End-users access the Policy Explorer to deploy, execute and monitor a security policy. The Policy Explorer internally uses the Process/Object Engine to enact the models. The separation into two sub-systems improves the reusability and flexibility of the engine. In addition, the engine can be embedded into a unit-test environment to support automated testing of the models.

Figure 5.2 shows the uses cases of the Process/Object Engine and Policy Explorer. The use case diagram includes existing functionality that is already built into the Activiti Engine and Explorer.

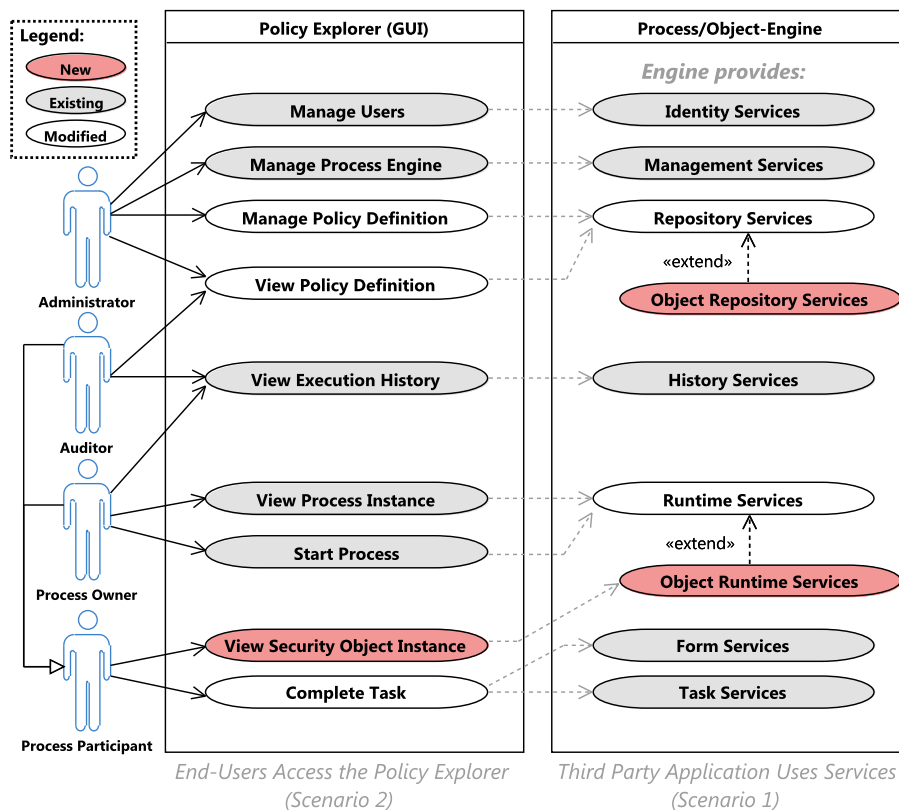


Figure 5.2: Use Case Diagram of the Policy Support System

The Process/Object Engine extends the standard Activiti (process) Engine. The following extensions

are added to support SOML models and security objects:

Store Security Object Definitions in the Repository

The repository of the Activiti Engine is responsible to load and store BPMN process models. The Process/Object Engine extends the repository and includes support for security objects. SOML security object models can be deployed to the repository. The engine parses the models and transforms them into security object definitions, which can be enacted. In addition, the object repository services can be used to retrieve and manage security object definitions.

Execute Security Object Instances at Runtime

The Activiti Engine manages the execution of process instances. It supports the creation of new process instances. The complete execution of a process instance is controlled through the runtime services. The Process/Object Engine extends the core Engine and integrates the execution of security object and process instances. Security object instances are stored and managed by the extension which ensures that their data is valid. The Process/Object Engine executes the SOML models and enacts the lifecycle of the objects. The object runtime services form the public API, which is used to interact with the security object instances. These services are used by clients to retrieve and change security object instances.

The Policy Explorer supports the end-users during the execution of a security policy. The Policy Explorer is built based on the existing Activiti Explorer and retains its capabilities. The following uses-cases are new or significantly change the behavior of the Activiti Explorer:

Manage and View Policy Definitions

A policy definition must be deployed to the Process/Object Engine before it can be enacted. An administrator can use the GUI to upload SOML deployment artifacts. These artifacts include process and security object models. The Policy Explorer can also be used to explore existing policy definitions which have already been uploaded.

View Security Object Instances

The Policy Explorer allows users to view existing security object instances. Just like process instances, a list of existing security object instances is presented to the user. Upon selecting an object instance, its detailed information is displayed to the user. The details contain the current attribute values and the state of the security object instance.

Start Processes and Complete Tasks

The Policy Explorer is used to start and execute processes. Internally, processes are executed by the Process/Object Engine which also enacts the security objects. In terms of the process execution, as seen by user, the management of the security objects is transparent. A policy process is executed in the same way as a normal process. In the background the engine however executes the SOML models and manages security objects. Users interact with the process execution by completing tasks. The Policy Explorer contains a task interface that shows the instructions of a task and allows users to complete them.

5.3 Architecture of the Policy Support System

The prototype implementation is built based on the Activiti (process) Engine and Explorer. The **Activiti-Engine** is the core component of the Activiti BPM Platform. The engine is responsible to enact BPMN process models. Different client components use the public API of the **Activiti-Engine**. These applications are used by users and other applications to interact with the engine. The Activiti Platform includes the **Activiti-Explorer**, a standard GUI to manage the process engine and execute processes.

The Policy Support System introduces two new components. The **Activiti-Object-Engine** extends the existing **Activiti-Engine** with the features that are required to enact SOML models.

The **Activiti-Policy-Explorer** is a custom web-application based on the **Activiti-Explorer**. Figure 5.3 shows an overview of the components of the Policy Support System.

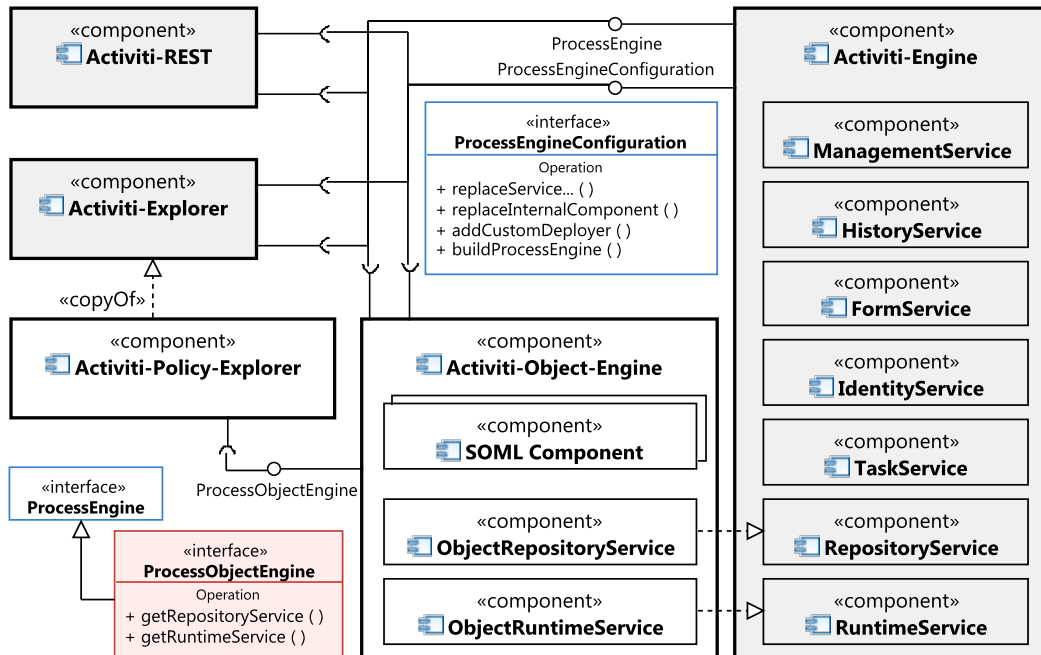


Figure 5.3: Architecture Overview (Standard Activiti Components in Gray)

The standard **Activiti-Engine** exposes the **ProcessEngineConfiguration** and **ProcessEngine** interfaces. A process engine is created and configured using the **ProcessEngineConfiguration** interface. The **ProcessEngine** provides access to the different services of the engine. These services are used by clients, such as the **Activiti-Explorer** to control the engine. The **Activiti** engine supports extension through the **ProcessEngineConfiguration** interface. During the configuration of an engine, a client can inject new behavior or replace internal components of the **Activiti-Engine**. Listing 5.1 shows how the configuration is used to extend and customize the behavior of the engine.

```
public ProcessEngine buildCustomizedEngine() {
    // Create a default process engine configuration
    ProcessEngineConfiguration cfg = new StandaloneProcessEngineConfiguration();

    // Add a custom deployer for SOML Models and replace the repository services
    cfg.setCustomPostDeployers(Arrays.asList(new SomlDeployer()));
    cfg.setRepositoryService(new ObjectRepositoryService());

    // Build a process engine using the injected custom extensions and components
    return cfg.buildProcessEngine();
}
```

Listing 5.1: Example: Extension Mechanism of the Activiti Process Engine

Our extension uses the `ProcessEngineConfiguration` interface to inject support for SOML models into the standard Activiti Engine. The `Activiti-Object-Engine` provides a new implementation of the `ProcessEngineConfiguration` which is used to create a Process/Object Engine. The `Activiti-Object-Engine` configures and creates an adapted instance of the `Activiti-Engine`:

- The configuration injects several new components into the `Activiti-Engine`. These components either add features or change the behavior of internal components. The components added by the `Activiti-Object-Engine` introduce support for SOML models and lifecycle management of security objects. An overview of the components is given in Section 5.3.1.
- The Process/Object Engine exposes an extension of the `ProcessEngine` interface. This new interface provides access to the `ObjectRepository`- and `ObjectRuntimeServices`. These services extend the default services and add support for security objects. Clients use this public API to deploy security object models and interact with security object instances. Section 5.3.2 presents the changes to the public API of the engine.

The `Activiti-Policy-Explorer` uses the new interfaces and services provided by our extension. The Policy Explorer is built based on the code base of the `Activiti-Explorer`. We have decided to copy the code base since the standard explorer is not designed to be extensible. The Policy Explorer remains a Vaadin web-application and retains the architecture of the Activiti Explorer.

5.3.1 Sub-Components of the Policy Support System

The `Activiti-Object-Engine` injects several components into the `Activiti-Engine` to extend or replace the default behavior. In this Section, we give an overview of the features that are added by these components. We briefly discuss important design decision of the components.

Loading Security Object Definitions

The standard engine is extended with the capabilities that are required to load security object models. A security object model is loaded and transformed into security object definitions. Once parsed, these definitions are stored in the `ACT_RE_SODEF` table of the activity database (See Section B.2). From there, the definitions are retrieved and concrete object instances are instantiated.

Security object models are transformed into definitions the same way a BPMN process model is converted into a process definition. Instead of using a custom parser to read the UML models, our implementation relies on the Eclipse MDT/UML2 parsing library.

Enacting Security Object Instances

Concrete instances of a security object definition can be created at runtime. Security object instances, their attributes and state are stored in the `ACT_RU_SO` and `ACT_RU_ATTRIBUTE` tables of the activity database (See Section B.2). From there, the security object instances can be retrieved. Any changes to the attributes or state of the instance is persisted into the database.

Security object instances are managed by the engine using the same principles as a process instance. A security object instance, in fact, shares the same execution interface as the process instance. The lifecycle of security objects can therefore be executed using the Activiti Process Virtual Machine (PVM) [BVF07] graph execution language that is also used to execute processes.

Using Security Objects in a Process

The `Activiti-Object-Engine` performs several changes to the standard process execution capabilities of the Activiti engine. These features are required to support and seamlessly integrate security objects during the execution of processes. The following features are added by our extension:

- A new Activiti process variable type which allows processes to work with security objects. A security object instance can thus be used by a process. Usage of normal process variables and security object instances is transparent for the process developer. Security object instances can even be used in custom script code that is included in the process definition.
- An extension to the standard BPMN parser which inspects the annotations attached to a BPMN activity. Using these annotations, a designer captures the intent to interact with security object instances. The content of the annotation is parsed according to the SOML language specification. Once parsed, these commands are added to the exit action of the BPMN activity. The attached SOML commands are executed when activities are completed. The commands then change or create security object instances.
- Conditional start and intermediate events which are currently not supported by the Activiti process engine. Using these events a designer captures conditional events stated on security objects. The conditional events are triggered as soon as the condition evaluates to true.

The SOML language specification defines a formal language to specify the interaction with security objects. We use the ANTLR parser generator [PQ95] to interpret this language. The source code that is required to parse the SOML commands can thus be automatically generated based on the language specification shown in Section B.1.

5.3.2 Process Engine API

The components presented in Section 5.3.1 are internally used by the **Activiti-Object-Engine**. Clients accessing the engine only use its public API. Figure 5.4 shows the extension of the existing Activiti service API which is introduced by the Policy Support System.

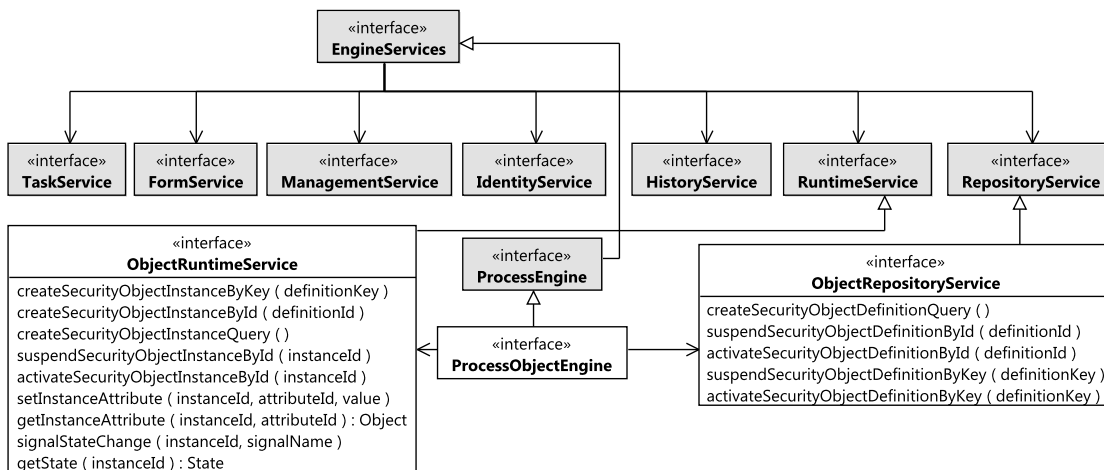


Figure 5.4: Service Extension Introduced by the Policy Support System

By using these services a client, such as the Policy Explorer, controls and interact with the engine. Non-Java client can use the Activiti REST interface to connect to the engine. Our extension does however not yet support the REST interface.

Listing 5.2 shows an example where a Process/Object Engine is created and used by a unit test. The Process/Object Engine is embedded into the application. This application scenario is typically used to write automated test for the process and security object models.


```

/**
 * Testcase for the create key process of the policy.
 *
 * The testcase deploys a policy (Policy.bar), creates a security object instances and executes a process.
 */
@Test public void testPolicyCreateKeyProcess() {
    // Create a process/object engine.
    StandaloneProcessObjectEngineConfiguration cfg = new StandaloneProcessObjectEngineConfiguration();
    ProcessObjectEngine engine = cfg.buildProcessEngine();

    // Deploy SOML security policy models. (Contained in a JAR file.)
    engine.getRepositoryService().createDeployment().addClasspathResource("policy.bar").deploy();

    // Check if 2 process and 5 security object definitions have been deployed.
    assertEquals(2, engine.getRepositoryService().createProcessDefinitionQuery().count());
    assertEquals(5, engine.getRepositoryService().createSecurityObjectDefinitionQuery().count());

    // Use runtime services to create a HSM security object instance.
    // (During normal operation this instance would have been created by another process.)
    engine.getRuntimeService().createSecurityObjectInstanceByKey("policy::HSM");

    // Start the create key process and execute it until its blocked (E.g. Waiting for user input).
    engine.getRuntimeService().startProcessInstanceByKey("createKey");

    // Use object runtime services to validate the result of the process step.
    final SecurityObjectInstanceQuery query = engine.getRuntimeService().createSecurityObjectInstanceQuery();
    final SecurityObjectInstance key = query.securityObjectDefinitionKey("policy::Key").singleResult();
    assertNotNull("Process must create a key security object instance", key);
    final String name = (String) engine.getRuntimeService().getInstanceAttribute(key.getId(), "KeyName");
    assertEquals("Process must set attribute name to XYZ", "XYZ", name);
    ...
}

```

Listing 5.2: Example: Process/Object Engine Embedded in a JUnit Test

5.4 The Policy Explorer

The Policy Explorer is a web-application which allows users to interact with the Process/Object Engine. An administrator uses it to deploy security policies. Users which have to execute the policy then start these processes and complete their tasks with support of the system.

The Policy Explorer is based on the Activiti Explorer and supports all capabilities of this standard GUI. Our extension changes existing usage scenarios and introduces new features as shown in Figure 5.2. For the remainder of this Section, we will focus only on the significant changes, which are added by the Policy Explorer. The features supported by the standard Explorer can be found in the Activiti user manual [Act].

Compared to the Activiti Explorer, our application is aware of security objects. The Policy Explorer can not only be used to view process definitions but also supports security object definitions. These two definition types together form a policy definition which can managed trough the interface. Figure 5.5 shows a screen-shot of a process and security object definition.

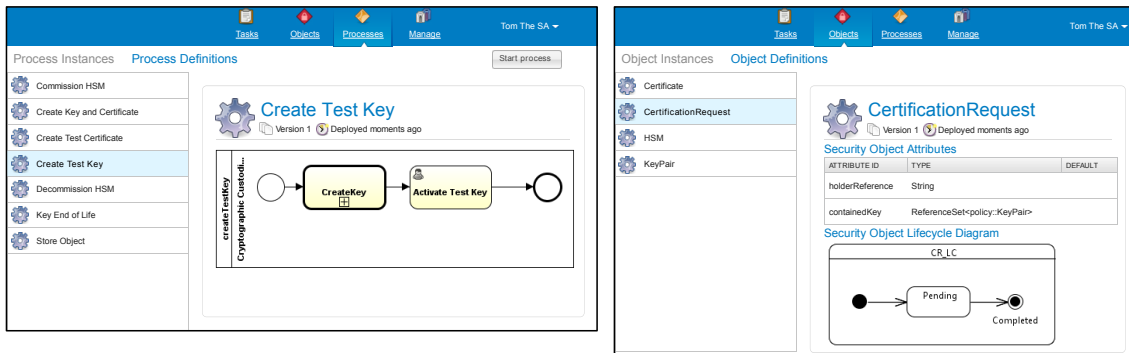


Figure 5.5: View & Manage Security Policy Definitions

The Policy Explorer can be used to view security object instances once they have been created. Similar to process instances, a user can access the explorer to view the attributes and the current state of security object instances. This interface allows users to get a quick overview of the security object instances which are managed by the system. Figure 5.6 shows a screen-shot of a security object instance, its attributes and current state.

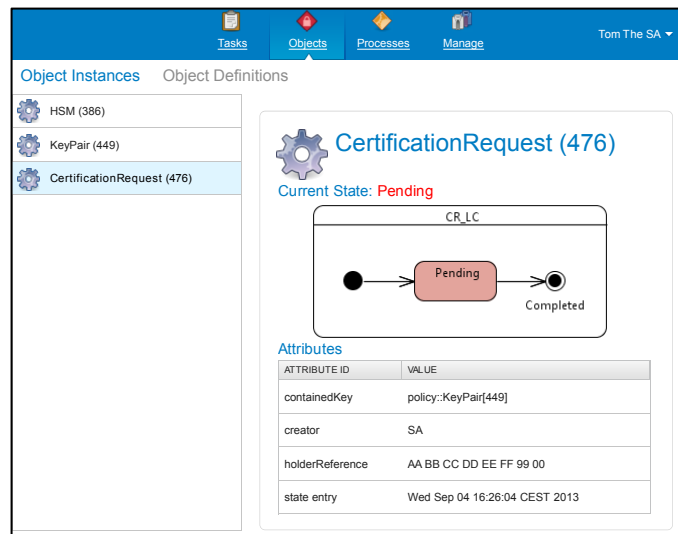


Figure 5.6: View Security Object Instances

The Activiti Explorer only provides very limited extension capabilities. Users can only customize the graphical theme and add new form types. In fact, the Activiti Explorer is not designed for use in a production environment. Developers are expected to create their own GUIs on top of the process engine. The Policy Explorer prototype is therefore built based on a copy of the Activiti code-base. The architecture of the Policy Explorer remains the same as for the standard Explorer. The web-application internally creates a Process/Object Engine and uses the API presented in Section 5.3.2 to control it.

Validation of the Integrated Modeling Framework

In Chapter 6, we present our validation of the Integrated Modeling Framework for Operational Security Policies. The validation is based on the information system introduced in Section 6.2. A traditional security policy, written in natural language, regulates how this system is operated. In Sections 6.3 and 6.4, we illustrate how these processes are designed and enacted using our framework. Finally, we discuss the results of the validation in Section 6.5.

6.1 Method of Validation

To validate our approach, we model and enact parts of the security policies used in a case study. We focus on the processes which regulate the creation of a member state key-pair and certificate. These processes are complex and crucial for the security of the system. The security policy, which we intend to model, is defined by a member state of the tachograph system. This policy has to comply with the European root CA policy [BN09] and practice statement [ERC04]. The models used for the validation are subject to the European Root Certification Authority (ERCA) security policies. We compare our approach to an existing, but not publicly available, member state security policy. This traditional policy is written in natural language and executed manually.

During the first phase of the validation (Section 6.3), we modeled the processes which are required to create an Member State Certification Authority (MSCA) key-pair and certificate. The key generation processes are among the most complex of the whole policy and comprise four security objects and eight sub-processes (Section ?? shows an extract of our modeling project). This first part of the evaluation will be used to answer the following question:

Research Question 1

The Integrated Modeling Framework for Operational Security Policies can be applied to model the security objects and operational processes of a security policy. The SOML language can express complex security policies. Yet, it remains simple enough to be useful for the stakeholders of a security policy (Experts, Auditors and Users).

These policy models then are enacted in a test environment during the second phase of the validation (Section 6.4). Long-term results of the framework applied in practice have not yet been collected. A semi-structured interview with three IBM security experts is used to qualitatively validate whether

our framework can address the problems identified in Section 1.1. The validation answers the following question regarding the execution of a policy specified based on our language:

Research Question 2

The use of SOML security object and process models improves the quality of the security policy. Formal specification removes ambiguities and increases the comprehensibility of the documents. The models also help to foster discussion during the design and implementation of a policy.

Research Question 3

The Policy Support System assists users during the execution of the security policy. It provides task instructions in a convenient way and ultimately reduces the risk of human errors. The historic data which is collected during the execution is appropriate to audit compliance with the policy.

Research Question 4

Security object lifecycle management, which is automated by the Policy Support System, is an important task. In automating it, the system reduces the risk of mistakes related to lifecycle management. (E.g.: Forgetting to renew a certificate before it expires.)

6.2 Case Study: The Tachograph System

The digital tachograph is a control device for road transport used across Europe. The digital tachograph is composed of a Vehicle Unit (VU) which is installed in the lorry and connected to the gearbox via a secured motion sensor. The vehicle unit contains a mass memory where data on drivers and their periods of driving and duty is held for about 12 months. Smart cards are used to access the vehicle unit and the data residing in it. Participating states of the system issue four different kind of smart cards:

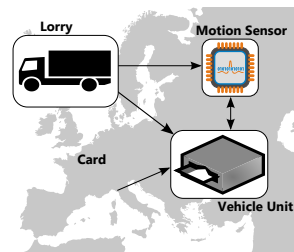


Figure 6.1: Tachograph System

- Driver Card** Drivers own a personal card which is used to identify them to the vehicle unit.
- Company Card** Company cards allow owners of a lorry to download data from the vehicle unit.
- Workshop Card** The vehicle unit and the sensor must be periodically calibrated. Workshop cards are available to authorized workshops which perform the calibration.
- Control Card** Control cards are used by enforcement officers to access data during road controls.

The tachograph system is used to monitor the activity of the drivers, lorries and companies. The core duty of the system is to record the working hours of the driver and his co-driver. The vehicle unit stores all driving, working, availability and rest periods of the drivers. Drivers are legally required to record their activities and provide this data during road controls. The vehicle unit also stores traveling distance and speed of the lorry as well as exceptional events. Examples for these exceptional events are overspending or attempts to manipulate the system.

Both, the driver and company may have an incentive to tamper the system in order to conceal any illegal behavior. The system is therefore protected by the application of symmetric and asymmetric cryptography. In particular the following two sub-systems are protected:

- Motion Sensor** A symmetric key is used to secure the communication between the sensor and the vehicle unit. Official manufacturers of motion sensors receive a key which they embed in the sensor. Trusted workshops then use their card to load the same key onto the vehicle unit and calibrate the sensor. Encryption prevents the end users from installing a forged motion sensor that reports a faulty (lower) speed.

Card Validity Asymmetric cryptography is used to ensure that all cards in use are valid. A Public–Key Infrastructure (PKI) is used to ensure that a card has been issued by an authority. Only cards with a certificate signed by a trusted CA are considered valid.

The tachograph system is used across Europe and managed by the European Union. Each participating member state is responsible to issue cards to its citizens and organizations. The EU manages the system specification and operates the European Root Certificate Authority, which acts as root of trust for the complete system.

A PKI is used since the system spans over multiple states across Europe. Each participating state operates a MSCA. All cards issued by the member state contain a certificate which is signed by the MSCA. The ERCA certificate is publicly available. The validity of a card is verified whenever a card is inserted into the vehicle unit. A card is valid only if the chain of trust between the card certificate and the ERCA certificate is complete.

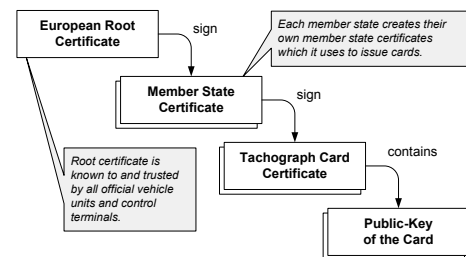


Figure 6.2: Tachograph Certificates

Tachograph System Security Policy

Every participating state of the system is responsible to operate a Member State Authority (MSA). The MSA has following responsibilities regarding the security of the tachograph system:

- The member state operates the MSCA which generates the MSCA certificates. These certificates are used by the state to create the smart cards. Only authorized member states that comply with the certification policy of the ERCA are eligible to issue cards. These states can submit their MSCA certificate to the ERCA which signs it using the root certificate. A certificate rollover is required regularly since MSCA certificates are only valid for two years.
- The MSCA requests the motion sensor key from the ERCA and stores it locally. The motion sensor key is included in workshop cards and distributed to the motion sensor manufactures.
- The member state handles card applications of drivers, companies, workshops and control authorities. Card issuance procedures start with the examination and validation of a card application. The state issues a digital identity document if an application is valid. Therefore it has to physically (visually) personalize the card and program the integrated circuit that resides on the smart card.

Security policies are used to regulate the operational procedures applied by the member states. These policies are currently written using natural language and describe the processes and standard operating procedures that are required to run the tachograph system. The policy documents contain key management processes that are crucial for the secure operation of the tachograph system.

6.3 Designing the Tachograph Security Policy

Elicit Security Objects: As advocated by our methodology, the design of a security policy starts with the identification of the security objects. Analysis of the policy requirements [BN09] and the existing member state policy quickly yields a variety of candidate objects which can be modeled. The identification of these objects is straight–forward since the ERCA policy [BN09] already lists security relevant assets. Other security objects are found by analyzing the possible relations of known objects. Table 6.1 lists some of the security object candidates of the tachograph policy.

<i>Security Object</i>	<i>Description</i>
Security Module (HSM)	The infrastructure where keys are generated, managed and stored.
MSCA Key-Pair	The asymmetric key pair of the MSCA.
MSCA Certificate	The certificate of the MSCA. It is signed by the ERCA and regularly renewed.
Certification Request	A request from the MSCA to the ERCA to sign a certificate.

Table 6.1: Security Object Candidates of the Tachograph Policy (Excerpt)

Build a Policy Landscape: Once the security objects have been identified, construction of a policy landscape can begin. Therefore we have to identify the policy functions which can be applied to the security objects. The purpose of these policy functions is to manage the security objects. As such they often create, delete or change security objects. The lifecycle of the security object is therefore a good starting point to identify the functions.

The object/function matrix is then used to map policy function to the security object to which the function can be applied. Based on this matrix we continue with the identification of the processes that are required by the policy. Process identification is a trade off between process size and the number of processes. In the two extremes a single process covers the whole policy or a dedicated process is used for each combination of function and object. Figure 6.3 shows the policy landscape of the tachograph security policy. We decided to model the key and certificate management parts of the policy using eight processes.

		<i>Security Object Type</i>						
		HSM	MSCA Key.	MSCA Cert.	Cert. Request	Pers. Site	Card	...
<i>Policy Function</i>	Configure	① X				X		
	Decommission	② X				X		
	Create Object		③ X	X	X			X
	Rollover		X	X	X			
	Archive Object		④ X	X				
	Revoke Object			⑤ X				
	Create Test Object		⑥ X	⑦ X				X
	End of Life		⑧ X					X
...								

^{1, 2} Commission & Decommission HSM

³ Create Key and Certificate

Figure 6.3: Tachograph Policy Landscape (Excerpt)

Design Security Objects and Processes: Modeling of the selected processes and security object is performed iteratively until the process and security object models are completed. The modeling activity follows the same principles as for normal business processes and information model. The focus of a typical business process is often not on the data and process models rarely use the data modeling elements. Our models however define detailed interaction with the security objects.

The result of the design phase are abstract models of the processes and security object which can be included in the policy documents. These models do not yet contain all details that are required for the execution. An abstract model may still make use of informal notations to define SOML queries and conditional events. Figure 6.4 shows a process that is used to create a key and certificate. The process model uses the BPMN data modeling elements to specify the interaction with the key and certificate security object.

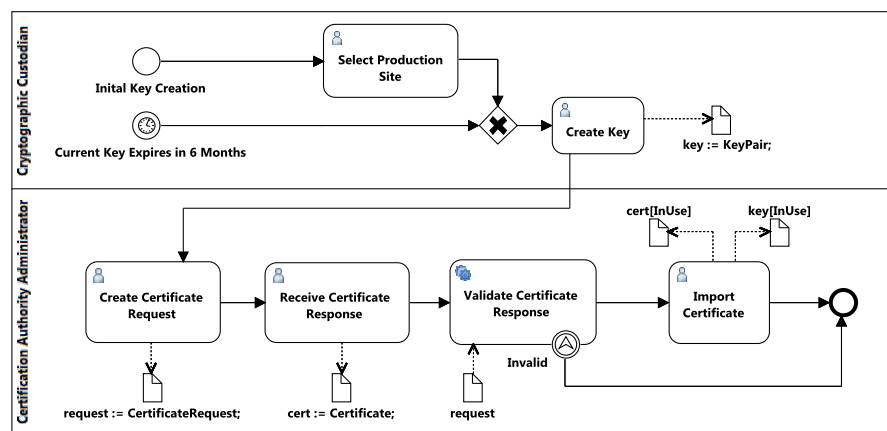


Figure 6.4: Create Key and Certificate Process Abstract Model

Implement the Processes: The process shown in Figure 6.4 makes use of an informal start event and contains not enough detail to execute it. The models were converted into an executable artifacts based on the instruction given in Section 4.2.2. During this conversion process, the models are enriched with implementation specific details and all SOML elements are formalized. Figure 6.5 shows how parts of the process introduced in Figure 6.4 are transformed into an executable form.

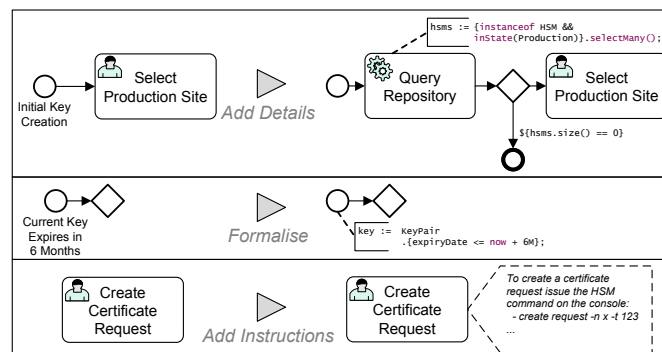


Figure 6.5: Transformation into an Executable Model

The final step is to collect the instructions of each task from the security policies and include them in the tasks description. These instructions are later displayed to the user and support him during the execution of a task. The quality of the support provided by the system finally relies on the accuracy of these instructions.

Task instructions of a traditional policy are often scattered over different paragraphs or documents. One reason for this practice is to avoid redundancies in the policy documents. To avoid redundancies in the process model, the designer has to make use of reusable components. BPMN allows the reuse of process parts by defining them as sub-processes or global tasks. These process fragments can be called from other processes and prevent the duplication of task instructions.

6.4 Enacting the Tachograph Security Policy

During the second phase of the evaluation, the tachograph security policy is enacted. The lifetime of key management processes used in practice normally spans over multiple years. Therefore we decided to **simulate** the execution of the policy in a controlled test environment.

A security policy based on the Integrated Modeling Framework for Operational Security Policies can either be executed manually or in the Policy Support System. **During manual execution, the models help users to understand the processes and security objects.** The models support the policy documents and clearly define the sequence in which tasks are to be executed.

The Policy Support System automatically provides assistance during the execution of the operational processes of the security policy. Therefore a user process logs into the Policy Explorer and starts a new process instance. Process participants then see a list of tasks, which are assigned to them. They select a task which they want to complete. The Policy Explorer presents detailed task instructions which have been included in the implementation models. These instructions provide all necessary information that is required to complete the task. A user therefore no more has to look up the instructions in the policy documents. The task may require additional input from the user which he can provide through a user form. Figure 6.6 shows an example of the instructions and forms used by the Policy Explorer.

Create High Value Key
 No due date Created 11 minutes ago Part of process: 'Create Key Sub Process'

Create a key pair
 High Value Key: Key generation must be executed on the HSM!
 The key generation process is as follows :

1. IS opens the data centre and unlocks the secure cabin.
2. CAA logs in to the SCS
3. CAA ensures that the production CAs are sorted into the HSM.
4. CAA triggers Key generation. Command : create key ... through the SCS user interface.

The algorithm for generating the key identifiers ensures that the Key Identifier (KID) and modulus (n) of keys ... the MSCA domain. The certificate holder reference has an 'additional info' field that is can be configured ... reference to be a self-managed counter.

Fill in the form below and complete the task:

Key Label	High Value Key
Key Identifier *	MSCA Key Sep 2013
Key Application *	Data Signing
Expiry Date *	13-09-2018

Complete task Reset form

Task Instructions
User Forms

Figure 6.6: Task Assistance by the Policy Support System

6.5 Discussion

In Section 6.5, we discuss the results of the two validation phases. The first phase covers the design of the policy, which was modeled by ourselves. In the second phase, we have presented the resulting models to security experts and interviewed them using the questionnaire shown in Section C.1.

Phase 1: Designing a Security Policy

Research Question 1

The Integrated Modeling Framework for Operational Security Policies can be applied to model the security objects and operational processes of a security policy. The SOML language can express complex security policies. Yet, it remains simple enough to be useful for the stakeholders of a security policy (Experts, Auditors and Users).

We have successfully modeled the processes that are used to create and manage the MSCA key-pair and certificate. The case study showed, that the SOML language is flexible and expressive enough to define the most complex processes of the tachograph security policy.

The execution semantics of the language match with the typical application patterns which we could observe during the modeling process. In particular, the state change semantics and concurrent access to object instances proved as a good design choice. The security object instances are used by multiple processes, but these processes are executed sequentially. The optimistic-offline lock, which we chose, provides sufficient locking, yet remains simple to use.

The BPMN language itself provides all the functionality that is required to model the processes of the security policy. Our extension allows us to tie these processes and the security objects together. The conditional events on security objects is important to react to changes in the objects. This kind of interaction is not supported by the existing approaches and has been a major reason to extend an existing language. Our own modeling effort showed, that the design experience could be improved. The SOML language is not always completely intuitive and the engine does not tolerate even small deviations from the SOML specification. Writing valid conditional events is, for example difficult. These issues degrade the overall modeling experience.

The interviewed security experts agree, that the design models can be understood by all involved stakeholders. The experts are confident that a specialist could be trained, to model a policy with our language. Transformation into an executable artifact should however be conducted by a developer, who has knowledge about process management and the execution engine.

Conclusion of Phase 1: Using the Integrated Modeling Framework for Operational Security Policies we could successfully model the most complex processes of the tachograph security policy. We are therefore confident, that our framework can be used to express a security policy. There is however improvement potential around the tooling and parts of the language itself.

Phase 2: Enacting a Security Policy

Research Question 2

The use of SOML security object and process models improves the quality of the security policy. Formal specification removes ambiguities and increases the comprehensibility of the documents. The models also help to foster discussion during the design and implementation of a policy.

All interviewees agreed, that the formal process models improve the quality of a security policy compared to relying on natural language only. The models removes ambiguities by defining a formal sequence of the tasks. Our task instructions however remain written in natural language, which involves the same problems as they exist for normal policies. The process models at least formalize and clarify some important parts of the policy.

The experts agree, that the most important stakeholders of a policy would be capable of reading and interpreting the graphical models. As such, the models can serve as an input for discussion among auditors, architects, users and security experts. An interviewee stressed the weakness of current policies, which do often not consider important lifecycle steps of security objects. Our lifecycle models can help to reveal uncovered transitions. Automatic validation of the models could further improve the framework. The utility of our model for a policy has been rated as follows:

Process Models	The models have been identified as very helpful by the interviewees. Our experts would include them in a security policy documentation. The process models could help users even during a manual execution of the policy.
Lifecycle Models	Our security experts emphasize the importance of the security object's lifecycle during the design of the policy. The models are therefore very important during the creation of the security policy.
Information Model	The interviewees identified the information model as the least important of all models. Some of the experts would not include it in the policy.

Research Question 3

The Policy Support System assists users during the execution of the security policy. It provides task instructions in a convenient way and ultimately reduces the risk of human errors. The historic data which is collected during the execution is appropriate to audit compliance with the policy.

The interviewees agree, that the Policy Support System can prevent common errors which are a result of users not adhering to the policy. Our experts also confirm, that most users would appreciate such a system. Since processes are only executed infrequently users would accept the strict governance and not feel restricted by the system. An interviewee also brought up that users themselves often worry about the mistakes they could make. Therefore they would gladly have a system assisting them.

To be used in practice, the usability of the Policy Support System should be improved. Especially, the input of data poses a treat and is cumbersome for the user. An idea brought up during the interview, was to use barcodes to replace manual data-input. In this setup, a secure system would print a receipt which the user later has to present to the Policy Support System.

The interviewed security experts confirm that the log information created by the Policy Support System could serve as input for an audit. They rate the historic data collected by our system as more accurate compared to the logs they ran across in practice. Further work would be required to improve the utility and trustworthiness of the log entires. The Policy Support System should either send log events to a secure logging system or use cryptography to protect log files from modification. An auditor would want to be sure, that users of the Policy Support System cannot modify the log information once it has been written.

Research Question 4

Security object lifecycle management, which is automated by the Policy Support System, is an important task. In automating it, the system reduces the risk of mistakes related to lifecycle management. (E.g.: Forgetting to renew a certificate before it expires.)

Our experts confirm that security object lifecycle management is a crucial task of a security policy application. They criticize the current practices, which are applied to track and manage security objects. Data of security objects is often not available or noted manually in spreadsheets. In some cases, the data is available on paper and stored in an archive which renders access to it both cumbersome and slow. Reactions to events are often managed by the employees on an individual basis. People use calendar entries or personal notes to remind them of their tasks. Problems therefore often occur when people change positions or are otherwise not available.

During the interview all participants acknowledge that the Policy Support System solves these problems. They believe that the system can prevent many errors which result due to the lack of a central lifecycle management solution.

Conclusion of Phase 2: The Integrated Modeling Framework for Operational Security Policies has a great potential to improve the way, we define and execute security policies. The Policy Support System is a promising solution to reduce the risk of human errors. Organization therefore have a clear incentive to operate such a support system which assists their employees. Different aspects of the Policy Support System prototype must be improved before using the system in practice. These limitations are only affect the implementation but do not concern our methodology and language.

6.5.1 Limitations of the Integrated Modeling Framework

The Integrated Modeling Framework for Operational Security Policies has limitations which reduce its possible potential. First of all, we decided not to implement all language features which we have identified in the requirements. The SOML modeling language is kept as simple as possible and does not provide syntactical sugar. Although we have managed to model the policy of the case study, we could identify some scenarios, such as conditional events, which were cumbersome to model.

The modeling language lacks tool support which renders it difficult to use. Constraints of the SOML language specification are not implemented by the UML profile. A user can thus create a valid UML which will result in a runtime error when parsed by the Process/Object Engine. The language also introduces a new Domain Specific Language to define the interaction between processes and objects. A designer has to learn this language before he can create a model. In addition, the standard UML and BPMN modeling tools do not support the designer in creating a SOML model. All these factors imply that a designer needs deep knowledge of the SOML specification if he wants to create a valid model.

The prototype implementation of the Policy Support System currently has some weaknesses which prevent it from being used in a production environment:

- Scalability and performance of the Process/Object Engine has not been tested for a larger amount of security object instances. The current implementation has known limitations which threaten the scalability of the system.
- The Process/Object Engine does not yet store historic data about security objects. Currently, only process execution history is collected by our system.
- The Policy Explorer is built based on the standard Activiti user interface. This web-application is designed for a generic process execution system and not for the execution of security policies. The usability of the GUI could be improved by tailoring it to the execution of policies.
- Security of the Policy Support System is currently not considered at all. The system uses the standard Activiti user management. User authentication and the security of the system must be improved, if it is installed in a productive environment.

6.5.2 Threats to the Validity of the Validation

There are several threats to the validity of our validation. A deeper evaluation of the Integrated Modeling Framework for Operational Security Policies should address the following concerns:

Selection of the Interviewees

All interviewees are security experts from the IBM research security division. These users can only represent the real end-users of the system to a limited degree. The interviewees are more proficient in defining and executing policies than typical users. Nevertheless they are not process modeling experts.

Selection of the Case Study

We have only modeled a sub-part of a case study, which we have selected ourselves. A broader validation should cover a full security policy and also consider security policies from other information systems. In addition, many of the requirements were elicited based on the same case study which is used to validate the framework.

Validation Method and Scope

During the first part of the validation, we have validated the modeling methodology and design process of security policy ourselves. To acquire significant results our methodology must be applied by other people to design different security policies.

In the second phase of the evaluation, we have covered the manual and assisted execution of a SOML security policy. We have presented the models and the Policy Support System to the interviewees. The interviewees could only act through the execution, since in practice the execution spans over a longer time frame. A long term study with more participants would be required to efficiently answer, whether our approach is accepted by the users and ultimately reduces the number of human errors.

Conclusion

The security of information systems used in practice, is often breached due to human errors or processes which are not being executed correctly. Security policies are typically used to specify these processes and regulate how people interact with the systems. We have identified several weaknesses in the current practices of creating and applying a security policy. We proposed the Integrated Modeling Framework for Operational Security Policies to address these problems using techniques known from Business Process Management (BPM) and information modeling.

In this thesis, we have proposed the Integrated Modeling Framework for Operational Security Policies to specify the processes and security objects of security policies. We have developed a new modeling language based on the Unified Modeling Language (UML) and Business Process Model and Notation (BPMN) standards. This language allows us to formally specify security objects and the processes that are required to manage these objects. These process and data models can be included in the policy documentation to clarify operating procedures and data. The formal models reduce possible ambiguities in contrast to policies written solely in natural language.

Our modeling framework includes a methodology to design and enact a security policy. The thesis is completed by a prototype implementation of the Policy Support System. This system can execute process and security object models. It manages security objects, such as keys, and assists users during the application of the policy.

7.1 Summary of Contributions

In this thesis, we proposed the Security Object Modeling Language (SOML). This language contributes a UML profile for modeling the information and the dynamic behavior of security objects. BPMN process models are then used to describe how these security objects are created and managed. Therefore we defined concrete execution semantics for the BPMN data modeling elements and the interaction between process and security object instances. These modeling constructs allow us to specify the interaction with security objects in the process models. In addition to standard BPMN process models, our processes can create new security object instances, change their data and state or react to conditional events. Our models thereby guarantee that processes do not violate the integrity of the security objects. In particular, processes can only perform state changes which are supported by a security object's lifecycle. In Section 2.4, we have presented existing work which combines process, information and lifecycle models. These approaches did not match with our requirements, which lead to the development of the SOML language. Table 7.1 summarizes the features of our language and compares it to the existing approaches shown in Table 2.3.

	<i>SOML</i>	<i>Advantages</i>	<i>Limitations</i>
<i>Information Model</i>	✓	The SOML information model is used to build a domain model of the objects that occur in the policy. Relying on UML models allows us to reuse existing tools to create and present the models. Approaches with XML definitions for the information model require custom design tools or do not provide a graphical notation.	The current SOML language specification does not allow inheritance and only supports basic type constraints. Adding sub-typing would allow the designer to reduce redundancies in the models. Advanced OCL constraints could be used to further restrict attributes and states.
<i>Technology</i>	UML		
<i>Attributes</i>	✓		
<i>Associations</i>	✓		
<i>Integrity Constraints</i>	–		
<i>Sub-Typing</i>	–		
<i>Lifecycle Model</i>	✓	The lifecycle model built on UML benefits from the same advantages as the information model. The model is kept as simple as possible and does not rely on advanced UML elements which unnecessarily increase the complexity of the diagrams.	The lifecycle model cannot be used to represent dependencies between two objects, as it is possible with some of the other approaches. The decision not to include branches and dependencies in the lifecycle model was chosen deliberately. Advanced logic, such as the dependency between the states of two objects is represented in the process models.
<i>Technology</i>	UML		
<i>Branches</i>	–		
<i>Dependencies Between Lifecycles</i>	–		
<i>Integrity Constraints</i>	–		
<i>Process Model</i>	✓	SOML process models can make use of BPMN standard elements. The language is compatible with the standard which allows us to use existing BPMN tools. SOML is suited for models used during the designing and the implementation phases. Design models abstract from details and are likely to be included in a policy. Transformation into an executable artifact follows the same principles as normal business processes. The language supports conditional events which are used to start or continue processes. This feature proved as extremely valuable during the evaluation. Our approach is the only one, which supports both, events and allows processes to retrieve object instances.	Compared with most of the existing approaches, two SOML processes can work with the same security object instance. The feature however increases the complexity and introduces the need for synchronization. Our evaluation showed that simultaneous access is currently not a problem for a small policy. Synchronization however poses a treat to the scalability of the models to larger problems. The SOML query and conditional event language, although simple, is sometimes difficult to use. Business rule languages are an alternative worth investigating.
<i>Technology</i>	BPMN		
<i>Data Flow</i>	✓		
<i>State Read/Write</i>	✓		
<i>Data Read/Write</i>	✓		
<i>Concurrent Access by Multiple Processes</i>	✓		
<i>Start Process Based on Data Notifications</i>	✓		
<i>Continue Process Based on Data Notifications</i>	✓		
<i>Integrity Constraints</i>	✓Code		
<i>Access Permissions</i>	✓	SOML supports access permissions to objects and their attributes.	
<i>Data Execution Semantics</i>	✓	SOML models can be enacted in the Policy Support System.	

Table 7.1: SOML Compared to the Approaches Shown in Table 2.3.

We have extended an existing BPM methodology in order to apply it to design and enact security policies. The methodology is used to elicit security objects and processes. During a design phase these processes and objects are modeled using the SOML language. The resulting models are integrated in the policy documents and converted into executable artifacts. Detailed instructions, which are required to complete the processes, are added to the executable models. These instructions are provided to the user once the process models are executed in the Policy Support System.

We have built a prototype implementation of the Policy Support System based on the Activiti process engine. This prototype extends the Activiti engine with new capabilities, which are required to execute SOML security object and process models. The original engine itself already provides native support for BPMN process models. Our extension raises security objects to the same level as processes. Security object models can be deployed along with process models. Our engine executes

the SOML models and provides an API which is used to interact with the system. The prototype includes the Policy Explorer, an adapted version of the Activiti Explorer, which allows users to interact with the engine. The explorer assists users during the application of the policy and provides detailed information about the existing security object instances.

Figure 7.1 shows how the Policy Support System uses (modeling) technology to reduce the risk of human errors and failed process executions. The system either automates tasks or provides detailed instructions which assist users as they complete tasks. On the other hand, the system governs the process execution and ensures that users comply with the processes of the security policy.

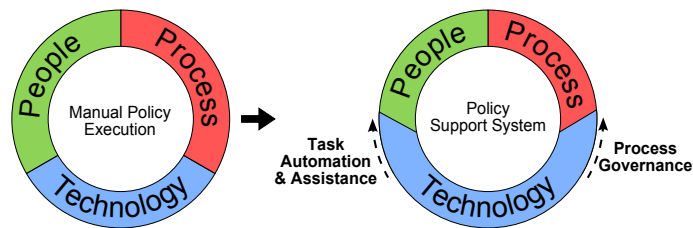


Figure 7.1: Reduce Risks with Task Automation and Process Governance

We have validated the Integrated Modeling Framework for Operational Security Policies including the Policy Support System based on a case study. The validation showed that our framework is capable of representing the operational processes and security objects which occur in the case study. The validation revealed several advantages of our approach over a manual execution of the policy.

The Policy Support System provides self-contained task instructions when they are required to complete a task. Our system integrates all documentation in a central place whereas in a normal security policy the instructions of a single task are often scattered over different documents.

Security object lifecycle management, a core task of any security policy application, is carried out by the Policy Support System. The system stores any information about the current data and state of the existing security objects. More important, it reacts to conditions stated on security objects. The system can, for example, proactively notify a user if a key is about to expire.

Last but not least, the Policy Support System governs the process execution. The system ensures that tasks are executed in the correct order, at the right time, and by the person in charge. Runtime data of the process executions is collected during the lifetime of a process instance. The data may be used by auditors to verify compliance between the execution and the security policy.

7.2 Future Work and Research

Further work could be performed to validate and increase the utility of the Integrated Modeling Framework for Operational Security Policies. First of all, our validation only considered a single security policy. Applying the language and design methodology to other security policies would be required to gain further insight on the applicability of the framework. A long-term validation in a production environment would be required to evaluate, whether the Policy Support System can reduce the number of security incidents during the actual execution of the policy.

During the design and modeling of the policy we observed that the standard UML and BPMN design tools poorly support the design process. Defining a valid model using these tools requires knowledge of the Security Object Modeling Language. Tailored tools, with support for the language, could simplify the development process. Such tools could also perform static validation of the

consistency between models. This verification could help to ensure that designers only create models which satisfy the definition of consistency as introduced in Section 4.2.1.

The validation also revealed that the usability of the Policy Explorer GUI could be improved. The current interface is based on a generic interface which is designed to be used with business processes. It could be further tailored to the usage scenarios in combination with policies. A particular problem of the system is, that users are sometimes not allowed to access other systems as they are in a highly secured data-center. A user might therefore not have direct access to our system and use it offline. As soon as the user leaves the data-center he would upload his progress and advance the process. This data input is both cumbersome and introduces the possibility of typing errors. An alternative would be to use print media, to replace manual data input. In the case study, some of the highly secured system print a receipt (a barcode) once an operation has been completed. These receipt could later be parsed by the Policy Support System and update the process state. The use of barcodes would ensure that the data is correctly supplied to the Policy Support System.

Modeling Beyond Processes and Security Objects

The focus of this thesis was on the security objects and operational processes of a security policy. Practitioners however face additional challenges during the definition, audit and application of security policies. Modeling and formal methods could be leveraged to radically change the practices of creating, managing and applying a security policy.

Security policies consist of more than security objects and operational processes. These aspects are currently not covered by our approach. To fully exploit the advantages of formal models, one could model roles, infrastructure, communication, risks, threats, audit and other policy requirements.

Many of these requirements could be reused across different organizations. In fact, policies used today are often created based on industry standard templates. Likewise a designer could choose model fragments or patterns from a library containing predefined best practices. The designer could further tailor these patterns to his needs or even adapt them through configuration. The creation of such a component library could allow cost efficient implementation of a security policy.

As part of this thesis we have focused on the implementation and execution of a security policy. On a larger scope, people already struggle to define policy requirements and turn them into a policy compliant implementation. The definition of high-level policy requirements is still done using natural language. Our approach merely helps to foster discussion of the implementation on the basis of formal process and information models. We envision that formal methods could be further leveraged to specify the high-level requirements of a security policy. Putting this together with the other ideas presented in this section could significantly change the way security policies are defined and managed.

Transfer the Framework to Other Domains

We have only applied the Integrated Modeling Framework for Operational Security Policies to the definition of processes and security objects in the context of security policies. The concepts of the SOML language and the Policy Support System could also be transferred to other application domains that require strict modeling of data and processes.

Security Object Modeling Language Requirements

A.1 Security Object Information Model

Model a security object (IM 01)

A security object information model can contain an arbitrary number of autonomous objects. These objects serve as blueprints, from which concrete object instances are derived at runtime. Every object is denoted by a unique name within the context of the model.

Define permissions to view object instances (IM 02)

An object can define process users that are allowed to view instances of the object at runtime. Users, that do not have the permissions, cannot see and access the instances.

Model inheritance between two objects (IM 03)

An object may inherit from another object in the same information model. The derived object inherits all properties and permissions from the super object. Permissions may be overwritten in the derived object. Properties cannot be overwritten and their name must remain unique. An object can be defined as abstract which prevents it from being instantiated at runtime.

Example (Inheritance between keys and certificates)

Cryptographic primitives such as keys and certificates often share common attributes. Inheritance support reduces the modeling effort by defining shared attributes in a common super class.



Define the asset classification of an object (IM 04)

The classification specifies the quality levels, such as confidentiality, integrity or availability required when working with the object. Likewise the processes or its sub-elements are annotated with qualities they can provide. Objects cannot be used in parts of the processes that do not comply with their requirements. The modeling framework guarantees that an object is only used in a part of the processor that fulfills the asset's classification level. An asset classification may depend on the current state and data of the object.

Example (Protecting a private key)

A private key should never be exposed to external parties. Organizations therefore take special precautions how a private key is stored and used. In the most rigorous setup the key only resides within a Hardware Security Module (HSM). The security analyst could therefore annotate the key with an asset classification: $[state = active \wedge type \neq 'test' \Rightarrow C_{HSM}]$. A key that is in state active and not marked as test key can only be used in parts of the process that are marked with C_{HSM} .

Model data properties of an object (IM 10)

An object may contain an arbitrary number of properties that define the information held by the object. Each property can be identified by its unique name within the object. In addition a property models the following information:

- The data-type of the data.
- An optional default value.

Define permissions to read a property (IM 11)

A property can define process users that are allowed to read the value of the property. Users that do not have the permission cannot read the property. Processes are not affected by the restriction regardless of which user is executing the process.

Mark a property as unique identifier (IM 12)

A property can be defined as unique. Such a property must have a unique value for all instances of the object. The property is not bound to this constraint as long as it has not yet been initialized.

Example (Unique key modulus and exponent)

Exponent and modulus of a key must be unique. It is very unlikely that the key generation algorithm generates two identical keys. The restriction would prevent users from loading the same key twice into the system.

Model relations between objects (IM 20)

An association models a relationship between exactly two objects. The association denotes a relationship where a container object “has a” related object. The related object can thus exist without the other object. In addition the following statements hold for the relationship:

- The relationship has a unique name in the scope of both objects.
- The relationship can only be navigated from the container object to the related object.
- The container-end must have multiplicity between zero and one.
- The related-end can have an arbitrary multiplicity.

Example (Key-Certificate relation)

A key-pair and a certificate are related to each other in different ways. A certificate signs the public-key of the users key pair and thereby guarantees that the key is indeed owned by the user. The private-key of the certification authority is used to create the certificate. These relations can then be used to access data or manipulate related objects:

- *A process that uses a certificate can access the signed key to validate, if the expiration date of the key and certificate are correct.*
- *A certificate is revoked if the user loses his private key. Changing the state of the key to*

“compromised” therefore also affects the certificate which has to be revoked.



Define constraints on data (IM 30)

Additional constraints on the data of an object can be defined. The constraints serve as invariants of the object and have to be fulfilled at any given time. A constraint restricts the possible data of a property based on:

- Other data held by the object.
- The current state of the object.
- The data and state of the related object. (An object can have multiple related objects. Set operations \exists and \forall must be supported.)

Example (Public–Key of a certificate must be valid)

A certificate should not be valid for a longer period than the public key it signs. The key therefore defines an invariant that checks the expiration dates:

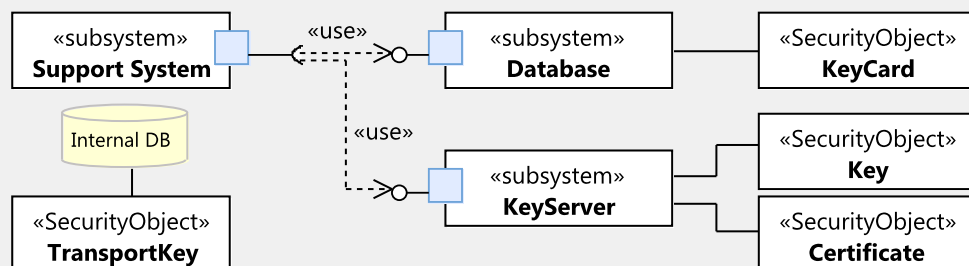
$[\forall c \in \text{Key.Certificates}: \text{Key.Expires} \geq c.Expires]$

Define the execution environment of the object (IM 40)

An object is not necessarily managed and stored within the Policy Support System. It may also be located on an external system such as a corporate database. The execution environment can therefore be defined when the process is implemented. The definition might require additional configuration data depending on the nature of the external system. The developer also has to provide the Policy Support System with an endpoint implementation that allows the system to access the object, when located on the external system. *Note: The creation of endpoints for external systems is not in the scope of the project.*

Example (Objects stored on external systems)

Legacy systems are already in use to store certain objects. These systems cannot be replaced and they have to be integrated with the Policy Support System.



A.2 Security Object Lifecycle Model

Model the behavior of an object (LC 01)

Each object is defined by a behavior that describes the lifecycle of the object. The lifecycle modeled is based on the definitions of a finite state machine [Bla]: [Lifecycle = $(Q, \Sigma, \delta, q_0, F)$] where:

- Q A finite set of states in which the object can reside during its lifetime. Exactly one state is active at any point in the object's lifetime.
- Σ A finite set of triggers that can fire a transition.
- δ A transition function $\delta : Q \times \Sigma \rightarrow Q$ that defines how the occurrence of a trigger changes the state of the object.
- q_0 One initial state. ($q_0 \in Q$)
- F A set of final states, which mark the end of the objects lifetime. ($F \subseteq Q$)

Share behavior between objects (LC 02)

Different objects can exhibit the same or similar behavior. These objects may share parts of their lifecycle models to prevent redundancies in the specification.

Define entry and exit actions of a state (LC 10)

Each state can contain entry or exit actions that are executed when the state is entered or left. Entry and exit actions are defined in an appropriate programming or scripting language. Entry and exit actions can also serve as replacement for state pre- and post-conditions.

Use composite states (LC 11)

A state may contain a sub-behavior that specifies the detailed behavior of a state. Within the composite state the same functionality of the lifecycle model may be used: [$CompositeState \in Q$] = $(Q', \Sigma', \delta', q'_0, F')$ where:

- δ' A transition function $\delta' : Q' \times \Sigma' \rightarrow Q'$ that defines how the occurrence of a trigger affects the state of the object. While the object is in a composite state both functions δ and δ' are active.
- F' A set of final states which mark the end of the contained behavior. ($F' \subseteq Q'$)

Define constraints on states (LC 12)

Additional invariant, pre- and post-condition constraints on the state can be defined in the lifecycle model. Invariant constraints have to be fulfilled while the object resides in this state. Pre- and post-conditions have to be fulfilled as the object enters or leaves the state. Violation of a constraint results in an error. Constraints restrict the state based on:

- Data held by the object.
- The data and state of the related objects. Set operations \exists and \forall must be supported since a relation can point to a collection of related objects.

Example (A key must have a certificate when in state active)

A CA-Key must be certified by the root-CA before it can be used to sign keys. The key state Active therefore defines an invariant that checks if a certificate issued by the root-CA exists:

$[\exists c \in Key.Certificates : c.Issuer = 'Root-CA']$

Define transition triggers (LC 20)

The transition triggers (Σ) define the signals that can cause a change of the objects state. The model can use two different types of transition triggers:

Initial The initial trigger connects the initial pseudo-state with the first state of the objects lifecycle. Only one initial transition is supported. The initial trigger is automatically fired upon creation of the object.

External External transitions are triggered by processes which request the execution of a state change. Each external trigger is named after the state to which it connects. Processes use the name of the target state to identify the transition that has to be taken.

Figure A.1 shows the different triggers. Transitions between two states implicitly receive an external trigger based on the target state. The designer does not have to model the triggers or name the transitions.

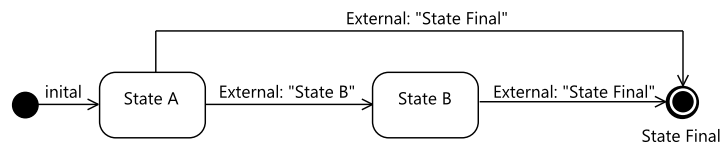
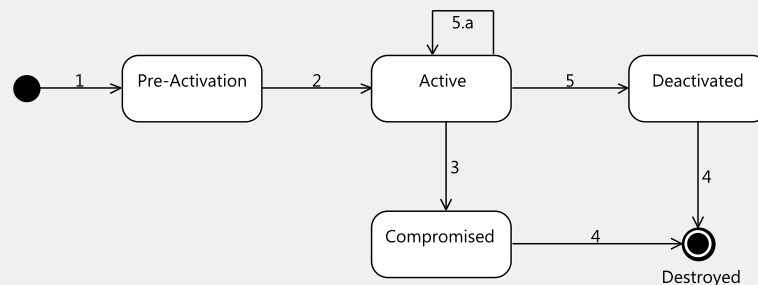


Figure A.1: Initial and External Triggers

Example (NIST key lifecycle)

The key lifecycle defined by NIST [BBB⁺12] shows the possible usage of the triggers:

**1 Initial**

Upon creation, the key is automatically moved into the state: “Pre-Activation”.

2 Active

The generation process is still running while the key is in the state “Pre-Activation”. Eventually the process will reach a point where the key has been distributed and the transition to “Active” is triggered by the process.

3 Compromised

Loss of a key is typically not planned and a result of an external event. The user therefore manually starts a process that changes the keys state to compromised.

4 Destroyed

A key is destroyed after it has been in the “Deactivated” state for a fixed time. Keys of a PKI infrastructure often define an expiration date. When the expiration date is reached, a process must be started which deactivates the key.

5 Deactivated

The expiration date and a negative offset can be used to warn users before the event happens (5.a).

Model choices (LC 30)

A choice $Q^{Choice} \in Q$ is a pseudo state with multiple outgoing triggers Σ . A choice is used to model a conditional path in the lifecycle of the object. Exactly one transition guard of the outgoing transitions must evaluate to **true**.

Define transition conditions (LC 31)

A transition can be guarded by a condition and is fired only if the guard evaluates to **true**. The condition is evaluated as the transition is about to fire. Conditions are defined in an appropriate programming or scripting language.

A.3 Using Security Objects in a Process

Create new object instances (P 01)

An instance of an object can be created from within a process. The name of the object is used to define the object type when creating an instance. The name of the security object model must be used in case multiple information models are present in a deployment container. The process automatically holds a reference to the instance after it has been created.

Delete existing object instances (P 02)

An instance of an object can be deleted from within a process.

Get data and state of an object instance (P 03)

A process can read the data and current state of an object instance. A reference to the object instance is required to access the information. The process can use the information as any other data. (*E.g. As a condition of a sequence flow.*)

Set data of an object instance (P 04)

A process can set the data of an object instance. A reference to the object instance is required to write the data.

Model accepted states of an activity (P 10)

An activity can define a set of accepted states in which the object must reside as the activity starts. Figure A.2 shows the constructs for defining one or more accepted states.



Figure A.2: Accepted State(s)

Model produced state of an activity (P 11)

An activity can model the state change that is executed on a security object when the activity completes. A reference to the object instance is required to initiate a state change. A single activity may define a produced state for multiple object instances. The state change is only successfully

completed by the object if a valid transition between the currently active state and target state exists. The process requests a state change by sending the corresponding trigger (= identifier of the target state) to the object. A process can use standard BPMN error handling mechanisms (Interrupting Boundary Event) to react to an unsuccessful state change. Figure A.3 shows the constructs for requesting a state change and handling an error.

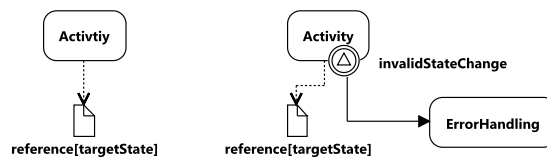


Figure A.3: Produced State and Error Handling

Retrieve existing object instances (P 20)

A process can use a global repository to search for existing object instances. The search interface supports searching for:

Object Type	The type (name) of the object.
Current Data	Conditions on the current data of the object itself and its related objects.
Current State	Conditions on the current state of the object itself and its related objects.

Example (Periodic destruction of old and invalid cards)

Returned or defect cards must be correctly disposed. A periodic card destruction process is executed to destroy these cards. This process needs to find all cards that should be destroyed. A query to find all expired cards and all invalid productive cards is used to retrieve the affected cards:
 $[this.state \in \{Expired\} \text{ OR } (this.state \in \{Invalid\} \text{ AND } this.type \neq 'Test')]$

Work on collections of object instances (P 30)

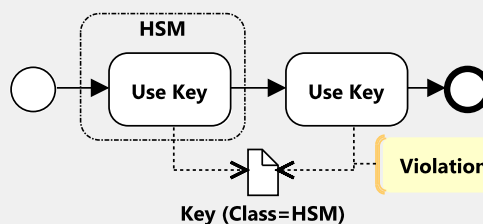
A process can simultaneously work on multiple security objects of the same type.

Annotate the process with security zones (P 40)

The process model can be annotated with security zones. A security zone annotation defines what requirements the annotated part of the process can fulfill. Security zones are generic and must be defined by the security analyst prior to including them into the model.

Example (Protecting a private key)

A private key that is annotated with a HSM asset classification is used in a process. Parts of the process that are not allowed to work on objects with the classification HSM cannot use the key.



A.4 Interaction between Security Objects and Processes

Trigger a conditional start event (I 01)

A process can use the BPMN conditional event construct in conjunction with security object instances. The BPMN standard defines conditional events as follows:

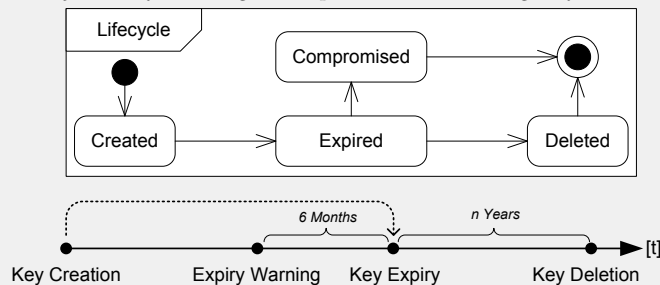
*This type of event is triggered when a condition ... becomes true. ... The Condition Expression of a Conditional Start Event MUST NOT refer to the data context or instance attribute of the Process (...). Instead, it MAY refer to static Process attributes and **states of entities in the environment**. The specification of mechanisms to access such states is out of scope of the standard. ... [OMG11a]*

A process can define a conditional start event based on the data and state of security objects. A conditional event is triggered whenever a condition changes from *false* to *true*. An instance of the process is created once a conditional event is fired. The process automatically receives the object reference that was responsible for firing the conditional start event. A distinct process instance is created for each security object, if multiple object instances fulfill the condition simultaneously. The following conditions can be stated on security object instances:

Object Type	The type (name) of the object.
Current Data	Conditions on the current data of the object itself and its related objects.
Current State	Conditions on the current state of the object itself and its related objects.
Time Data	Conditions on a data property compared to the current time/date.

Example (Key Lifecycle)

Cryptography material is often tied to an expiry date. Different processes need to be executed to handle the expiry of a key. The following example shows the usage of the conditional start event.



Key Creation	<i>A key is generated by a process that has been manually started. The expiry date of the key is set when it is generated and stored as a property of the key.</i>
Key Expiry	<i>A conditional start event is used to start the process to handle the expiry of the key. The conditional event defines a condition on the expiry date property of the key: $this.expiryDate \equiv NOW$</i>
Expiry Warning	<i>Expiry of a key used in production can have severe consequences. Therefore users would like to receive a notification beforehand. A simple process that notifies the user can be started based on a condition: $this.expiryDate - 6M \equiv NOW \text{ AND } this.type \neq "Test"$</i>
Key Deletion	<i>A key can be deleted after it has been expired for a certain amount of time. The deletion process is triggered if the key remains in the state expired for a fixed delay: $this.state \in \{Expired\} \equiv n \text{ Years}$</i>

Trigger a conditional intermediate event (I 02)

Intermediate conditional events work the same way as conditional start events. Instead of starting a new process instance they continue the execution of a running instance. Figure A.4 shows the definition of an intermediate conditional event.

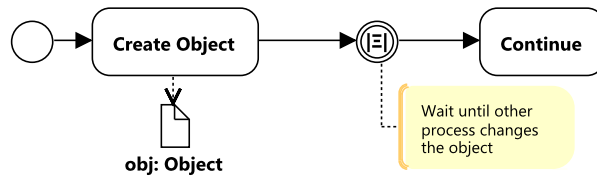


Figure A.4: Intermediate Conditional Event

Change the state of an object instance (I 10)

A process can request a state change in the object's lifecycle model by sending the corresponding signal. A reference to the object instance is required to request a transition. In order to change the state of a security object it needs to receive a signal from a process that causes the state transition to be executed. Figure A.5 shows how signals are received and processed by the object.

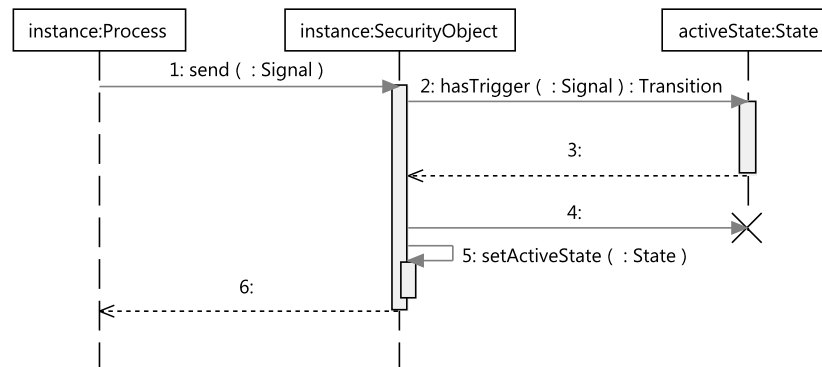


Figure A.5: Sequence Diagram: Send/Receive Signals

Signals are always generated outside the scope of the security object and received by the object. Receiving a signal in a security object can cause a transition to be executed. The following steps are executed when a signal is sent to a security object:

- 1 The signal is sent by the process and received by the security object. The process is blocked until the state transition has been processed.
- 2-3 The security object checks whether the current state is waiting for the signal.
- 4-5 If the requested transition is valid, it is executed. The security object changes its active state to the new states. Any exit actions of the old state and entry actions of the new state are executed as part of the transition.
- 6 The process receives a return value indicating whether the transition was successfully completed.

A.5 Initial Modeling Example

Section A.5 shows a real world example from the case study. The process, information and behavior models only cover parts that are relevant in the context of the *MSCA Key-Pair* security object. *The examples use visualization and model constructs that are not fully compatible with the Integrated Modeling Framework for Operational Security Policies and therefore not executable. The example serves as basis to elaborate different requirements and their relations based on a real world example.*

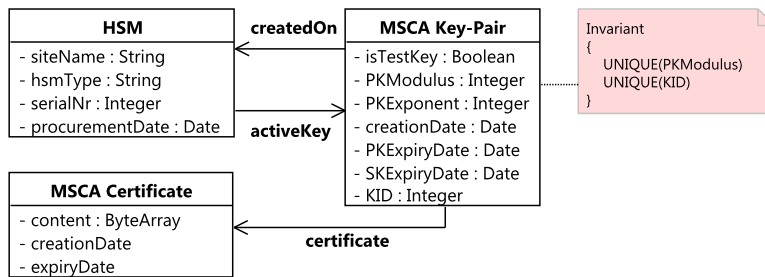


Figure A.6: Information Model

Figure A.6 shows the information model of the *MSCA Key-Pair*. The model part defines the attributes and relations of the *MSCA Key-Pair* security object. *Remarks:*

- The *KID* property of the *MSCA Key-Pair* is unique. Each instance of a *MSCA Key-Pair* is therefore guaranteed to have a unique *KID* among all instances of *MSCA Key-Pairs*.
- The model specifies the relations of the *MSCA Key-Pair* with other security objects. An *MSCA Key-Pair* is always created on a distinct *HSM*. The *HSM* in turn has a relation to the currently active *MSCA Key-Pair*. ...

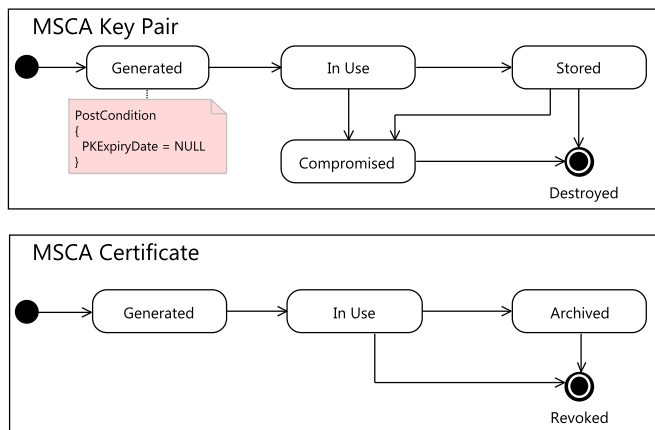


Figure A.7: MSCA Key-Pair & Certificate Behavior

Figure A.7 shows the dynamic behavior of the *MSCA Key-Pair* and *MSCA Certificate*. Both security objects are created by the *MSCA Key Generation* process. *Remarks:*

- The example shows the different triggers that can fire a transition:
 - Initial** The initial transition activates the first state upon creation of the object.
 - External** All other transitions are executed when a process instance sends the corresponding signal.
- The behavior model defines constraints (Pre- and Post-conditions) on different states. Violating such a condition results in a runtime error. Constraints are implemented using entry and exit actions of the state.

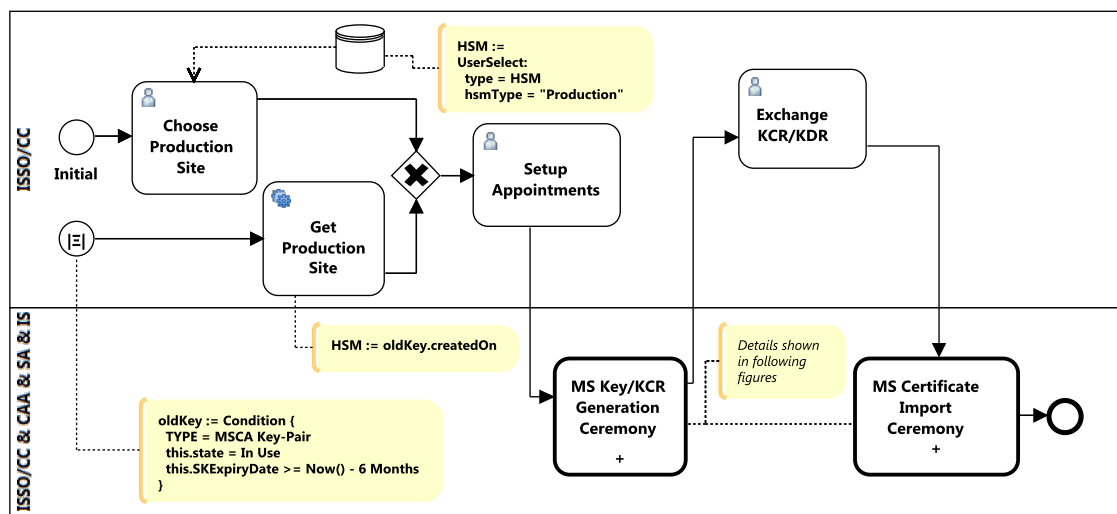


Figure A.8: Key Generation Process Overview

Figure A.8 shows the *MSCA Key Generation* process. The process instance creating the first *MSCA Key-Pair* must be manually started. Further process instances are automatically started based on the previously created *MSCA Key-Pair* security object. *Remarks:*

- The first activity of the process selects the *HSM* on which the *MSCA Key-Pair* is generated. The *HSM* is either already known, based on the previous *MSCA Key-Pair* that started the process or must be selected by the user. Therefore the process uses a repository to detect available *HSM* instances and presents them to the user. The *HSM* instances are used by the process just as any other process variable.

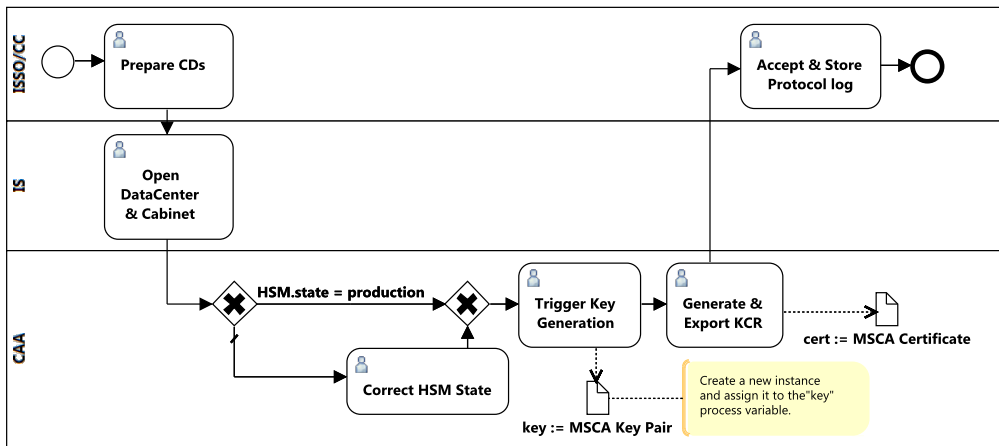


Figure A.9: Key Generation Process Detail (1/2)

Figure A.9 shows the *Generation Ceremony* part of the process. *Remarks:*

- The example shows, how the state of the *HSM* process variable (which is a security object instance) is used as an input for a BPMN gateway.
- Executing the activity that creates the key also constructs a new instance of the *MSCA Key-Pair* security object. After the instance has been constructed, the data and relationships are initialized. Creating and editing security objects is done using script code.

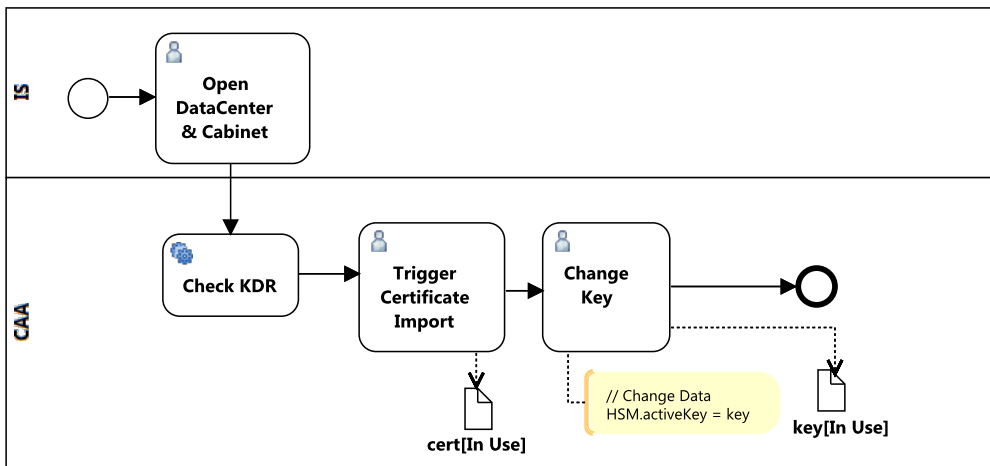


Figure A.10: Key Generation Process Detail (2/2)

Figure A.10 shows the *Import Ceremony* part of the process. *Remarks:*

- The example shows, how the currently active *MSCA Key-Pair* of the *HSM* is changed to match with the new *MSCA Key-Pair* once the key has been imported.

- The activity *Change Key* causes a state change in the *MSCA Key-Pair* by sending the *Key Usage* signal.

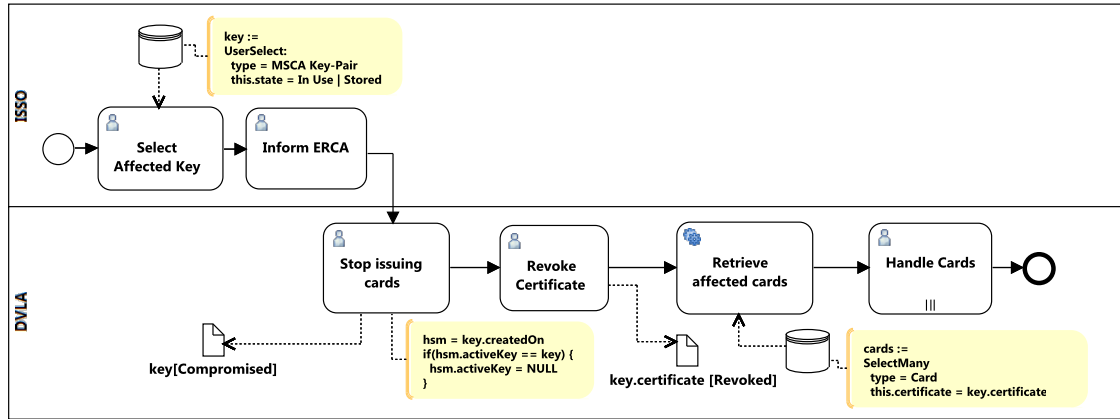


Figure A.11: Key Compromise Process Detail

Figure A.11 shows the process to handle the loss of an *MSCA Key-Pair*. *Remarks:*

- The first activity of the *Key Compromise* process asks the user for the affected key.
- Revoking the related *MSCA Certificate* is required and performed by the process.
- The example shows an advanced usage of the repository: The process uses a query to find a list of cards that were created using the affected *MSCA Key-Pair*.

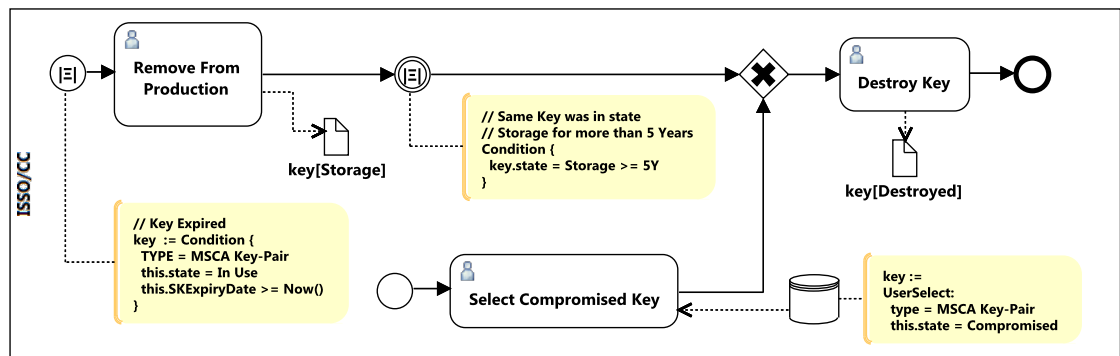


Figure A.12: Key End of Life (Simplified)

Figure A.12 shows the process to handle the end of life of an *MSCA Key-Pair*. *Remarks:*

- The process is started automatically if a key expires or it is manually triggered in case of a compromised key.
- The process uses an intermediate event to wait until the key has been in the state *Storage* for five years and then continues to process it.

Specification and Implementation Details

B.1 Data Modeling Grammar

Section B.1 covers the grammar used by the data modeling elements of key management processes. The notation of the parser generator ANTLR v4 [PQ95] is used to present the grammar.

```
// The SOML grammar is used to parse the SOML comands that are contained in BPMN
// Data Objects, Annotations and Conditional Events. The gammar is based on the
// Java grammar. (See https://github.com/antlr/grammars-v4)

// Starting point for parsing a SOML command or conditional expression
soml : (somlCommands | somlEventCondition) EOF;

// A list of one or more SOML commands.
somlCommands : (createInstance | setAttribute | changeState | queryRepository)+;
// Command used to create an instance of a SO.
createInstance : soReference ':= ' soType ';' ;
// Command used to set an attribute of a SO instance
setAttribute : soReference (',' soAttribute)+ ':= '
               (literal | processVariable | soProcessVariableSet) ';' ;
// Command used to change the state of a SO instance.
changeState : soReference (',' soAttribute)* '[' soState ']' ';' ;
// Command used to query the repository and assign the result to a variable.
queryRepository : soReference ':= ' '{' condition '}'
                 ',' ('selectOne()' | 'selectMany()') ';' ;

// Condition expression for a BPMN start or intermediate conditional event
somlEventCondition
: soReference ':= ' soType ',' '{' (dataCondition | stateCondition)? '}' ';' ;
```

Listing B.1: SOML Grammar (1/3: Commands and Events)

```
soAttribute       : Identifier ;
nestedAttribute   : soAttribute (',' soAttribute)* ;
processVariable   : Identifier ;
soProcessVariableSet : '{' processVariable (',' processVariable)* '}' ;
soReference       : Identifier ;
soType           : Identifier '::' Identifier ;
soState         : Identifier ;
```

Listing B.2: SOML Grammar (2/3: Types)

```

condition
  : '(' condition ')'
  | notCondition
  | typeCondition
  | dataCondition
  | stateCondition
  | setCondition
  | condition '&&' condition
  | condition '||' condition;

// Negation of another condition
notCondition : '!' condition;

// Is instance of condition
typeCondition : 'instanceof' soType;

// Condition on data
dataCondition
  : soAttribute ('<=' | '>=' | '>' | '<' | '==' | '!=')
    (literal | processVariable | ('now' (('+' | '-') IntegerLiteral)?));

// Condition on state
stateCondition
  : ('inState' | 'notInState') '(' soState (',' soState)* ')';

// Condition on referenced data
setCondition
  : ( 'forAll' | 'exists' | 'notExists' )
    '(' nestedAttribute ',' (dataCondition | stateCondition) ')';

```

Listing B.3: SOML Grammar (3/3: Conditions)

The SOML grammar uses the same specification of identifiers and literals as defined by the Java language specification [GJSB05]. (*Identifier*, *literal* and *IntegerLiteral* are based on the Java specification and not shown in this grammar specification.)

B.2 Process/Object Engine Database Schema

Section B.2 specifies the database extensions added by the *Activiti-Object-Engine*.

<i>Field</i>	<i>Description</i>
ID_ (Primary Key)	Generated id in the form of: <Key><Version>:<autoIncrementInteger>
REV_	The revision. The revision is incremented on every update of the table.
NAME_	The name of the security object definition. (<objectName>)
KEY_	The fully qualified name of the definition. (<modelName>:<objectName>)
VERSION_	The version. The version is incremented when a definition with the same key is already deployed.
DEPLOYMENT_ID_	Foreign key to the deployment in which the definition is contained.
RESOURCE_NAME_	The name of the resource, in which the security object was contained.
DGRM_RESOURCE_NAME_	The name of the diagram image resource.
SUSPENSION_STATE_	The current suspension state of the definition.

Continued on next page

Table B.1 – Continued from previous page

	Field	Description
ACT_RU_SO	ID_ (Primary Key)	Generated id in the form of: <autoIncrementInteger>
	REV_	The revision. The revision is incremented on every update of the table.
	BUSINESS_KEY_	An optional, unique string that can be provided by the user upon creation.
	SO_DEF_ID_	Foreign key to the definition of which the instance is derived.
	STATE_ID_	The identifier of the current state as defined in the lifecycle model.
	SUSPENSION_STATE_	The current suspension state of the instance.
ACT_RU_ATTRIBUTE	ID_ (Primary Key)	Generated id in the form of: <autoIncrementInteger>
	REV_	The revision. The revision is incremented on every update of the table.
	TYPE_	The data type of the attribute.
	SO_ID_	Foreign key to the instance to which the attribute belongs.
	NAME_	The name of the attribute.
	VALUE_	Serialized data of the attribute.
	VALUE_DOUBLE_	The value in double format. (If applicable.)
ACT_RU_CE	ID_ (Primary Key)	Generated id in the form of: <autoIncrementInteger>
	REV_	The revision. The revision is incremented on every update of the table.
	SO_ID_	Foreign key to the instance to which the state belongs.
	STATUS_	The current state of the condition

Table B.1: Process/Object Database Fields

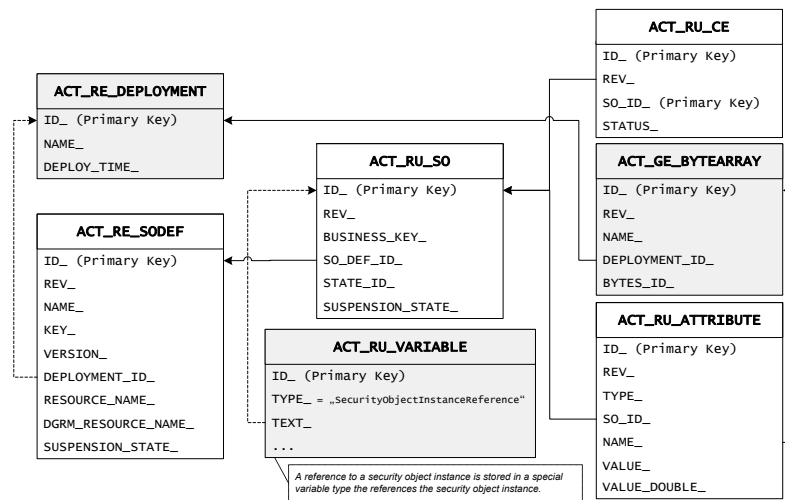


Figure B.1: Process/Object-Engine Database Schema

Validation

C.1 Interview Questionnaire

The following questionnaire is used to validate the Integrated Modeling Framework for Operational Security Policies. The questionnaire uses the roles described in Table C.1.

<i>Role</i>	<i>Description</i>
Security Expert	A security domain expert which is responsible to create security policies. A common qualification held by such an expert is the Certified Information Systems Security Professional (CISSP) certification.
User	A typical end-user which has to apply the security policy.
Auditor	An internal or external auditor which audits the policy documents and controls the application of the policy in practice.

Table C.1: Roles Used in the Questionnaire

Security Policy Definition and Implementation

The following questions are asked after the interviewee has examined the tachograph policy which we have modeled using the Integrated Modeling Framework for Operational Security Policies:

-
- 1 Is a formal specification of attributes, relations and lifecycle of security objects required to define and implement a security policy? If yes, how have existing policies been created?

 - 2 Does the formal specification of the lifecycles and processes make it easier to reason about the completeness of the policy?

 - 3 Are the security object models of the case-study comprehensible to a security expert?

 - 4 Are the process models of the case-study comprehensible to a security expert?

 - 5 Could a security expert, which has received a training, model and implement a security policy with our framework? Could he also build an executable process model by himself?

 - 6 Are the implementation process models too complex/detailed to be included in a policy documentation? Is the distinction between design and implementation models required?
-

-
- 7 Would you include security object models in a policy or practice statement document?

 - 8 Are there aspects of security objects which are not represented by the security object model?

 - 9 Is the security object model specification missing any important features or notations?

 - 10 Does a policy consist of more than one security object models? Is it necessary to split the definitions into multiple model to cope with the complexity/size of the domain?

 - 11 Do lifecycle models need to capture dependencies between the states of two related security objects? Or is it sufficient if process model can contain such constraints?
(E.g.: If key is compromised then the related certificate must be in the state revoked.)

 - 12 Are there operations on object instances which are not supported by the process models?

 - 13 Are conditional events expressive enough to react to changes of security object instances?
-

Security Policy Application (Manual)

The following questions are asked after the interviewee has manually executed the "key generation" process according to our models:

-
- 14 Are the security object and process models of the case-study comprehensible to a user?

 - 15 Does the formal specification increase the comprehensibility of the policy?

 - 16 Does the formal specification reduce ambiguities in the policy?

 - 17 What could be added to the process and security object models to further reduce the risk of errors during the execution? How could the comprehensibility of the models be improved?

 - 18 Is important to track data and state of security object instances in a central place?

 - 19 Is the Policy Support System mandatory or are models useful for documentation alone?
-

Security Policy Application (With the Process Support System)

The following questions are asked after the interviewee has executed the "key generation" process with the Policy Support System:

-
- 20 Does the system support the user and reduce the risk of human errors?

 - 21 Are the task execution instructions which are pushed to the user more adequate than the policy documents? Is the level of detail of the instruction used in the case-study adequate?

 - 22 Would users and security experts appreciate support during the execution of processes?

 - 23 Would users accept the strict process governance imposed by the Policy Support System?

 - 24 With the current concept, a user interacts with processes which, in the background, modify security objects. Would it be more natural, if users interacted with security objects? Changing the interaction paradigm would however increase the complexity of the security object model. (E.g.: Select a key and click compromised instead of starting the "compromised" process.)
-

-
- 25 The Policy Support System lays the foundation to automated tasks and integrate different system. How much of the work that is currently carried out manually would be suitable for automation? Could you convince an auditor to automate tasks which are currently executed manually?
-
- 26 Do the constraints on attributes (Types) and state changes improve the accuracy of the knowledge about security object instances compared to the current state of the art?
-
- 27 Do users need sophisticated access to the security object instances? (Search, Filters, Predefined Filters, Queries, ...)
-
- 28 Are there aspects of security object lifecycle management which are not covered?
-

Policy and Execution Audit

The following questions are asked after the interviewee has examined the execution logs created by the "key generation" process :

-
- 29 Are the security object and process models of the case-study comprehensible to an auditor?
-
- 30 Are process execution logs collected by the Policy Support System a suitable base to audit a process execution? Would these logs be regarded as trustworthy?
-
- 31 Are execution logs which are automatically collected by the Policy Support System more accurate/trustworthy than the logs which are currently in use?
-
- 32 What is missing to satisfy an auditor? (E.g.: Explicit audit element in a process model which triggers the creation of a log entry. Proof validity of the logs. ...)
-

Productize the Policy Support System

-
- 33 Is there a business value added by the Policy Support System?
-
- 34 What is missing to operate the system in a production environment?
-
- 35 How should the system deal with the long lifetime of processes? How likely are updates to processes/policy? How should an organization deal with the long lifetime of the policy?
-
- 36 Are there aspects of a policy which must be included in the models? (E.g.: Sophisticated Role Modeling, Infrastructure, Security Zones, ...)
-
- 37 Where should the usability of the current client application be improved?
-
- 38 Are there systems which could be integrated with the Policy Support System? (E.g.: Trigger key generation on a Key Server, HSM, ...)
-
- 39 Currently all security objects are stored in the Policy Support System. The system is not aware of security objects which are already managed by other systems. Should the Policy Support System support security objects which are stored on 3rd party systems?
-

Content on the Delivered CD-ROM

- **Abstract.txt** The abstract of the thesis in plain.
- **Zusfsg.txt** The German abstract of the thesis in plain.
- **MasterThesis.pdf** The master thesis (this document).

Glossary

Activiti	Activiti™ is a light-weight workflow and BPM Platform targeted at business people, developers and system admins. Its core is a super-fast and rock-solid BPMN process engine for Java. (xiii, xv, 42, 43, 46–48, 56, 58–66, 76, 78, 79)
Business Activity Monitoring	The discipline of analyzing process execution data/event in real-time. (58)
Business Intelligence	The discipline of analyzing historic process execution data. (58)
e–Business	The use of information and communication technology to support the interaction between corporations, employees and customers. (1)
e–Government	The use of information and communication technology to support the interaction between a government and its citizens. (1)
Eclipse MDT/UML2	An EMF-based implementation of the UML 2.x OMG metamodel for the Eclipse platform. (34, 63)
Groovy	A dynamic, object-oriented programming language for the Java platform. (xv, 41)
identity document	A document which is used to verify aspects of a person's identity. (2, 12, 68)
information modeling	Using a model to specify an abstract, formal representation of entity types that may include their properties, relationships and the operations that can be performed on them. (3, 5, 77)
information security	Information security means protecting information and information systems from unauthorized access, use, disclosure, disruption, modification, or destruction. (5)
Integrated Modeling Framework for Operational Security Policies	Our modeling framework which is used to model, enact and control operational processes of security policies. (3, 4, 24, 25, 28, 31, 32, 52, 54, 67, 69, 72, 73, 75–77, 79, 80, 90, 99)
JUEL	A Java implementation of the Unified Expression Language. (xv, 41, 46)

policy explorer	The web-application that is used to access the Policy Support System. (60, 61, 63–66, 72, 76, 79, 80)
policy support system	The system used to support the execution of a security policy. (4, 25–28, 32–34, 36–38, 42, 53, 54, 56–62, 64, 69, 71, 72, 74–80, 83, 100, 101, 106)
process/object engine	The process and object execution engine of the Policy Support System. (60, 61, 63–66, 76)
security policy	A (information) security policy is typically a document that outlines requirements or rules that must be met by the system and its users in order to be secure. (2–5, 7, 11, 12, 23, 24, 26, 28, 31–34, 52, 53, 55–57, 59–61, 65, 67–80, 99, 105)
smart card	A (plastic) card with an embedded integrated circuit. (2, 6, 7, 12, 67, 68)
Vaadin	Vaadin is a Java framework for building modern web-applications. (59, 63)

Acronyms

API	Application Programming Interface (61, 63, 64, 66, 79)
BAR	Business Archive (57)
BEDL	Business Entity Definition Language (19, 23)
BPM	Business Process Management (xiii, 3–5, 8–10, 32, 33, 52, 55, 59, 61, 77, 78, 105)
BPMN	Business Process Model and Notation (xiii, 4, 8, 9, 16, 18–20, 23, 27, 31, 33, 42–47, 55–59, 61, 63, 64, 70–73, 76–79, 87, 88, 92, 105)
BPMS	Business Process Management System (xiii, 10, 57, 58)
CA	Certification Authority (6–8, 68, 69)
CISSP	Certified Information Systems Security Professional (99)
Corepro	Configuration Based Release Processes (xiii, 16, 17)
CP	Certificate Policy (7, 8)
CPS	Certification Practice Statement (8)
CRUD	Create, Read, Update and Delete (31)
DI	Diagram Interchange (57)
DSL	Domain Specific Language (33, 34, 76)
EMF	Eclipse Modeling Framework (105)
EPC	Event–Driven Process Chain (8)
ERCA	European Root Certification Authority (68, 69)
ERM	Entity Relationship Model (10)
FSM	Finite State machine (10, 11, 17, 39)
GUI	Graphical User Interface (61, 65, 66, 76, 80)
HSM	Hardware Security Module (12, 27, 69, 82)
KPI	Key Performance Indicator (10)
MOF	Meta Object Facility (34)
MSA	Member State Authority (68)

MSCA	Member State Certification Authority (68, 69, 73)
NIST	National Institute of Standards and Technology (xiii, 30)
OCL	Object Constraint Language (20, 21, 23, 34, 51, 78)
OMG	Object Management Group (9, 34, 35)
PK	Public-Key (5, 6)
PKI	Public-Key Infrastructure (2, 6, 7, 12, 68, 85)
PVM	Process Virtual Machine (63)
REST	Representational State Transfer (59, 64)
SK	Private (Secret)-Key (5, 6)
SOML	Security Object Modeling Language (xiii, 4, 25, 33-37, 42, 43, 46, 47, 51, 55-61, 63, 64, 69-71, 73-80, 96)
UML	Unified Modeling Language (xiii, 4, 8, 10, 11, 18, 20, 21, 23, 33-37, 39, 40, 55, 57, 63, 76-79, 105)
WS-BPEL	WS-Business Process Execution Language (8, 9)
XMI	XML Metadata Interchange (36, 57)
XML	Extensible Markup Language (19, 23, 78)
YAWL	Yet Another Workflow Language (8)

Bibliography

- [Act] Activiti. The Activiti BPM Platform. <http://www.activiti.org/>. Accessed: 2013-05-13.
- [Alt01] Steven Alter. *Information systems: foundation of e-business*. Prentice Hall PTR, 2001.
- [And01] R. Anderson. Why Information Security is Hard – An Economic Perspective. In *Proceedings of the 17th Annual Computer Security Applications Conference, ACSAC '01*, Washington, DC, USA, 2001. IEEE Computer Society.
- [BBB⁺12] Elaine Barker, William Barker, William Burr, William Polk, and Miles Smid. Recommendation for Key Management Part 1: General (Revision 3), 2012.
- [BDL03] David Basin, Jürgen Doser, and Torsten Lodderstedt. Model driven security for process-oriented systems. In *Proceedings of the eighth ACM symposium on Access control models and technologies*, pages 100–109. ACM, 2003.
- [BGH⁺07] Kamal Bhattacharya, Cagdas Gerede, Richard Hull, Rong Liu, and Jianwen Su. Towards formal analysis of artifact-centric business process models. In *Business Process Management*, pages 288–304. Springer, 2007.
- [BH07] James Bruck and Kenn Hussey. Customizing UML: Which technique is right for you. *White paper, Eclipse UML Project*, page 47, 2007.
- [BHW06] Pauline Bowen, Joan Hash, and Mark Wilson. *Information Security Handbook: A Guide for Managers*, 2006.
- [Bla] Paul E. Black. "finite state machine", in Dictionary of Algorithms and Data Structures [online], U.S. National Institute of Standards and Technology. <http://www.nist.gov/dads/HTML/finiteStateMachine.html>. Accessed: 2013-05-13.
- [BN09] J.W. Bishop and J-P Nordvik. Digital Tachograph System European Root Policy Version 2.1. <http://dtc.jrc.ec.europa.eu/>, 2009. Accessed: 2013-05-13.
- [BP83] James E Bailey and Sammy W Pearson. Development of a tool for measuring and analyzing computer user satisfaction. *Management science*, 29(5):530–545, 1983.
- [BVF07] Tom Baeyens and Miguel Valdes Faura. The Process Virtual Machine. <http://docs.jboss.com/jbpm/pvm/article/>, 2007. Accessed: 2013-05-13.

- [CFS⁺03] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. RFC 3647 (Informational), 2003.
- [Che76] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, 1(1):9–36, 1976.
- [Dav93] Thomas H. Davenport. *Process innovation: reengineering work through information technology*. Harvard Business School Press, Boston, MA, USA, 1993.
- [DDLS01] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The ponder policy specification language. In *Policies for Distributed Systems and Networks*, pages 18–38. Springer, 2001.
- [DLRMR13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. *Fundamentals of Business Process Management*. Springer, 2013.
- [EQST13] Montserrat Estañol, Anna Queralt, Maria Ribera Sancho, and Ernest Teniente. Artifact-Centric Business Process Models in UML. In *Business Process Management Workshops*, pages 292–303. Springer, 2013.
- [ERC04] ERCA. Digital Tachograph System European Root Certification Authority Certification Practices Statement. <http://dte.jrc.ec.europa.eu/>, 2004. Accessed: 2013-05-13.
- [Erl08] Thomas Erl. *Soa: principles of service design*, volume 1. Prentice Hall Upper Saddle River, 2008.
- [Fou12] The YAWL Foundation. YAWL – User Manual Version 2.3, 2012.
- [Fow03] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison-Wesley, Boston, first edition, 2003.
- [G⁺11] Marko Grönroos et al. *Book of Vaadin*. Vaadin Limited, 2011.
- [GBJ02] Martin Glinz, Stefan Berner, and Stefan Joos. Object-oriented modeling with ADORA. *Information Systems*, 27(6):425–444, 2002.
- [GJSB05] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java (TM) Language Specification, The (Java (Addison-Wesley))*. Addison-Wesley Professional, 2005.
- [GL02] Lawrence A Gordon and Martin P Loeb. The economics of information security investment. *ACM Transactions on Information and System Security (TISSEC)*, 5(4):438–457, 2002.
- [Gli07] Martin Glinz. On non-functional requirements. In *Requirements Engineering Conference, 2007. RE'07. 15th IEEE International*, pages 21–26. IEEE, 2007.
- [GRC04] Matteo Golfarelli, Stefano Rizzi, and Iuris Cella. Beyond data warehousing: what's next in business intelligence? In *Proceedings of the 7th ACM international workshop on Data warehousing and OLAP*, pages 1–6. ACM, 2004.
- [GZ05] Vincenzo Gervasi and Didar Zowghi. Reasoning about inconsistencies in natural language requirements. *ACM Trans. Softw. Eng. Methodol.*, 14(3):277–330, July 2005.

- [Har87] David Harel. Statecharts: A visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
- [IBM] IBM. IBM Business Process Manager. <http://www-03.ibm.com/software/products/us/en/business-process-manager-family/>. Accessed: 2013-05-13.
- [KN92] G Keller and M Nüttgens. AW scheer. semantische Prozessmodellierung auf der Grundlage Ereignisgesteuerter Processketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, 1992.
- [KR94] Haim Kilov and James Ross. *Information modeling: an object-oriented approach*. PTR Prentice Hall, 1994.
- [Kru95] Philippe Kruchten. The 4+1 View Model of Architecture. *IEEE Software*, 12(6):42–50, 1995.
- [KS91] Gerti Kappel and Michael Schrefl. Object/behavior diagrams. In *Data Engineering, 1991. Proceedings. Seventh International Conference on*, pages 530–539. IEEE, 1991.
- [Küs04] Jochen M Küster. Consistency management of object-oriented behavioral models, 2004.
- [KWB03] Anneke G Kleppe, Jos B Warmer, and Wim Bast. *MDA explained, the model driven architecture: Practice and promise*. Addison-Wesley Professional, 2003.
- [LBD02] Torsten Lodderstedt, David Basin, and Jürgen Doser. SecureUML: A UML-based modeling language for model-driven security. In *UML 2002—The Unified Modeling Language*, pages 426–441. Springer, 2002.
- [LIC03] Paul Legris, John Ingham, and Pierre Colletette. Why do people use information technology? A critical review of the technology acceptance model. *Information & management*, 40(3):191–204, 2003.
- [Mar83] J. Martin. *Managing the data-base environment*. The James Martin books on computer systems and telecommunications. Prentice-Hall, 1983.
- [MD01] Guel Michele D. A Short Primer for Developing Security Policies. http://www.sans.org/resources/policies/Policy_Primer.pdf, 2001. Accessed: 2013-05-13.
- [Moo56] Edward F Moore. Gedanken-experiments on sequential machines. *Automata studies*, 34:129–153, 1956.
- [Mül09] Dominic Müller. *Management datengetriebener Prozessstrukturen*. PhD thesis, Ulm University, 2009.
- [MVOV10] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 2010.
- [MvR10] Rolf M. von Roessing. *The Business Model for Information Security*. ISACA, 2010.
- [NC03] Anil Nigam and Nathan S Caswell. Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445, 2003.

- [NKF94] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *Software Engineering, IEEE Transactions on*, 20(10):760–773, 1994.
- [NKM⁺10] Prabir Nandi, Dieter Koenig, Simon Moser, Richard Hull, Vlad Klicnik, Shane Claussen, Matthias Kloppmann, and John Vergo. Data4BPM, Part 1: Introducing Business Entities and the Business Entity Definition Language (BEDL). http://www.ibm.com/developerworks/websphere/library/techarticles/1004_nandi/1004_nandi.html, April 2010. Accessed: 2013-05-29.
- [NKW12] Prabir Nandi, Vlad Klicnik, and Steve White. Data4BPM, Part 3: Modeling processes with business entities using extended BPMN. http://www.ibm.com/developerworks/bpm/library/techarticles/1210_nandi/1210_nandi.html, October 2012. Accessed: 2013-05-29.
- [NPK⁺11] Prabir Nandi, Florian Pinel, Dieter Koenig, Simon Moser, and Richard Hull. Data4BPM, Part 2: BPEL4Data: Binding WS-BPEL to Business Entity Definition Language (BEDL). http://www.ibm.com/developerworks/websphere/library/techarticles/1105_nandi/1105_nandi.html, April 2011. Accessed: 2013-05-29.
- [OAS07] OASIS. Web Services Business Process Execution Language Version 2.0, April 2007.
- [OMG03] OMG. *Model Driven Architecture (MDA) Guide*, 2003.
- [OMG11a] OMG. Business Process Modeling Notation (BPMN) Version 2.0, 2011.
- [OMG11b] OMG. Meta Object Facility (MOF) Core (Version 2.4.1), 2011.
- [OMG11c] OMG. MOF 2 XMI Mapping, 2011.
- [OMG11d] OMG. OMG Unified Modeling Language (OMG UML), Infrastructure Specification (Version 2.4.1), 2011.
- [OMG11e] OMG. OMG Unified Modeling Language (OMG UML), Superstructure Specification (Version 2.4.1), 2011.
- [OMG12] OMG. OCL 2.3.1 Specification, 2012.
- [PQ95] Terence J. Parr and Russell W. Quong. ANTLR: A predicated-LL (k) parser generator. *Software: Practice and Experience*, 25(7):789–810, 1995.
- [PSM07] S. Pahlila, M. Siponen, and A. Mahmood. Employees' Behavior towards IS Security Policy Compliance. In *System Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on*, pages 156b–156b, 2007.
- [Rad12] Tijs Rademakers. *Activiti in Action : Executable business processes in BPMN 2.0*. Manning Publications, Shelter Island, NY, first edition, 2012.
- [RBP⁺91] James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, and William Lorenson. *Object-oriented modeling and design*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.
- [RTHEvdA05] Nick Russell, Arthur HM Ter Hofstede, David Edmond, and Wil MP van der Aalst. Workflow data patterns: Identification, representation and tool support. In *Conceptual Modeling-ER 2005*, pages 353–368. Springer, 2005.

- [RvdATHW06] Nick Russell, Wil MP van der Aalst, Arthur HM Ter Hofstede, and Petia Wohed. On the suitability of UML 2.0 activity diagrams for business process modeling. In *Proceedings of the 3rd Asia-Pacific conference on Conceptual modelling-Volume 53*, pages 95–104. Australian Computer Society, Inc., 2006.
- [Shi00] Robert Shirey. Rfc 2828: Internet security glossary. *The Internet Society*, 2000.
- [SL02] Morris Sloman and Emil Lupu. Security and management policy specification. *Network, IEEE*, 16(2):10–19, 2002.
- [SS02] Michael Schrefl and Markus Stumptner. Behavior-consistent specialization of object life cycles. *ACM Trans. Softw. Eng. Methodol.*, 11(1):92–148, January 2002.
- [SS08] Adam Shostack and Andrew Stewart. *The new school of information security*. Addison-Wesley Professional, first edition, 2008.
- [SSV98] Ian Sommerville, Pete Sawyer, and Stephen Viller. Viewpoints for requirements elicitation: a practical approach. In *Requirements Engineering, 1998. Proceedings. 1998 Third International Conference on*, pages 74–81. IEEE, 1998.
- [TD92] Arthur R Tenner and Irving J DeToro. *Total quality management: Three steps to continuous improvement*. Addison-Wesley Reading, MA, 1992.
- [VDABEW01] Wil MP Van Der Aalst, Paulo Barthelmeß, Clarence A Ellis, and Jacques Wainer. Proclats: A framework for lightweight interacting workflow processes. *International Journal of Cooperative Information Systems*, 10(04):443–481, 2001.
- [vdAtH05] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Inf. Syst.*, 30(4):245–275, June 2005.
- [VdB94] Michael Von der Beeck. A comparison of statecharts variants. In *Formal techniques in real-time and fault-tolerant systems*, pages 128–148. Springer, 1994.
- [VDS05] Ragnhild Van Der Straeten. *Inconsistency Management in Model-Driven Engineering*. PhD thesis, PhD thesis, Vrije Universiteit Brussel, 2005.
- [W3C10] W3C. XQuery 1.0: An XML Query Language (Second Edition), 2010.
- [W3C12a] W3C. XML Path Language (XPath) 2.0 (Second Edition), 2012.
- [W3C12b] W3C. XML Schema 1.1 Datatypes & Structures, 2012.
- [Wah09] KSenia Wahler. *A framework for integrated process and object life cycle modeling*. PhD thesis, University of Zurich, 2009.
- [Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.
- [ZM04] Michael Zur Muehlen. *Workflow-based process controlling: foundation, design, and application of workflow-driven process information systems*, volume 6. Michael zur Muehlen, 2004.