

Deliverable AP 5.1

Requirements Analysis

Requirements for Specification of Security Policies

21.02.2007

Ute Faltin, Fraunhofer SIT
Mathias Kohler, SAP AG
Stefan Kowski, Parks Informatik GmbH
Ruben Wolf, Fraunhofer SIT

ORKA is funded by the German Ministry of Education and Research (BMBF) as part of its Software Engineering 2006 programme.

© 2006 ORKA Consortium



Federal Ministry
of Education
and Research

Internal document information:

\$Id: del-ap5.1.tex,v 1.48 2007/03/14 18:26:36 kowski Exp \$

Contents

- List of Requirements** **5**

- 1 Introduction** **9**
 - 1.1 Requirements 9
 - 1.2 Security Goals and Requirements 10
 - 1.3 Policies 10
 - 1.4 General Overview about Policy Management Tools 11
 - 1.4.1 Policy Life-Cycle and Policy Management 11
 - 1.4.2 Interface to Context Provider 12
 - 1.5 The Ponder Toolkit as Example 12

- 2 Requirements Engineering Process** **14**
 - 2.1 Requirements engineering process description 14

- 3 ORKA Security Goals** **15**

- 4 Case Study Requirements** **16**

- 5 Policy Administration Model Requirements** **18**
 - 5.1 Decentralized and Distributed Administration 19
 - 5.2 Separation of Organization and Administration 20
 - 5.3 Traceability and Auditing 21
 - 5.4 Policy Representation 23
 - 5.5 Enterprise Integration 24
 - 5.6 Policy Engineering 25
 - 5.7 Usability 26

- 6 Policy Management Requirements** **29**
 - 6.1 Working with Policies 30
 - 6.1.1 Create a policy 30
 - 6.1.2 Edit a policy 30
 - 6.1.3 Validate a policy 31
 - 6.1.4 Preview policy changes 31
 - 6.1.5 Release a policy 32
 - 6.1.6 Delete a policy 32
 - 6.2 Working with Users 33
 - 6.2.1 Create a user 33
 - 6.2.2 Edit a user 33
 - 6.2.3 Delete a user 34
 - 6.2.4 Assign a role 34
 - 6.2.5 Remove a role 35
 - 6.2.6 Show effective roles 35
 - 6.2.7 Show effective permissions 36
 - 6.2.8 Show administrative permissions 36
 - 6.3 Working with Roles 37
 - 6.3.1 Create a role 37

6.3.2	Edit a role	37
6.3.3	Delete a role	38
6.3.4	Assign a user	38
6.3.5	Remove a user	39
6.3.6	Show effective role users	39
6.3.7	Show effective role permissions	40
6.3.8	Create role inheritance	40
6.3.9	Delete role inheritance	41
6.4	Working with workflows	42
6.4.1	Assign role to workflow task	42
6.4.2	Remove role from workflow task	42
6.5	Audit	43
6.5.1	Show audit protocol	43
6.5.2	Search audit protocol by event	43
6.5.3	Search audit protocol by user	44
6.5.4	Search audit protocol by role	44
6.5.5	Search audit protocol by policy	45
7	Requirements for Constraint Specification in Workflows	45
7.1	Introduction	45
7.2	Business Process Specification	46
7.3	Workflow specific Requirements for constraint specification	46
8	Conclusion	49

List of Requirements

- AR4.1 Case Study: Access only for Authorized Users (Mandatory) 16
- AR4.2 Case Study: Auditing Measures (Mandatory) 16
- AR4.3 Case Study: Domain Responsibility (Mandatory) 16
- AR4.4 Case Study: Separation of Duties for Policy Administration (Optional) 17
- AR5.1 Decentralization: Decentralized Administration (Mandatory) 19
- AR5.2 Decentralization: Rights of Domain Administrators (Mandatory) 19
- AR5.3 Decentralization: Policy Splitting (Optional) 19
- AR5.4 Decentralization: Composite Policies (Optional) 19
- AR5.5 Decentralization: Check Consistency in Decentralized Environments (Mandatory) 19
- AR5.6 Decentralization: Check Policy Rules (Mandatory) 19
- AR5.7 Decentralization: Check against Enterprise Security Principles (Mandatory) . . 19
- AR5.8 Decentralization: Simultaneous Administration (Optional) 19
- AR5.9 Separation: Efficient Policy Engineering (Optional) 20
- AR5.10 Separation: Authorization of Administrators (Mandatory) 20
- AR5.11 Separation: Separation of Administrative Tasks (Mandatory) 20
- AR5.12 Separation: Domain-specific Administration (Mandatory) 21
- AR5.13 Separation: Notification of Actually Access Rights (Optional) 21
- AR5.14 Separation: Execute Administrative Operations (Mandatory) 21
- AR5.15 Reference Monitor: Controlling Administrative Operations (Mandatory) 21
- AR5.16 Reference Monitor: Authorization of Administrative Operations (Mandatory) . 21
- AR5.17 Compliance: Support for Compliance Rules (Optional) 22
- AR5.18 Compliance: Support for Static Constraints (Optional) 22
- AR5.19 Supervision: Actually Relevant Policy (Mandatory) 22
- AR5.20 Supervision: Controlling Policy Modifications (Mandatory) 22
- AR5.21 Supervision: Specification of Logging Rules (Optional) 22
- AR5.22 Supervision: Specify Security-Relevant Events (Optional) 22
- AR5.23 Supervision: Monitor Policy Check Results (Mandatory) 22
- AR5.24 Supervision: Monitoring Events with Time and Causer (Mandatory) 22
- AR5.25 Supervision: Monitoring Administrative Operations (Mandatory) 22
- AR5.26 Supervision: Controlling Administrative Access Rights (Mandatory) 22

AR5.27	Supervision: Controlling Identification and Authentication (Mandatory)	23
AR5.28	Supervision: Human Controlling of Selected Security-relevant Events (Optional)	23
AR5.29	Supervision: Human Controlling of Log Information (Optional)	23
AR5.30	Supervision: Automatic Controlling of Log Information (Optional)	23
AR5.31	Supervision: Log Protection (Mandatory)	23
AR5.32	Representation: Expressiveness of Policy Language (Optional)	23
AR5.33	Representation: Actual Snapshot (Optional)	23
AR5.34	Representation: Possible Actions (Mandatory)	23
AR5.35	Representation: Forbidden Actions (Mandatory)	24
AR5.36	Representation: Preview (Mandatory)	24
AR5.37	Representation: Various Kinds of Illustrations (Optional)	24
AR5.38	History: Policy Versions (Optional)	24
AR5.39	History: Policy Comments (Optional)	24
AR5.40	Representation: Context Information (Optional)	24
AR5.41	Integration: Transform Policy Representation (Mandatory)	24
AR5.42	Integration: Modular and Flexible Architecture (Mandatory)	24
AR5.43	Integration: Modular wrt. Validation and Enforcement (Mandatory)	25
AR5.44	Integration: Distributed Components – Secure Communication (Mandatory) . .	25
AR5.45	Engineering: Round-Trip-Policy-Administration (Optional)	25
AR5.46	Engineering: Definition, Maintenance, and Query using GUIs (Optional) . . .	26
AR5.47	Engineering: Specification of Policy Statements (Mandatory)	26
AR5.48	Engineering: Support Context Information (Mandatory)	26
AR5.49	Engineering: Execute Consistency Checks (Mandatory)	26
AR5.50	Engineering: Execute Completeness Checks (Mandatory)	26
AR5.51	Engineering: Execute Security Property Checks (Mandatory)	26
AR5.52	Usability: Context-dependent Access to Operations (Optional)	27
AR5.53	Usability: User Interface Design (Optional)	27
AR5.54	Usability: Feedback After Processing (Mandatory)	27
AR5.55	Usability: Ease of Use (Optional)	27
AR5.56	Usability: System Documentation, Online Help System (Optional)	27
AR5.57	Usability: Provide an Undo Function (Optional)	28
AR5.58	Usability: Support for Cognitive Models (Optional)	28

AR5.59 Usability: Prevent Dangerous Errors (Optional)	28
AR5.60 Usability: Visualize the State of the System (Mandatory)	28
AR5.61 Usability: Support for What-If Scenarios (Optional)	28
AR5.62 Usability: Inform About Potentially Unwanted Consequences (Optional)	28
AR5.63 Usability: Support for Various Views (Optional)	28
AR5.64 Usability: Machine-readable and -writable Policies (Mandatory)	29
AR5.65 Usability: Human Writability and Readability (Controllability) (Optional)	29
AR5.66 Usability: Support for Comments (Optional)	29
AR5.67 Usability: Support for Macros (Optional)	29
AR5.68 Usability: Maintainability (Optional)	29
AR6.1 Use Case: Create a policy (Mandatory)	30
AR6.2 Use Case: Edit a policy (Mandatory)	30
AR6.3 Use Case: Validate a policy (Mandatory)	31
AR6.4 Use Case: Preview policy changes (Optional)	31
AR6.5 Use Case: Release a policy (Mandatory)	32
AR6.6 Use Case: Delete a policy (Mandatory)	32
AR6.7 Use Case: Create a user (Mandatory)	33
AR6.8 Use Case: Edit a user (Mandatory)	33
AR6.9 Use Case: Delete a user (Mandatory)	34
AR6.10 Use Case: Assign a role (Mandatory)	34
AR6.11 Use Case: Remove a role (Mandatory)	35
AR6.12 Use Case: Show effective roles (Optional)	35
AR6.13 Use Case: Show effective permissions (Optional)	36
AR6.14 Use Case: Show administrative permissions (Optional)	36
AR6.15 Use Case: Create a role (Mandatory)	37
AR6.16 Use Case: Edit a role (Mandatory)	37
AR6.17 Use Case: Delete a role (Mandatory)	38
AR6.18 Use Case: Assign a user (Mandatory)	38
AR6.19 Use Case: Remove a user (Mandatory)	39
AR6.20 Use Case: Show effective role users (Optional)	39
AR6.21 Use Case: Show effective role permissions (Optional)	40
AR6.22 Use Case: Create role inheritance (Mandatory)	40

AR6.23	Use Case: Delete role inheritance (Mandatory)	41
AR6.24	Use Case: Assign role to workflow task (Mandatory)	42
AR6.25	Use Case: Remove role from workflow task (Mandatory)	42
AR6.26	Use Case: Show audit protocol (Mandatory)	43
AR6.27	Use Case: Search audit protocol by event (Optional)	43
AR6.28	Use Case: Search audit protocol by user (Optional)	44
AR6.29	Use Case: Search audit protocol by role (Optional)	44
AR6.30	Use Case: Search audit protocol by policy (Optional)	45
AR7.1	Workflows: Read in of Workflow Definition (Mandatory)	47
AR7.2	Workflows: Unique Workflow Identification (Mandatory)	47
AR7.3	Workflows: Interpretation of Workflow Context Information (Optional)	47
AR7.4	Workflows: Task Extraction (Mandatory)	47
AR7.5	Workflows: Task Identification (Mandatory)	47
AR7.6	Workflows: Interpretation of Task Context Information (Optional)	47
AR7.7	Workflows: Identification of linear Order of Tasks (Mandatory)	47
AR7.8	Workflows: Basic Workflow Pattern Recognition (Optional)	47
AR7.9	Workflows: Advanced Workflow Pattern Recognition (Optional)	48
AR7.10	Workflows: Identification of non-linear Order of Tasks (Optional)	48
AR7.11	Workflows: Distinction btw. Manual and Automatic Tasks (Optional)	48
AR7.12	Workflows: Composite Tasks (Optional)	48
AR7.13	Workflows: Process Dataflow and/or related Data Objects Interpretation (Optional)	48
AR7.14	Workflows: Distinction of Different Versions of Workflow Specifications (Optional)	49
AR7.15	Workflows: Independent Constraint Assignment (Optional)	49

1 Introduction

The ORKA project develops a security management system supporting roles and workflows. A unified security policy is being used to control access to sensible resources, such as personal or financial data.

Being implemented using open middleware technologies, ORKA can be used from various service-oriented applications.

The subject matter of this paper is to analyze and specify the requirements needed to specify, validate and implement ORKA security policies.

1.1 Requirements

Each system to be developed and used has different types of stakeholders, that want to make sure that it meets their specific intentions. In requirements analysis, the stakeholder's intentions are to be formalized in a requirements document. Each requirement describes the system's operations and behavior under specific circumstances.

Requirements describe *what* has to be done to meet the objectives of the stakeholders. They do not describe *how* the system will be implemented, this will be specified in the system design.

Requirements can be subdivided into functional and non-functional requirements (see [20]). Requirements never specify *what* has to be done during design or implementation.

Functional requirements describe what has to be done by the system to fulfill the business goals. They include the input of an operation, the detailed processing to be performed, perhaps including interaction with a user or other hardware or software, and the resulting output.

Non-functional requirements impose constraints on the design or implementation of the system. They describe the overall operation of the system, not specific behaviors. Some examples of non-functional requirements are:

- performance
- reliability
- availability
- scalability
- usability
- security

Mainly security requirements restrict the functionality of the system and therefore bar the users from doing any operation they want. But on the other hand, they enhance other non-functional requirements. Usability enhancements can be achieved by disabling menu items if the user is not allowed to perform the operation, thus preventing security error messages to pop up. Better

data validation to prevent e.g. SQL injection enhances the reliability of the system as a whole, resulting in positive user experiences.

1.2 Security Goals and Requirements

Security requirements can be derived from risk analysis. Risk analysis describes the assets of the systems that could result in harm if the system is misused.

The assets to be taken into account depend on the business goals, because these goals define the organizational assets to be processed by the system.

The type of harm that can happen depends on the asset type. Main harms are financial loss or information exposure, loss or corruption.

To prevent harm, the security goals of the system have to be defined. They are derived from the possible harm to assets, management control principles or application business goals. The business goals will determine which management control principles are applicable. The management control principles can be extended by external security standards (e.g. SOX or privacy regulations) if the system handles the appropriate type of information.

Some examples of security goals are (see [12]):

- Ensure that confidential communications and data are kept private
- Ensure that communications and data are not intentionally corrupted.
- Enable security personnel to audit the status and usage of the security mechanisms.

Other security goals can be found in section 3 *ORKA Security Goals*. A discussion of security goals and security requirements can be found in [15].

Security requirements are being derived from the security goals and will be applied as a constraint to the functional requirement.

1.3 Policies

A policy can be defined as a set of goals or rules specifying a system behavior in a domain to achieve the desired objectives [18]. ORKA defines a security policy according to this definition.

The policy can be specified as a document containing formal statements or as an informal, unstructured document. A formally defined policy is machine understandable and can be automatically enforced into a system.

Because of its formal representation, administrators may have difficulties to understand the formally defined policies. On the contrary, administrators should have no difficulties understanding policies in an informal representation. But problems may arise in the informal representation, such as undetected policy conflicts, imprecisely defined policies, etc. So an ORKA project goal is to define formal policies, but making them readable to human users.

The policy must always be consistent and correct. Formally, each modification of the policy is a state transition. In ORKA, we use constraints and obligations to maintain secure state transitions

of the policy. Each administrative operation modifying the policy changes the system and is therefore a state transition. So a policy engine must assure that all state transitions result in a secure, consistent state (see AR5.5, AR5.6, AR5.49, AR5.50).

1.4 General Overview about Policy Management Tools

The idea of a policy management tool is to provide the functionality to easily define and manage the security policies needed for a system. Policy management generally spoken begins with the definition of new policies, continues with making them appropriately available to the system and ends with maintaining existing policies. In other words, the tool is generally an interface for a security administrator to manage (access control) policies in all phases of a policy life-cycle.

1.4.1 Policy Life-Cycle and Policy Management

The policy life-cycle comprises the following four stages.

1. Policy Definition
2. Policy Storage
3. Policy Maintenance
4. Policy Removal from the system

Please see Figure 1 for illustration.

The first stage is the definition of a policy specification.

This is where a security policy is created¹. Ideally, this step should be supported by a policy management tool by acting as interface between administrator and access control management system and abstracting the complexity of the underlying policy specification process in such a way that the administrator can specify policies, without having to know, the details of a specific policy language used within the system. This phase can also include some kind of policy conflict analyzer or model checker as, for example, presented in [19].

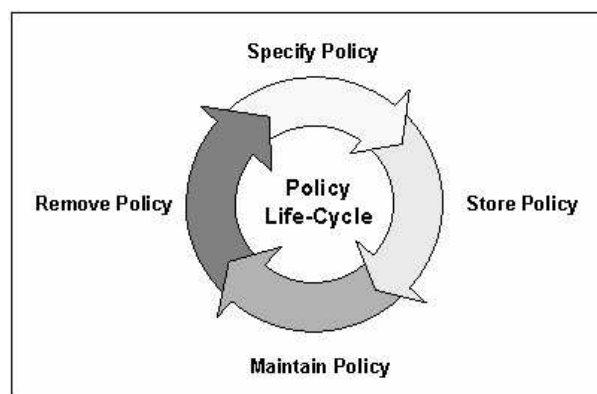


Figure 1: Policy Life Cycle

¹Alternatively policies from other systems may be imported (and converted).

The second stage is the storage of the defined policy in a policy repository. Furthermore, if needed, the policies can be translated into enforcement mechanisms (e.g. transforming the policies into access control policies for the Java platform). Examples are given in [10] and [21].

The third stage is the maintaining process for policies. This includes, for instance, setting a policy as enabled or disabled, which allows defining policies beforehand and only activating them if needed. Updating and editing stored policies belongs to this phase of the policy life-cycle.

With stage four the policy life-cycle ends by removing a policy from the system.

1.4.2 Interface to Context Provider

A security administrator needs to be able to administer security policies for a given system. Therefore, a policy management tool should cover the above mentioned stages of a policy life-cycle. However, for this requirement, the management tool needs to have access to several context providers to either get the information which is needed to define the policies or to provide the service for storing and maintaining them.

Three types of context provider can be identified for access control in workflow environments, namely: the workflow management system itself, a directory service, and a policy repository. (Please see Figure reffig:context-provider for illustration.)

The **workflow management system** provides context information about the business processes (and business objects in general) and their implemented tasks. Examples for workflow management systems are jBPM on JBOSS [13] or openWFE [17].

The **directory service** stores and provides the necessary information about subjects and their assigned roles to bind access control rules to them. An LDAP server is an example for such a context provider.

The **policy repository** stores the defined policies. The repository might also keep the information, which policies are activated or deactivated. Possible storage provider range from XML-files over relational databases [21] to directory services using LDAP [11].

In the following section, a management tool for the policy specification language Ponder is introduced as an example for a policy management tool.

1.5 The Ponder Toolkit as Example

In [10], Damianou et. al describe a toolkit for "the specification, deployment and management of policies specified in the Ponder language". This toolkit is developed for the policy specification language named Ponder, "a declarative, object oriented language for specifying security and management policies for distributed systems" [10]. The functionality of the tool, however, shows capabilities which reflect a general possible way of managing policies and is therefore introduced as example.

The Ponder-toolkit basically comprises the following three tools:

1. A policy editor,

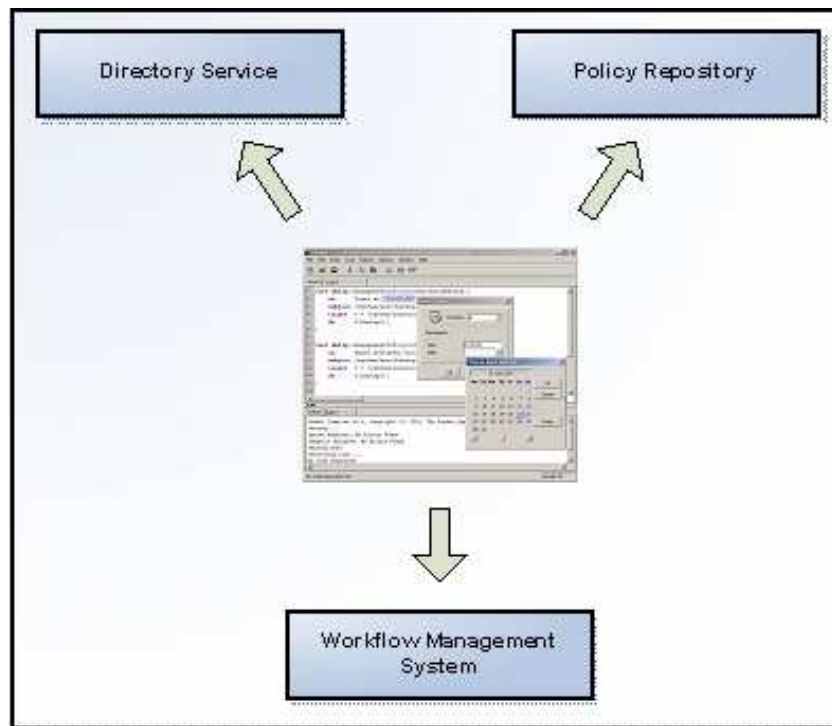


Figure 2: Context Provider

2. a policy management console, and
3. a user-role management tool.

The **policy editor** provides security administrators a development environment for specifying, reviewing and modifying policies. For creating new policies templates can be used. Already existing policies can be selected from the policy repository² and loaded into the policy editor. They can be modified and afterwards stored back to the repository.

A conflict analyzer integrated with the editor checks for conflicting policies within the system. It performs a static analysis of the policies and allows specifying precedence for single policies to resolve conflicts.

For policy enforcement, policies specified with Ponder language have to be transformed into enforcement mechanisms. This is supported by providing an integrated compiler framework with several modules such as syntax analyzer, semantic analyzer, code assembler, and exchangeable code generators. There are implementations for translating Ponder policies onto various access control platforms like Java, Windows 2000 or Linux.

Furthermore, with the editor also roles can be created. They are stored with the LDAP directory service.

With the **policy management console** policies can be selected from the repository. Policies then can be loaded, unloaded, and enabled or disabled.

²In the Ponder Management System Architecture policies are stored as "policy objects" in the so called "domain service" which is an extended implementation of an LDAP directory.

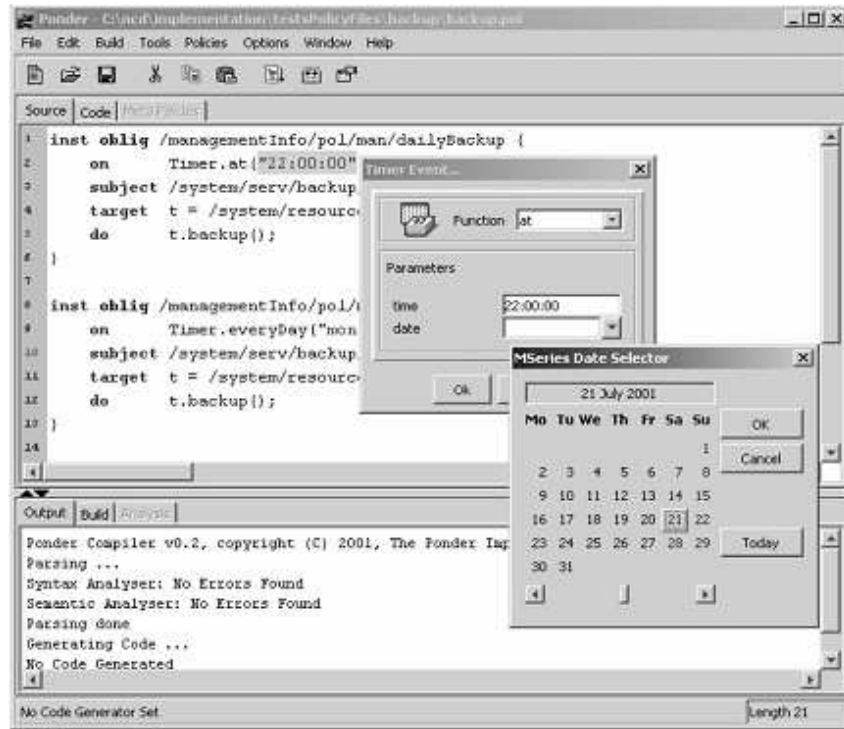


Figure 3: Ponder Policy Editor

With the **user-role management tool** users can be assigned and removed from roles. Also, roles can be activated and deactivated to implement the RBAC concept of sessions.

2 Requirements Engineering Process

Within this workpackage, we will take into account the requirements concerning the administrative function within the ORKA system. Maybe some of the requirements will also be found in the requirements analysis of the workpackages CONTROL, SPEC, ENFORCE and VALID. This workpackages will define other requirements not mentioned here as well. During ORKA system design, we will use the requirements from all workpackages.

2.1 Requirements engineering process description

First we will define the ORKA security goals, i.e. specific properties of the system that have to be met by the later project stages.

Other functional and non-functional requirements will be derived from the case studies developed in work package 1.1 and from use case definitions that specify the operations of the system.

The requirements will be specified within four sections:

- Case study requirements

- Policy administration model requirements
- Policy management requirements
- Requirements for constraint specification in workflows

Functional requirements will be defined with use cases. Within each use case, non-functional requirements will be defined as a constraint.

According to Bradner [8], requirements may be classified as MUST, SHOULD, MAY, MUST NOT, SHOULD NOT, or MAY requirements. Depending on their importance or possibility to be implemented within the project time frame, they will be classified as *Optional* or *Mandatory*.

Next we will define some criteria that can be used to evaluate administration models and software tools to be used during development.

After that, existing security administration models and software tools will be analyzed if they fulfill the requirements of the ORKA project. This will be done in work package 5.2.

Within work package 5.3, the ORKA security administration model will be developed based on the results of work package 5.2. The result may be a combination of features of existing administration models or a new security model that may be needed to reach the ORKA security goals.

3 ORKA Security Goals

This section describes security goals that have to be met by the ORKA security administration model and during design and implementation of the ORKA software system.

All requirements in the next chapters will be derived from the security goals.

Security goals:

- Ensure that the system is not accessible by unauthorized users
- Ensure that system data is protected against unauthorized modification or eavesdropping
- Ensure that the system policy model supports different administrators with different clearance levels

To support large user population, multiple administrators are required.

- Ensure that the system detects attempted intrusions by unauthorized users or applications
- Ensure that system policy or audit data is not intentionally corrupted

It must be sure that no faked audit events can be created and harm innocent users.

- Ensure that all interactions with the system can be audited

All actions must be auditable by the system, without the use of external, e.g. paper-based solutions.

- Ensure that parties working with the system cannot later repudiate those interactions
- Ensure compliance with relevant law, standards, and enterprise rules

4 Case Study Requirements

This section describes the requirements which can be extracted by the analysis of the windows oriented use case scenarios of work package 1.1 illustrated in the document 'Case Study 2' (see [1]).

These case studies mainly focused on scenarios in the windows domain with special emphasis on administration issues such as the management of access restrictions on resources or user/role management.

The administration tool for ORKA will be used to manage user/role associations and to specify, validate and implement ORKA security constraints. This means, for the administrative side, there are parallels to the case studies given in work package 1.1, and it is consequently possible to derive requirements for the ORKA administration tool - also concerning security measures for the administration tool itself.

The following list will give the identified requirements. We will cite the original source scenario or statement of the case study document and will show how the statement can be applied as requirement to the ORKA administration tool.

AR4.1 – Case Study: Access only for Authorized Users (Mandatory)

For the Windows environment holds: 'The security controls within a Windows network must in particular ensure that only permitted users gain access to functions and services [...] and that administrative operations can only be performed by permitted administrators (i.e. users possessing administrative roles). [1, page 4]'

The same must hold for the administration tool, so that *only permitted users MUST gain access to administrative functions and operations; i.e., they MUST only be performed by users possessing the adequate administrative roles.*

This requirement mainly speaks for itself. It ensures that only users which have the right to add, update, delete, or manage security constraints in general are have access to them. This must also hold for any user/role administration activities.

AR4.2 – Case Study: Auditing Measures (Mandatory)

For the Windows domain, there is the need if an access right change is requested 'appropriate means [must] ensure correct logging [1, page 8]'

Transferred to the ORKA administration tool, there is a need that *every administrative change of one or more access control constraints as well as user management activities MUST be recorded by appropriate auditing measures to log the execution of a certain action by a certain user/administrator.*

This is especially important for sufficient transparency and traceability of performed activities. The reader may also be referred to section 5.3 Logging/Auditing where this rather general requirement is subdivided into concrete auditing and logging requirements.

AR4.3 – Case Study: Domain Responsibility (Mandatory)

In the Windows domain, 'when a user accesses a resource (e.g., a directory or a file) and realizes he has no access to the file, he must request the responsible resource owner for access to this resource [1, page 8]'. This implies that every resource owner has the right

and the responsibility to manage his or her assigned resources and is further able to only give those users the right for access that need it.

For the ORKA administration tool this gives two requirements.

1. *Policy management MUST be divisible into specific domains so that every administrator can be assigned for a certain domain and be accounted for the activities within a certain domain.* For instance, there must be a measure to define the set of security constraints which belong to a certain domain.
2. *Administrators MUST only be able to perform changes to policies/security constraints in the domain for which they are assigned and consequently responsible.*

The two given requirements only combined make eventually sense and can be counted to measures to enforce the least privilege principle.

AR4.4 – Case Study: Separation of Duties for Policy Administration (Optional)

Important for Windows configuration settings are so called 'Group Policy Objects' (GPOs) which might have great impact on the overall Windows security. GPOs define security settings for a complete group of objects within the Active Directory and have therefore great impact on security definitions made by a security administrator.

Case study scenario 3.4 'GPO Management in Active Directory' of work package 1.1 (see [1, page 19]) describes due to the facts that 'Windows does not implement separation of duties in any way by default wrt. GPO management: any enterprise administrator can perform all operations [on security relevant settings]' within the Active Directory. This offers great potential for either accidental or intentional maliciously performed actions since administrators have consequently 'full control on the Active Directory [and] cannot be prevented from implementing and deploying 'malicious' settings. [...] In most cases such modifications will not be noticed at all.'

For control enhancement, the case study analysis concludes by proposing to introduce separation of duties for the GPO administration by implementing a mandatory approval of the changes made by a different administrator. Changes are only made if the approval is successful (please see Figure 4 for illustration, taken from [1]).

The policies defined with the ORKA administration tool certainly have impact on the security behavior of a secured object such as a business process. The derived requirement for the ORKA administration tool is *the administration tool SHOULD support the four-eyes-principle in such a way that always two persons have to be involved in the process of performing and approving changes to the policy repository.*

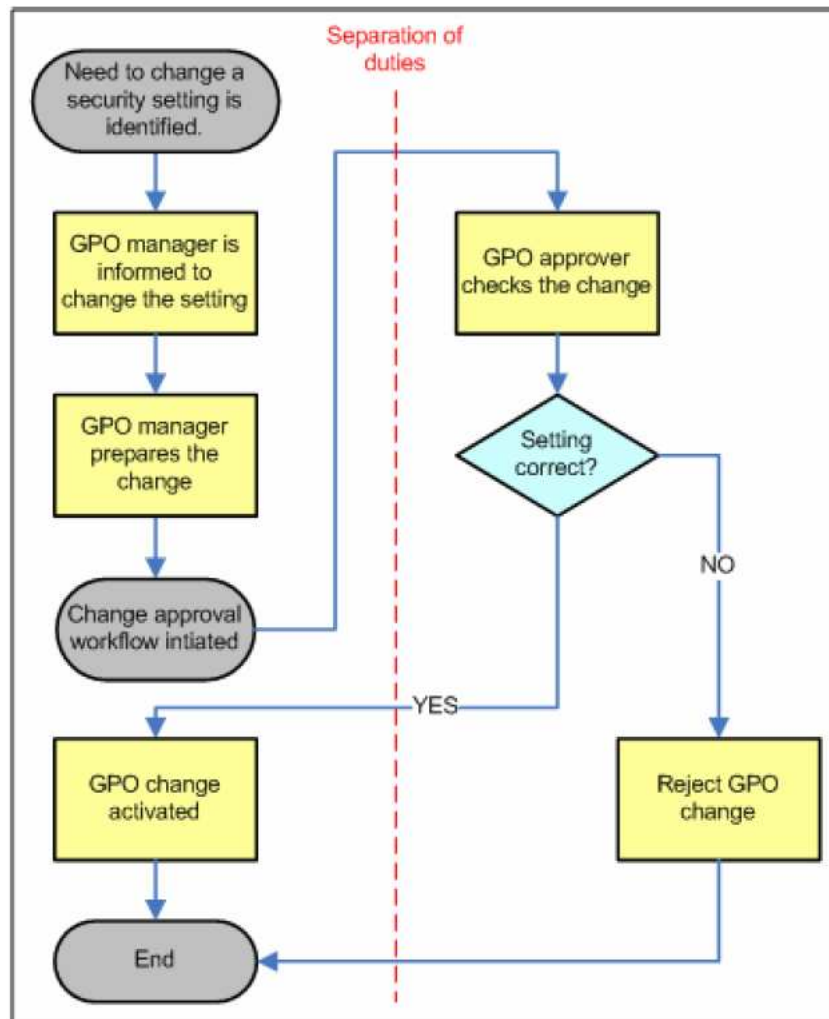


Figure 4: Separation of Duties for GPO administration (taken from [1])

This section presented the results of the analysis of the use case studies for the windows domain of work package 1.1. The above listed items are not be considered as of exhaustive with respect to a complete list of requirements for the administration tool. It rather has the intention to extract important requirements from the given case studies to build a starting point for further requirement specifications within the following sections.

5 Policy Administration Model Requirements

Policies are being increasingly used for controlling the behavior of complex systems and automated system management. This section describes the non-functional requirements the ORKA policy administration model and its system integration should met.

5.1 Decentralized and Distributed Administration

With respect to the rise of internet-based systems used by large enterprises that have to integrate inter-organizational networks and internet-based services, the task of managing such distributed systems has become very challenging. The following requirements are intended to ease and secure the management of policies for administrators in decentralized and distributed environments.

AR5.1 – Decentralization: Decentralized Administration (Mandatory)

The ORKA architecture **MUST** allow decentralized administration. A decentralized architecture is required to provide flexibility and manageability in complex and distributed systems.

AR5.2 – Decentralization: Rights of Domain Administrators (Mandatory)

Administrators and security administrators provided with different administration rights **MUST** act in different domains to prevent misuse of access rights. No administrator should be permitted to change its own access rights. This requirement meets the security goals confidentiality and integrity and prevents infiltration. It also meets the case study requirement AR4.3 *Domain Responsibility*.

AR5.3 – Decentralization: Policy Splitting (Optional)

It **SHOULD** be possible to split a global policy into multiple parts stored in different data containers (e.g. files, database tables). This requirement enlarges flexibility and manageability of policy administration.

AR5.4 – Decentralization: Composite Policies (Optional)

The administration model **SHOULD** support composite policies generated by distributed administration. An automatic component should be capable to joint together policy parts. This requirement simplifies the administration of policies.

AR5.5 – Decentralization: Check Consistency in Decentralized Environments (Mandatory)

The administration interface **MUST** be capable to provide consistency checks, i.e. checking whether a composite policy or its parts contain inherent contradictions. This requirement prevents inconsistencies caused by decentralized and distributed administration.

AR5.6 – Decentralization: Check Policy Rules (Mandatory)

It **MUST** be possible to verify the consistency of all policy rules within a policy. This requirement simplifies the administration of policies and prevents inconsistencies.

AR5.7 – Decentralization: Check against Enterprise Security Principles (Mandatory)

It **MUST** be possible to check a composite policy or its parts against given enterprise security principles. This requirement meets the security goal compliance.

AR5.8 – Decentralization: Simultaneous Administration (Optional)

The administration model **SHOULD** support simultaneous administration. This requirement is a special case of distributed administration.

5.2 Separation of Organization and Administration

Each enterprise is divided into more critical and more public domains. Often security goals like protection of sensitive data compete with the need of information sharing and availability to meet the business goals. Separation of organization and administration makes it more easier to manage large user data in an efficient manner and to protect more critical administrative acts. It makes it more easier to translate organizational structures into role hierarchies and to specify adequate security policy rules.

Organization Organization means the assignment and revocation of users to roles. In some models users are grouped in teams according to the organizational structure of an enterprise. In this case members of a team are assigned to and revoked from roles. Assignment and revocation is an administrative function that must be performed whenever a person is hired or leaving an organization or will change its job position. According to the organizational structure and enterprise goals a member off staff will change its job position and project assignment pretty much frequently. In comparison to administration, organization demands more often administrative interventions.

Administration Administration means the specification of security policies and their transformation into policy languages. It means the assignment of permissions to roles and the design and transformation of role hierarchies. In addition, it means the modification and revocation of policies, assignments, and role hierarchies by administrative interventions. In comparison to organization, administration demands more rare administrative interventions. Administration is the more critical administrative act and must be accompanied by more plausibility and consistency checks.

The following requirements can derived with respect to separation of organization and administration:

AR5.9 – Separation: Efficient Policy Engineering (Optional)

The ORKA architecture SHOULD allow the efficient translation of companies organizationally structure into role hierarchies and security policy rules. This requirement is intended to provide the security goals confidentiality and compliance without restriction of performance and availability.

AR5.10 – Separation: Authorization of Administrators (Mandatory)

Only authorized and identified persons (via password or certificate) MAY have access to the administrator console / interface. This requirement meets the security goals confidentiality, non-repudiation, and integrity. It preserves infiltration and misuse of access rights and sensitive data. It also meets the case study requirement AR4.1 *Access only for Authorized Users*.

AR5.11 – Separation: Separation of Administrative Tasks (Mandatory)

It MUST be possible to separate administrative tasks into organization-related (e.g. assignment of users to roles) and administration-related (e.g. assign a new permission to a

role) tasks. This requirement is intended to manage and protect access rights in an efficient manner and to preserve more critical administrative acts and sensitive data against misuse and harm.

AR5.12 – Separation: Domain-specific Administration (Mandatory)

Users, administrators, and security administrators **MUST** act in different domains according to their organizational role and their integration in a role based hierarchy. No user or administrator should be permitted to change its own access rights. This requirement meets the security goals confidentiality and integrity and prevents infiltration.

AR5.13 – Separation: Notification of Actually Access Rights (Optional)

The administrative interface **SHOULD** allow to notify the actually relevant access rights for users, administrators, and security administrators. This requirement is intended to simplify administration and to avoid erroneous administrative acts.

AR5.14 – Separation: Execute Administrative Operations (Mandatory)

The administrative interface **MUST** allow to execute all relevant administrative operations. The needful operations are listed in section 6 *Policy Management Requirements*.

5.3 Traceability and Auditing

To provide reasonable assurance it is meaningful to monitor all security-relevant events and to save audit trails including time and causer. To secure administration it is needful to control administrators identity and access rights. To enable administrative interventions by human administrators it is meaningful to notify violations of given enterprise security principles.

Reference Monitor for Administrative Actions According to the proof of identity and authorization of users, administrators rights must be controlled by a decision component to ensure that the administration component is not accessible for unauthorized personal or attackers. This component is called *Reference Monitor*.

AR5.15 – Reference Monitor: Controlling Administrative Operations (Mandatory)

All administrative operations **MUST** be controlled by a reference monitor.

AR5.16 – Reference Monitor: Authorization of Administrative Operations (Mandatory)

All administrative operations **MUST** be authorized by a reference monitor before execution.

Compliance It is needful that an enterprise adheres to the relevant law and meaningful to specify company-wide relevant rules. This rules are put down in the company-wide policy. The adherence to relevant standards should be provided automatically. Compliance is meaningful to achieve the business goals.

AR5.17 – Compliance: Support for Compliance Rules (Optional)

There SHOULD be a component to prevent violations of organizational security policies, standards and legal stipulations.

AR5.18 – Compliance: Support for Static Constraints (Optional)

There SHOULD be a component to prevent the creation of role hierarchies that are disallowed by the constraints defined in a given security policy. Additionally, there SHOULD be a component to prevent the assignment of access permissions that are disallowed by the constraints defined in a given security policy.

Logging / Auditing Logging and auditing are the prerequisites to provide reasonable assurance. All security relevant operations performed at the administration interface must be monitored by automatic or manual controllable components. The monitored system data should be audited periodically by independent personal to prevent misuse of sensitive data and critical enterprise assets. The requirements listed in this section subdivide the case study requirement AR4.2 *Auditing Measures* into differentiated auditing and logging requirements.

AR5.19 – Supervision: Actually Relevant Policy (Mandatory)

There MUST be a component that monitors the version of the actually relevant policy including date and author.

AR5.20 – Supervision: Controlling Policy Modifications (Mandatory)

There MUST be a component that monitors policy modifications by security personnel.

AR5.21 – Supervision: Specification of Logging Rules (Optional)

There SHOULD be a component that is able to store logging rules as part of the security policy or separately.

AR5.22 – Supervision: Specify Security-Relevant Events (Optional)

There SHOULD be a component that is able to interpret policy elements that specify logging requirements for security-relevant events. Realizing AR5.21 it is meaningful to realize this requirement too.

AR5.23 – Supervision: Monitor Policy Check Results (Mandatory)

There MUST be a component that monitors the violation of policy property checks or policy consistency checks (e.g., see Sections 5.3 and 5.6)

AR5.24 – Supervision: Monitoring Events with Time and Causer (Mandatory)

There MUST be a component that monitors all security-relevant events including time and causer.

AR5.25 – Supervision: Monitoring Administrative Operations (Mandatory)

There MUST be a component that monitors all administrative operations.

AR5.26 – Supervision: Controlling Administrative Access Rights (Mandatory)

There **MUST** be a component that monitors successful and unsuccessful authorization events and changes in access control.

AR5.27 – Supervision: Controlling Identification and Authentication (Mandatory)

There **MUST** be a component that monitors successful and unsuccessful identification and authentication.

AR5.28 – Supervision: Human Controlling of Selected Security-relevant Events (Optional)

Human administrators **SHOULD** be able to require notification of selected security-relevant events.

AR5.29 – Supervision: Human Controlling of Log Information (Optional)

Human administrators **SHOULD** be able to startup and shutdown security audit.

AR5.30 – Supervision: Automatic Controlling of Log Information (Optional)

There **SHOULD** be a component that requires a review of log information periodically.

AR5.31 – Supervision: Log Protection (Mandatory)

All security-relevant log information **MUST** be protected with respect to integrity and unauthorized modifications by appropriate measures like timestamps and digital signatures.

5.4 Policy Representation

Policy Representation and Preview Policy representation and preview is intended to simplify administration and to avoid erroneous administrative acts. It supports human interactions and interventions.

AR5.32 – Representation: Expressiveness of Policy Language (Optional)

The administration interface **SHOULD** be capable to represent various views of assignments according to expressiveness requirements as described in the SPEC work packages of ORKA. Some examples of constraints and assignments, that may be represented at the administration interface, are given in section 5.6, paragraph *Expressive power*. All those requirements characterized as mandatory in deliverable WP 2.2 [2] *Requirements for the Policy Language* **MUST** be representable at the administration interface.

AR5.33 – Representation: Actual Snapshot (Optional)

The administration interface **SHOULD** be capable to represent an actual snapshot of assignments e.g. users to roles, permission to roles, and roles to tasks.

AR5.34 – Representation: Possible Actions (Mandatory)

The administration interface **MUST** be capable to represent **the possible (allowed)** actions that could be performed by administrators / users at a given point of time.

AR5.35 – Representation: Forbidden Actions (Mandatory)

The administration interface **MUST** be capable to represent **the forbidden** actions that could be performed by administrators / users at a given point of time.

AR5.36 – Representation: Preview (Mandatory)

The administration interface **MUST** be capable to represent **a preview** of proposed assignments performed by administrators / users.

AR5.37 – Representation: Various Kinds of Illustrations (Optional)

The administration interface **SHOULD** be capable to represent various kinds of illustrations / notations of policies. E.g a graphical role hierarchy diagram or an access matrix representing the given assignments (e.g. users to roles, permission to roles, and roles to tasks).

History of policy definitions The requirements, listed in this subsection, support the requirements given in subsection 5.3 *Logging/Auditing*. They make it more easier for human administrators to review a given policy.

AR5.38 – History: Policy Versions (Optional)

The administration interface **SHOULD** be capable to represent the version of a policy including date and author.

AR5.39 – History: Policy Comments (Optional)

The administration interface **SHOULD** be capable to represent comments given within a policy specification.

Context information According to policy representation and review it is meaningful to provide representation of context information to support administration and human controlling.

AR5.40 – Representation: Context Information (Optional)

The administration interface **SHOULD** be capable to represent given context information. This requirement corresponds to AR5.48 in section 5.6 *Policy Engineering*.

5.5 Enterprise Integration

The requirements listed in this subsection are intended to describe a meaningful system architecture considering security aspects as well as common system integration aspects.

AR5.41 – Integration: Transform Policy Representation (Mandatory)

It **MUST** be possible to transform a policy representation into other description languages to submit formal validation and efficient enforcement.

AR5.42 – Integration: Modular and Flexible Architecture (Mandatory)

The policy administration system architecture **MUST** be modular and flexible allowing it to be integrated with diverse application environments.

AR5.43 – Integration: Modular wrt. Validation and Enforcement (Mandatory)

The policy administration interface **MUST** be designed and implemented independent of the validation and enforcement components.

AR5.44 – Integration: Distributed Components – Secure Communication (Mandatory)

Using a distributed system architecture (e.g. client/server architecture) between administration components and decision and enforcement components, communication **MUST** be secure (e.g. using SSL).

5.6 Policy Engineering

Expressive power The expressive power of the policy language used in ORKA is defined in deliverable WP 2.2 [2] *Requirements for the Policy Language*. The policy administration interface should be able to provide (not comprehensive):

- access roles - typically RBAC model
- constraints: various types of static and dynamic SoD constraints, constraints on delegation
- cardinality constraints: cardinality constraints for roles and permissions, definition of maximum and minimum cardinality
- context constraints: time constraints, history-based constraints
- role hierarchies
- administrative scope in role hierarchies
- workflow constraints: definition of job positions, definition of constraints for jobs
- assignment of roles to tasks (workflow specific)
- delegation of tasks
- role activation constraints
- activation of a role by presenting a credential

It should be possible to express and provide constraints particularly with regard to the rights and duties of administrators. As an example the reader may see the case study requirement AR4.4 *Separation of Duties for Policy Administration*.

The following requirements can be derived from the above. Some of them are also match requirements of work package 2.2.

AR5.45 – Engineering: Round-Trip-Policy-Administration (Optional)

It **SHOULD** be possible to import policies from subsystems and, after checking and maintaining the composite policy, to export the split policy into subsystems again.

AR5.46 – Engineering: Definition, Maintenance, and Query using GUIs (Optional)

Despite other kinds of administration (like command line tools), a visual policy builder (graphical user interface) MAY support the definition, maintenance and query of security policies.

AR5.47 – Engineering: Specification of Policy Statements (Mandatory)

The policy administration interface MUST be capable to be used to define the kind of policy specified in ORKA, and this policy MUST be transformed into rules needed by the enforcement component to make decisions.

AR5.48 – Engineering: Support Context Information (Mandatory)

The administration interface MUST be capable to add context information given in a file or a database or taken from some sensors for system-internal or external information.

AR5.49 – Engineering: Execute Consistency Checks (Mandatory)

The administration interface MUST have access to a component that provide consistency checks, i.e. checking whether a policy contains inherent contradictions.

AR5.50 – Engineering: Execute Completeness Checks (Mandatory)

The administration interface MUST have access to a component that provide completeness checks, i.e. checking whether a given policy is sufficient to express some required organizational control principles.

AR5.51 – Engineering: Execute Security Property Checks (Mandatory)

The administration interface MUST have access to a component that provide security property checks, i.e. checking whether a given policy adheres to some security properties such as the principle of least privilege.

5.7 Usability

Since computer systems become more large scaled, ubiquitous and connected via networks, the importance and complexity of computational infrastructures has increased [4]. This section provides requirements for usable security. We will differentiate between usability requirements that hold for computer users in general and usability requirements that specifically refer to administrators of computer systems.

As far as we know, the first definition on usability for IT security was given by *Whitten* and *Tygar* in [25, 26]. They define:

Definition: Security software is usable if the people who are expected to use it:

1. are reliably made aware of the security tasks they need to perform;
2. are able to figure out how to successfully perform those tasks;
3. don't make dangerous errors; and
4. are sufficiently comfortable with the interface to continue using it.

Recently, *Chiasson, van Oorschot, and Biddle* have proposed to extend this definition by adding another two criteria for usability of security software [9]:

5. are able to tell when their task has been completed; and
6. have sufficient feedback to accurately determine the current state of the system.

Maskery et al. [14] gives the following advantages of considering usability aspects in application design:

- mechanism becomes invisible to user,
- permits more natural interaction,
- clarifies thinking,
- reduces errors,
- increases productivity,
- increases intrinsic motivation,
- reduces confusion,
- reduces training time,
- allow flexibility, and
- give robustness.

The following usability requirements can be derived from the criteria above.

AR5.52 – Usability: Context-dependent Access to Operations (Optional)

The user interface SHOULD allow to access all relevant operations in a context-dependent matter, i.e. not relevant operations are not accessible.

AR5.53 – Usability: User Interface Design (Optional)

User interface and interaction elements SHOULD be labeled and visualized in an appropriate way in order to allow the user to find the expected element and to perform the intended task.

AR5.54 – Usability: Feedback After Processing (Mandatory)

The user interface MUST provide the user with some feedback, whether an operation successfully finished or some error occurred.

AR5.55 – Usability: Ease of Use (Optional)

All user interface elements and interaction patterns SHOULD follow the *ease of use* principle.

AR5.56 – Usability: System Documentation, Online Help System (Optional)

All functions and interaction elements of the user interface SHOULD be well documented. The system MAY also provide an online help system for context-dependent help.

AR5.57 – Usability: Provide an Undo Function (Optional)

There SHOULD be an undo function for all operations on the user interface. This allows the computer user to revoke an operation that was just performed.

AR5.58 – Usability: Support for Cognitive Models (Optional)

In order to reduce the cognitive load of computer users, the user interface and interaction patterns MAY be based on the user's cognitive model of the system.

AR5.59 – Usability: Prevent Dangerous Errors (Optional)

The model SHOULD prevent dangerous errors by means of e.g. plausibility checks and confirmation dialogs.

In addition to criteria for usability of security in general, the role of administrators needs to be considered. In [4], *Barret, Chen, and Maglio* point out that

The focus of human-computer interaction work has been on the end users of computer systems. However another important class of computer users is the cohort of administrators who design, build, maintain and troubleshoot computer systems as their main work.

The notion of usable security for end users and security administrators is not the same, as *Zurko et al.* show in [29]: Administrators have different background, training, goals, constraints and use different tool. Due to these differences, the (perception of) usability of the protection mechanisms and other security tools may also be affected.

The special role of computer administrators was also pointed out by *Beznosov et al.* [5, 6], *Botta* [7] and *Yurcik et al.* [28]. Only if administrators are provided with appropriate and usable tools, the security of IT systems can be assured.

The following usability requirements with focus on administrators of computer systems have been derived:

AR5.60 – Usability: Visualize the State of the System (Mandatory)

The administration interface MUST allow the administrator to inspect the current state of the system (at least with respect to policy specification and enforcement).

AR5.61 – Usability: Support for What-If Scenarios (Optional)

The administration model MAY have support for so called what-if scenarios. That is, the administrator may try out some operations and preview the consequences of the results before the operation are actually carried out.

AR5.62 – Usability: Inform About Potentially Unwanted Consequences (Optional)

The administration model MAY inform an administrator before the execution of an operation about potentially unwanted consequences as a result of the operation.

AR5.63 – Usability: Support for Various Views (Optional)

The user interface SHOULD provide various visualizations with different granularity in order to provide the administrator with the necessary information.

AR5.64 – Usability: Machine-readable and -writable Policies (Mandatory)

The administration model **MUST** translate between the views and operations provided at the user interface and the underlying policy specification as base of visualizations and target of policy changes.

The policy language must not only be machine-readable, it must also be machine-writable. Machine writability is necessary for applications that aid users with creating appropriate policies by offering wizards or graphical point and click interfaces.

AR5.65 – Usability: Human Writability and Readability (Controllability) (Optional)

Humans **SHOULD** be able to read and write policy specifications without too much effort. In general, human administrators must be able to have control over their policy definitions by being able to understand definitions and to express demands.

This requirement is somewhat reduced in scenarios where a (possibly graphical) policy administration tool offers an easy to use interface for specifying policies. Nevertheless, even in such scenarios, the underlying language must be comprehensible to skilled users for trouble-shooting and development purposes.

AR5.66 – Usability: Support for Comments (Optional)

The policy specification language **SHOULD** allow adding comments within a policy specification. Comments are necessary for storing meta information like explanations why a certain rule is necessary and what it is intended for. Moreover, comments allow keeping information why, when, and by whom certain parts of the specification have been changed.

AR5.67 – Usability: Support for Macros (Optional)

The administration interface **MAY** support macro expansion including parameter substitution to allow central maintenance of recurring definitions.

AR5.68 – Usability: Maintainability (Optional)

In general the administration interface **SHOULD** allow easy maintenance of the policy definitions. This requirement subsumes all requirements that are necessary for easy policy maintenance but are not explicitly mentioned in the previous requirements. For example the policy language **MAY** be text-based in order to allow the application of standard text-editing tools and version control systems.

6 Policy Management Requirements

This section describes the use cases that have been identified for supporting the administrative operations of the ORKA system.

Some of the use cases have also been defined in the RBAC standard [3]. We will define the use cases that are mandatory for the ORKA project within this document. Depending on the time available for the ORKA implementation, we may implement other use cases from the RBAC standard as well.

6.1 Working with Policies

6.1.1 Create a policy

AR6.1 – Use Case: Create a policy (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number:	P.1
Application:	Policy management
Use case name:	Create a policy
Use case description:	A policy administrator creates a new policy object
Primary actor:	Policy administrator
Preconditions:	The policy administrator is logged on
Trigger:	The policy administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator enters an account id of the policy to be created• The system validates that the account id does not yet exist• The policy administrator enters additional policy information<ul style="list-style-type: none">– description (optional)• The policy administrator saves the policy
Alternate flows:	None

6.1.2 Edit a policy

AR6.2 – Use Case: Edit a policy (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number:	P.2
Application:	Policy management
Use case name:	Edit a policy
Use case description:	A policy administrator edits an existing policy object
Primary actor:	Policy administrator
Preconditions:	The policy administrator is logged on
Trigger:	The policy administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator enters a policy account id• The system validates that the account id exists• The policy administrator modifies the policy• The policy administrator saves the policy
Alternate flows:	None

6.1.3 Validate a policy

AR6.3 – Use Case: Validate a policy (Mandatory)

The ORKA administration model **MUST** support the use case as described below.

Use case number:	P.3
Application:	Policy management
Use case name:	Validate a policy
Use case description:	A policy administrator validates an existing policy object
Primary actor:	Policy administrator
Preconditions:	The policy administrator is logged on
Trigger:	The policy administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator enters a policy account id• The system validates that the account id exists• The system performs a policy validation and presents the results to the policy administrator
Alternate flows:	None

6.1.4 Preview policy changes

AR6.4 – Use Case: Preview policy changes (Optional)

The ORKA administration model **SHOULD** support the use case as described below.

Use case number:	P.4
Application:	Policy management
Use case name:	Preview policy changes
Use case description:	The policy administrator queries which changes will occur if the current policy is being released
Primary actor:	Policy administrator
Preconditions:	<ul style="list-style-type: none">• The policy administrator is logged on• The policy has been validated successfully
Trigger:	The policy administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator enters a policy account id• The system validates that the account id exists• The policy administrator modifies the policy• The system analyzes if there are possible side effects and presents the result to the policy administrator
Alternate flows:	None

6.1.5 Release a policy

AR6.5 – Use Case: Release a policy (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number:	P.5
Application:	Policy management
Use case name:	Release a policy
Use case description:	A policy administrator releases a policy object
Primary actor:	Policy administrator
Preconditions:	<ul style="list-style-type: none">• The policy administrator is logged on• The policy has been validated successfully
Trigger:	The policy administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator enters a policy account id• The system validates that the account id exists• The system releases the policy into production
Alternate flows:	None

6.1.6 Delete a policy

AR6.6 – Use Case: Delete a policy (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number:	P.6
Application:	Policy management
Use case name:	Release a policy
Use case description:	A policy administrator deletes a policy object
Primary actor:	Policy administrator
Preconditions:	The user administrator is logged on
Trigger:	The user administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator enters an account id of the policy to be deleted• The system validates that the account id exists• The policy administrator confirms the deletion of the account• The system deletes the policy and all associated information
Alternate flows:	None

6.2 Working with Users

6.2.1 Create a user

AR6.7 – Use Case: Create a user (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number:	U.1
Application:	User management
Use case name:	Create a user
Use case description:	A user administrator creates a user that can use ORKA-enabled applications to get his work done
Primary actor:	User administrator
Preconditions:	The user administrator is logged on
Trigger:	The user administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The user administrator enters an account id of the user to be created• The system validates that the account id does not yet exist• The user administrator enters additional user information<ul style="list-style-type: none">– given name (mandatory)– family name (mandatory)– description (optional)– email address (optional)– phone number (optional)• The user administrator saves the user
Alternate flows:	None

6.2.2 Edit a user

AR6.8 – Use Case: Edit a user (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: U.2
Application: User management
Use case name: Edit a user
Use case description: A user administrator edits a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters an account id of the user to be edited
- The system validates that the account id exists
- The user administrator edits the user information
 - given name (mandatory)
 - family name (mandatory)
 - description (optional)
 - email address (optional)
 - phone number (optional)
- The user administrator saves the user

Alternate flows: None

6.2.3 Delete a user

AR6.9 – Use Case: Delete a user (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: U.3
Application: User management
Use case name: Delete a user
Use case description: A user administrator deletes a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters an account id of the user to be deleted
- The system validates that the account id exists
- The user administrator confirms the deletion of the account
- The system deletes the user and all associated information

Alternate flows: None

6.2.4 Assign a role

AR6.10 – Use Case: Assign a role (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: U.4
Application: User management
Use case name: Assign a role to a user
Use case description: A user administrator assigns a role to a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters a user account id
- The system validates that the user account id exists
- The user administrator enters a role account id
- The system validates that the role account id exists
- The system validates the the role has not been assigned yet
- The system confirms that the assignment will not violate the security policy
- The system assigns the role to the user

Alternate flows: None

6.2.5 Remove a role

AR6.11 – Use Case: Remove a role (Mandatory)

The ORKA administration model **MUST** support the use case as described below.

Use case number: U.5
Application: User management
Use case name: Revoke a role from a user
Use case description: A user administrator revokes a role from a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters a user account id
- The system validates that the user account id exists
- The system displays the roles assigned to the user
- The user administrator selects an assigned role
- The system confirms that the revoke operation will not violate the security policy
- The system revokes the role from the user

Alternate flows: None

6.2.6 Show effective roles

AR6.12 – Use Case: Show effective roles (Optional)

The ORKA administration model **SHOULD** support the use case as described below.

Use case number: U.6
Application: User management
Use case name: Show effective user roles
Use case description: A user administrator queries the effective roles of a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters a user account id
- The system validates that the user account id exists
- The system displays the effective roles assigned to the user

Alternate flows: None

6.2.7 Show effective permissions

AR6.13 – Use Case: Show effective permissions (Optional)

The ORKA administration model SHOULD support the use case as described below.

Use case number: U.7
Application: User management
Use case name: Show effective user permissions
Use case description: A user administrator queries the effective permissions of a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters a user account id
- The system validates that the user account id exists
- The system displays the effective permissions assigned to the user

Alternate flows: None

6.2.8 Show administrative permissions

AR6.14 – Use Case: Show administrative permissions (Optional)

The ORKA administration model SHOULD support the use case as described below.

Use case number: U.8
Application: User management
Use case name: Show administrative permissions
Use case description: A user administrator queries the administrative permissions of a user
Primary actor: User administrator
Preconditions: The user administrator is logged on
Trigger: The user administrator selects the operation in the user interface.
Basic flow:

- The user administrator enters a user account id
- The system validates that the user account id exists
- The system displays the administrative permissions assigned to the user

Alternate flows: None

6.3 Working with Roles

6.3.1 Create a role

AR6.15 – Use Case: Create a role (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.1
Application: Policy management
Use case name: Create a role
Use case description: A policy administrator creates a new role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters the account id of the role to be created
- The system validates that the role account id does not yet exist
- The policy administrator enters additional role information
 - description (optional)
- The policy administrator saves the role

Alternate flows: None

6.3.2 Edit a role

AR6.16 – Use Case: Edit a role (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.2
Application: Policy management
Use case name: Edit a role
Use case description: A policy administrator edits an existing role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters the account id of the role to be edited
- The system validates that the role account id exists
- The policy administrator edits the role information
 - description (optional)
- The policy administrator saves the role

Alternate flows: None

6.3.3 Delete a role

AR6.17 – Use Case: Delete a role (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.3
Application: Policy management
Use case name: Delete a role
Use case description: A policy administrator deletes an existing role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters the account id of the role to be deleted
- The system validates that the role account id exists
- The policy administrator confirms the deletion of the role
- The system deletes the role and all associated information

Alternate flows: None

6.3.4 Assign a user

AR6.18 – Use Case: Assign a user (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.4
Application: Policy management
Use case name: Assign a user
Use case description: A policy administrator assigns a user to a role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters a role account id
- The system validates that the role account id exists
- The policy administrator enters a user account id
- The system validates that the user account id exists
- The system validates the the user has not been assigned yet
- The system confirms that the assignment will not violate the security policy
- The system assigns the user to the role

Alternate flows: None

6.3.5 Remove a user

AR6.19 – Use Case: Remove a user (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.5
Application: Policy management
Use case name: Remove a user
Use case description: A policy administrator removes a user from a role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters a role account id
- The system validates that the role account id exists
- The system displays the users assigned to the role
- The policy administrator selects an assigned user
- The system confirms that the revoke operation will not violate the security policy
- The system revokes the user from the role

Alternate flows: None

6.3.6 Show effective role users

AR6.20 – Use Case: Show effective role users (Optional)

The ORKA administration model SHOULD support the use case as described below.

Use case number: R.6
Application: Policy management
Use case name: Show effective role users
Use case description: A policy administrator queries the effective users of a role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters a role account id
- The system validates that the role account id exists
- The system displays the effective users assigned to the role

Alternate flows: None

6.3.7 Show effective role permissions

AR6.21 – Use Case: Show effective role permissions (Optional)

The ORKA administration model SHOULD support the use case as described below.

Use case number: R.7
Application: Policy management
Use case name: Assign a user
Use case description: A policy administrator queries the effective permissions of a role
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters a role account id
- The system validates that the role account id exists
- The system displays the effective permissions assigned to the role

Alternate flows: None

6.3.8 Create role inheritance

AR6.22 – Use Case: Create role inheritance (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.8
Application: Policy management
Use case name: Create role inheritance
Use case description: A policy administrator creates an inheritance relationship between two roles
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters the role account id of the parent role
- The system validates that the parent role account id exists
- The policy administrator enters the role account id of the child role
- The system validates that the child role account id exists
- The system checks that no role graph cycle will be introduced by the new inheritance edge
- The system creates the inheritance relationship

Alternate flows: None

6.3.9 Delete role inheritance

AR6.23 – Use Case: Delete role inheritance (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: R.9
Application: Policy management
Use case name: Delete role inheritance
Use case description: A policy administrator deletes an inheritance relationship between two roles
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator enters the a role account id
- The system validates that the parent role account id exists
- The system displays all inheritance relationships that belong to the role
- The policy administrator selects the inheritance relationship to be deleted
- The policy administrator confirms the deletion of the inheritance relationship
- The system deletes the inheritance relationship

Alternate flows: None

6.4 Working with workflows

6.4.1 Assign role to workflow task

AR6.24 – Use Case: Assign role to workflow task (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number:	W.1
Application:	Policy management
Use case name:	Assign role to workflow task
Use case description:	A policy administrator assigns a role to a workflow task
Primary actor:	Policy administrator
Preconditions:	The policy administrator is logged on
Trigger:	The policy administrator selects the operation in the user interface.
Basic flow:	<ul style="list-style-type: none">• The policy administrator selects the workflow• The system displays the workflow tasks• The policy administrator selects the workflow task to be modified• The policy administrator changes the assigned role of the workflow task• The system validates the policy constraints• The policy administrator saves the workflow
Alternate flows:	None

6.4.2 Remove role from workflow task

AR6.25 – Use Case: Remove role from workflow task (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: W.2
Application: Policy management
Use case name: Remove role from workflow task
Use case description: A policy administrator removes a role from a workflow task
Primary actor: Policy administrator
Preconditions: The policy administrator is logged on
Trigger: The policy administrator selects the operation in the user interface.
Basic flow:

- The policy administrator selects the workflow
- The system displays the workflow tasks
- The policy administrator selects the workflow task to be modified
- The policy administrator removes the assigned role of the workflow task
- The system validates the policy constraints
- The policy administrator saves the workflow

Alternate flows: None

6.5 Audit

6.5.1 Show audit protocol

AR6.26 – Use Case: Show audit protocol (Mandatory)

The ORKA administration model MUST support the use case as described below.

Use case number: A.1
Application: Policy management
Use case name: Show audit protocol
Use case description: A policy auditor lists the audit protocol
Primary actor: Policy auditor
Preconditions: The policy auditor is logged on
Trigger: The policy auditor selects the operation in the user interface.
Basic flow:

- The system displays the audit protocol, entries sorted by date, newest entries first

Alternate flows: None

6.5.2 Search audit protocol by event

AR6.27 – Use Case: Search audit protocol by event (Optional)

The ORKA administration model MUST support the use case as described below.

Use case number: A.2
Application: Policy management
Use case name: Search audit protocol by event
Use case description: A policy auditor queries the audit protocol for certain events
Primary actor: Policy auditor
Preconditions: The policy auditor is logged on
Trigger: The policy auditor selects the operation in the user interface.
Basic flow:

- The policy auditor selects the events to be filtered
- The system displays all audit events the match the filter, sorted by date, newest entries first

Alternate flows: None

6.5.3 Search audit protocol by user

AR6.28 – Use Case: Search audit protocol by user (Optional)

The ORKA administration model MUST support the use case as described below.

Use case number: A.3
Application: Policy management
Use case name: Search audit protocol by user
Use case description: A policy auditor queries the audit protocol for certain user events
Primary actor: Policy auditor
Preconditions: The policy auditor is logged on
Trigger: The policy auditor selects the operation in the user interface.
Basic flow:

- The policy auditor selects the users to be filtered
- The system displays all audit events the match the filter, sorted by date, newest entries first

Alternate flows: None

6.5.4 Search audit protocol by role

AR6.29 – Use Case: Search audit protocol by role (Optional)

The ORKA administration model MUST support the use case as described below.

Use case number: A.4
Application: Policy management
Use case name: Search audit protocol by role
Use case description: A policy auditor queries the audit protocol for certain role events
Primary actor: Policy auditor
Preconditions: The policy auditor is logged on
Trigger: The policy auditor selects the operation in the user interface.
Basic flow:

- The policy auditor selects the roles to be filtered
- The system displays all audit events the match the filter, sorted by date, newest entries first

Alternate flows: None

6.5.5 Search audit protocol by policy

AR6.30 – Use Case: Search audit protocol by policy (Optional)

The ORKA administration model MUST support the use case as described below.

Use case number: A.4
Application: Policy management
Use case name: Search audit protocol by policy
Use case description: A policy auditor queries the audit protocol for certain policy events
Primary actor: Policy auditor
Preconditions: The policy auditor is logged on
Trigger: The policy auditor selects the operation in the user interface.
Basic flow:

- The policy auditor selects the policies to be filtered
- The system displays all audit events the match the filter, sorted by date, newest entries first

Alternate flows: None

7 Requirements for Constraint Specification in Workflows

7.1 Introduction

This section describes requirements to be able to specify authorization constraints in workflow environments with respect to the Policy Administration Tool. It basically addresses the question what aspects about a workflow and its definition have to be known to be able to define constraints on it.

It will contribute as foundation for further work packages in ORKA, especially work package 5.3 where the technical design of the administration tool for the specification of authorization constraints will be developed.

7.2 Business Process Specification

Business processes are defined with workflow specifications. These specifications are usually defined using business process definition languages. There are different languages available to define a process flow. Examples are the 'Business Process Execution Language' (BPEL), the 'jBPM Process Definition Language' (jPDL) or the 'XML Process Definition Language' (XPDL). BPEL, for instance, one of the more popular workflow specification languages has been submitted to the OASIS WS-BPEL technical committee for standardization of its current version 'WS-BPEL 2.0' [16].

The mentioned languages mainly differ in their expressiveness and diversity according to their supported workflow patterns. An overview about which patterns of the identified workflow patterns listed in [23] are supported by the different process definition languages is given [22]. Next we will give a short overview about three process languages to give a short impression what the following section is referring to.

The 'Business Process Execution Language' (BPEL) provides a language for formal specification of business processes within web services. Its current version WS-BPEL 2.0 defines the process flow using an XML-based language and describes the process logic of invocation of the involved Web services.

The 'jBPM Process Definition Language' (jPDL) is provided by JBOSS for their workflow engine called jBPM. jBPM is 'a platform for multiple process languages supporting workflow, BPM, and process orchestration' [13]. JPDL specifies the process definition using XML and defines process archive to package all the process definition related files into one archive.

The 'XML Process Definition Language' (XPDL) is as its name already implies an XML based workflow definition language standardized by the Workflow Management Coalition group [24]. In its version 2.0 available since May 2005 the language not only focuses on the executable aspect of business processes but also supports a graphical representation as process diagram.

Important for the analysis of requirements for the administration tool with respect to the workflow management system is to be aware that the single tasks representing a business process are defined using certain workflow definition languages. To eventually define authorization constraints on certain tasks or a business process in total the process definitions have to be available to the system, especially to the administration tool.

7.3 Workflow specific Requirements for constraint specification

This sections names and describes the requirements for constraint specification for business processes with regard to the administration tool.

AR7.1 – Workflows: Read in of Workflow Definition (Mandatory)

The workflow definition is the basis for any constraint definition since the constraints are geared to the single business process definitions. It therefore **MUST** be possible to read in predefined workflow specifications.³

AR7.2 – Workflows: Unique Workflow Identification (Mandatory)

Every workflow should be unambiguously identifiable. If available, this could be done by interpreting a given identifier within the workflow definition.

AR7.3 – Workflows: Interpretation of Workflow Context Information (Optional)

A workflow definition may provide information about its creator, a creation date, information about updates, a description or comments given by the author. Such information may be helpful for the security administrator and hence **MAY** be interpreted for later presentation.

AR7.4 – Workflows: Task Extraction (Mandatory)

The ORKA administration tool will be used to assign security constraints to the a business process. To be able to perform this step, there is the crucial necessity that the individual tasks of a business process can be for the assignment. This means that an extraction of single task definitions **MUST** be possible from the process definitions.

AR7.5 – Workflows: Task Identification (Mandatory)

Along with the task extraction, unique identification of a task is necessary to be able to uniquely assign constraints to tasks. Tasks of a workflow definition **MUST** be uniquely identified.

AR7.6 – Workflows: Interpretation of Task Context Information (Optional)

Similar to the context information for a whole workflow (see AR 7.3 there might also be such information as comments or descriptions for single tasks of a process which may be helpful for the security administrator and hence **MAY** be interpreted for later presentation.

AR7.7 – Workflows: Identification of linear Order of Tasks (Mandatory)

A workflow is in first place a process of tasks with linear order⁴. It **MUST** be possible to identify the linear order of tasks to be able to apply security constraints which rely on the order of tasks.

AR7.8 – Workflows: Basic Workflow Pattern Recognition (Optional)

The five basic workflow specific patterns identified by [23] **SHOULD** be recognized and provided to the security administrator.⁵ It would help the administrator to define authorization constraints more easily.

The basic patterns include namely⁶:

³This requirements refers to the general possibility reading workflow definitions and interpreting its structure. The need for the interpretation of single elements of such an definition is covered by the remaining requirements of this section.

⁴It may become non-linear when using workflow patterns. See also AR7.8.

⁵Please note that the five basic patterns include also the *sequence pattern* which is already covered by AR7.7.

⁶Descriptions taken from [27]

1. Sequence (mandatory, see AR 7.7): An activity in a workflow process is enabled after the completion of a preceding activity in the same process.
2. Parallel Split: The divergence of a branch into two or more parallel branches each of which execute concurrently.
3. Synchronization: The convergence of two or more branches into a single subsequent branch such that the thread of control is passed to the subsequent branch when all input branches have been enabled.
4. Exclusive Choice: The divergence of a branch into two or more branches. When the incoming branch is enabled, the thread of control is immediately passed to precisely one of the outgoing branches based on the outcome of a logical expression associated with the branch.
5. Simple Merge: The convergence of two or more branches into a single subsequent branch. Each enablement of an incoming branch results in the thread of control being passed to the subsequent branch.

AR7.9 – Workflows: Advanced Workflow Pattern Recognition (Optional)

Further workflow patterns identified by [23] MAY be supported.

AR7.10 – Workflows: Identification of non-linear Order of Tasks (Optional)

It SHOULD be possible to identify non-linear order of tasks, e.g. the parallel execution of tasks. This requirement becomes MANDATORY if the basic patterns of AR 7.8 become implemented.

AR7.11 – Workflows: Distinction btw. Manual and Automatic Tasks (Optional)

There are tasks which are intended to be performed automatically by a certain predefined component. There are also tasks which are intended to be performed with user interaction. To assign security constraints properly the difference between tasks intended to be performed manually and tasks intended to be performed automatically SHOULD be recognized from the workflow definition and made available to the administrator.

AR7.12 – Workflows: Composite Tasks (Optional)

There are workflow definition languages which support subflows/composite tasks. Such a structure describes that one task consists of multiple sub-tasks. An appropriate handling for constraint specification for sub-tasks SHOULD be available.

AR7.13 – Workflows: Process Dataflow and/or related Data Objects Interpretation (Optional)

Workflow definitions may also contain dataflow definitions (e.g. input/output parameters for a certain tasks within a process) which might be necessary for constraint specifications. Hence, such information SHOULD be interpreted and made available to the security administrator. This requirements becomes MANDATORY if constraints are supported by the administration interface which need such type of data objects or dataflow information and the underlying workflow definition also provides it (see also section 5.6).

AR7.14 – Workflows: Distinction of Different Versions of Workflow Specifications (Optional)

Business processes and consequently their definitions are subject to change over time. This means also already implemented process definitions might get updated and subsequently authorization constraint definitions have to be updated as well. It should therefore be possible to distinguish between different versions of a certain workflow definitions.

AR7.15 – Workflows: Independent Constraint Assignment (Optional)

As listed in section 7.2, there are different languages available to specify the execution plan of business processes (e.g. BPEL, jPDL, XPDL, etc.). The definition of constraints for a certain workflow specification SHOULD be independently with regard to the underlying workflow definition language used to specify the workflow process. This requirement would help to be not bound to one specific specification language.

8 Conclusion

Within this document we have identified the requirements needed for the design of the ORKA policy management system. The requirements consist of functional requirements that define the user's interaction with the system, and the non-functional requirements like security of availability that impose constraints on the function requirements.

Security goals have defined that will have to be met by the ORKA policy administration design and implementation.

We analyzed the requirements for policy management tools in general. They have to support the policy lifecycle of definition, storage, maintenance and removal. This lifecycle will have to be supported by ORKA too.

We described the Ponder policy management tool as an example of a real world implementation.

In the ORKA requirements engineering process, the requirements have been derived from the case studies developed in ORKA workpackage 1.1, from the administration model itself, from the policy administration process and from workflow constraints.

As quite a lot of requirements have been collected, we had to classify them as mandatory or optional. With this classification, we can provide better support for the ORKA project schedule and make sure all important requirements will be met by the administration model and implementation.

The policy administration model requirements have been grouped into categories for easier classification and handling.

The policy management requirements have been specified using use cases. We identified the use case categories Policies, Users, Roles, Workflows and Audit.

The workflow requirements have been derived from workflow language specifications and from constraints found while analyzing the real-world workflows documented in workpackage 1.1.

In the next workpackage 5.2, we will develop a requirements reference catalog and analyze existing policy administration models if and which requirements they support.

Within workpackage 5.3, we will use the results gained from workpackage 5.2 to define the ORKA policy administration model, as a new administration model or as a combination of features from existing administration models.

References

- [1] Windows Use and Administration: Requirements for Dynamic Permissions in the Context of Windows, 2007. ORKA Deliverable AP 1.1.
- [2] Christopher Alm, Thomas Apel, Michael Drouineaud, and Mathias Kohler. Requirements for the Policy Language, 2006. ORKA Deliverable AP 2.2.
- [3] American National Standard Institute (ANSI) for Information Technology. Role based access control. Technical Report ANSI INCITS 359-2004, ANSI, Feb. 2004.
- [4] Rob Barrett, Yen-Yang Michael Chen, and Paul P. Maglio. System Administrators are Users, Too: Designing Workspaces for Managing Internet-Scale Systems. In *CHI '03: CHI '03 extended abstracts on Human factors in computing systems*, pages 1068–1069, New York, NY, USA, 2003. ACM Press.
- [5] Konstantin Beznosov. Human Factor in Security Administration: Brainstorming the Research Directions. presentation given at SEEDS, Dec 2003.
- [6] Konstantin Beznosov. Toward Usable Security Administration. presentation given at the 4th Annual Advanced Networks Conference, Vancouver, Canada, April 2004.
- [7] David Botta, Rodrigo Werlinger, Andr e Gagn e, Konstantin Beznosov, Lee Iverson, Brian Fisher, and Sidney Fels. HOT Admin: Human, Organization, and Technology Centred Improvement of IT Security Administration. In *Proceedings of the Annual Computer Security Applications Conference*, Miami Beach, Florida, 2006.
- [8] S. Bradner. Key words for use in RFCs to Indicate Requirement Levels, 1997. RFC 2119, <http://rfc.net/rfc2119.html>.
- [9] Sonia Chiasson, P. C van Oorschot, and Robert Biddle. A Usability Study and Critique of Two Password Managers. In *Proceedings of the 15th USENIX Security Symposium*, 2006.
- [10] N. et. al. Damianou. The Ponder Specification Language. 2001.
- [11] N. et. al. Damianou. Tools for Domain-based Policy Management of Distributed Systems. 2002.
- [12] Donald G. Firesmith. Engineering Security Requirements, 2003.
- [13] JBoss. JBoss jBPM - Product Page, 2006. <http://www.jboss.com/products/jbpm>.
- [14] Helen Maskery, Gord Hopkins, and Tim Dudley. Context: What Does It Mean to Application Design. In *ACM SIGCHI Bulletin*, volume 24. ACM Press, April 1992.
- [15] Jonathan D Moffett, Charles B Haley, and Bashar Nuseibeh. Core Security Requirements Artefacts, 2003.
- [16] OASIS. OASIS Web Services Business Process Execution Language (WSBPEL) TC, 2006. <http://www.oasis-open.org>.
- [17] openWFE. openWFE - Product Page, 2006. <http://web.openwfe.org/display/openwfe/Home>.
- [18] Taufiq Rochaeli and Ruben Wolf. Requirements on SicAri Security Policies, 2004.

- [19] Andreas Schaad, Volkmar Lotz, and Karsten Sohr. A model-checking approach to analysing organisational controls in a loan origination process. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 139–149, New York, NY, USA, 2006. ACM Press.
- [20] Ian Sommerville. *Software Engineering*. Addison-Wesley, seventh edition edition, 2004.
- [21] D. Thomsen, D. O'Brien, and J. Bogle. Role-based access control framework for network enterprises. In *ACSAC '98: Proceedings of the 14th Annual Computer Security Applications Conference*, page 50, Washington, DC, USA, 1998. IEEE Computer Society.
- [22] Wil M. P. van der Aalst, Marlon Dumas, and Arthur H. M. ter Hofstede. Web service composition languages: Old wine in new bottles? In *EUROMICRO '03: Proceedings of the 29th Conference on EUROMICRO*, page 298, Washington, DC, USA, 2003. IEEE Computer Society.
- [23] W. M. P. Van Der Aalst and A. H. M. Ter Hofstede and B. Kiepuszewski and A. P. Barros. Workflow Patterns. *Distrib. Parallel Databases*, 14(1):5–51, 2003.
- [24] WfMC. WfMC - Workflow Management Coalition, 2006. <http://www.wfmc.org/>.
- [25] Alma Whitten and J. D. Tygar. Usability of Security. A Case Study. Technical Report CMU-CS-98-155, Carnegie Mellon University, School of Computer Science, 1998.
- [26] Alma Whitten and J. D. Tygar. Wha Johnny Can't Encrypt: A Usability Evaluation of PGP 5.0. In *Proceedings of the 9th USENIX Security Symposium*, 1999.
- [27] Wil van der Aalst and Arthur ter Hofstede. Workflow Patterns, 2007. <http://www.workflowpatterns.com>.
- [28] William Yurcik, James Barlow, and Jeff Rosendale. Maintaining Perspective on Who Is The Enemy in the Security Systems Administration of Computer Networks. In *ACM CHI Workshop on System Administrators Are Users, Too: Designing Workspaces for Managing Internet-Scale Systems*. ACM Press, 2003.
- [29] Mary Ellen Zurko, Steve Chan, Greg Conti, and Konstantin Beznosov. Usability of Security Administration vs. Usability of End-user Security. Panel-Slides, Symposium on Usable Privacy and Security (SOUPS) 2005, Pittsburgh, July 2005.