

ORKA – Organizational Control Architecture



Deliverable AP 1.3

Organisational Control

Synchronization of Subject Areas

02.07.2007

Christian Wolter, Mathias Kohler, SAP AG

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-ap1.3.tex 744 2007-11-14 12:51:22Z kohler \$

Contents

- 1 Introduction** **4**

- 2 Synchronization Process** **5**

- 3 The ORKA Model** **7**
 - 3.1 Subject Area Dependencies 7
 - 3.1.1 Security Checks/Design Validation 7
 - 3.1.2 Specification Language for Policy Enforcement 8
 - 3.1.3 Specification Language for Validation 8
 - 3.1.4 Administration constraint definition 8
 - 3.1.5 Data-exchange interface between Validation and Administration components 8
 - 3.1.6 Policy storage interface 8
 - 3.2 ORKA Entities 9
 - 3.3 Consolidated ORKA Entities 9
 - 3.4 SPECification of the ORKA Model 12

- 4 Policy Translation Approaches** **14**
 - 4.1 Two-Step-Transformation 15
 - 4.2 One-Step-Transformation 16

- 5 Enforcement Architecture** **17**
 - 5.1 High Level Architecture 17
 - 5.1.1 Three layers for interaction 17
 - 5.1.2 Dedicated ORKA components 18
 - 5.2 Communication Channels 20
 - 5.3 Security Assumptions 20
 - 5.3.1 Authentication 21
 - 5.3.2 Integrity 21
 - 5.3.3 Confidentiality 21
 - 5.3.4 Privacy 22
 - 5.3.5 Web of Trust 22
 - 5.3.6 Scope for ORKA 23

- 6 Conclusion** **23**

1 Introduction

The ORKA project is separated into five different subject areas: CONTROL, SPEC, ENFORCE, VALID, and ADMIN. Every area contributes to the overall project but concentrates on a particular aspect of the project by having a specific point of view:

- **SPEC**
Several definition of static and dynamic authorization constraints are identified in the literature, such as operation separation of duty, case handling, permission delegation, and authorization revocation. Therefore, one of the major tasks of the subject area SPEC is to define a taxonomy covering all these aspect. Based on this taxonomy interfaces will be defined enabling an administrator to define such security constraints accordingly. To define the interfaces the taxonomy is a starting point to develop or extend an existing formal security specification language. It must be evaluated to what extend already existing specification languages already fit onto the developed taxonomy under consideration of usability aspects.
- **ENFORCE**
The subject area ENFORCE focuses on the development of authorization component such as policy enforcement and policy decision components. These components must be integrated into a process-aware information systems and must follow a service oriented paradigm. Best to our knowledge exists no reference implementation that is capable of handling various types of authorization constraints. To support platform independence the whole architecture is implemented by the JAVA programming language. To enable the integration of legacy systems and applications security functionality is not directly integrated into each system, but will be made available as a security service.
- **VALID**
The subject area of VALID, covers the aspect of analyzing defined security polices to reveal inconsistencies and conflicts. Such validation becomes difficult to be performed manually, when the policy complexity increases. Therefore, the subject area of VALID addresses this problem by defining formal mechanisms to automatically analyze defined security policies for inconsistencies and gives immediately feedback at design-time to avoid errors and deadlock in a productive system. The analysis will be based on formal and semi-formal approaches such as UML/OCL.
- **ADMIN**
The last subject area ADMIN covers the aspects of tool supported policy definition, management, and analysis. To reduce the complexity of security management an administrative interface is necessary to support an administrator to define security policies and to trigger policy validation. Therefore, dedicated user interfaces are developed to provide the necessary information of available service, organisational information, and policy repositories. The user interface is a central place for all administrative activities. Each policy modification shall be automatically verified and simulated to allow immediate feedback to reduce administrative mistakes.

However, these areas must not be treated separately as there are dependencies among these areas. Throughout the project it is therefore necessary that the dependencies between all areas are determined and their intermediate results are considered as input for related work packages during the whole project.

This work package is part of the area CONTROL which basically consolidates the other subject areas. It is the first of a series of work packages with the goal to initiate and strengthen the cooperation during the planning, design, and implementation phases of the other areas SPEC, ENFORCE, VALID, ADMIN, as well as CONTROL itself. To accomplish this goal, we will focus on three different elements we introduced to initiate and document the interactions between the fields and their respective work packages.

The second element is the identification of interfaces between each of the fields. This includes the determination of the dependencies between all fields. A dependency reflects that certain deliverables within one field require some of the outputs of other fields. Especially in cases where such dependencies are not obvious they should be determined and recorded in an early stage such that they can be considered for design decisions.

The second element for the coordination in this project is to provide a common understanding of the underlying architecture of the overall project. Although the architecture's development is mainly done in the field of ENFORCE, all other fields highly depend on it. For instance, the exchange of data between the components developed in each field must be defined in terms of protocols and security requirements. This document will help to centralize the architecture's main which was developed during the last months such that every partner can rely on a common basis and develop the components comprising ORKA by having this overall picture in mind.

2 Synchronization Process

All subject areas started with some initial work packages investigating and collecting sets of requirements that have to be fulfilled by the organizational control architecture ORKA. These requirements can be divided into expressiveness and complexity of the policy specification language, formal aspects, such as consistency analysis and validation, as well as technical and administrative details, such as the support of a distributed system landscape, decentralized policy repositories, and fine-grained administrative policies. In what follows we give an overview of the process that is used to align the requirements of the different work packages by collecting all identified requirements and generate an abstract consolidated list of requirements in a central place. In the scope of this work package we define an overall model that integrates all requirements into an ORKA security model. This model will be developed by incorporating all partners in the model definition phase.

This model will be used as an input for work packages of the subject area SPEC, ENFORCE, VALID, and ADMIN. This will guarantee that the policy specification language is mostly aligned with the overall model. It is in the scope of this work package to define an ORKA model that forms the basis of the developed policy specification language that is part of the subject area SPEC. This means that all entities of the policy specification language and relations between them can be mapped onto entities and corresponding relations of the ORKA model that is sketched out in this document.

Next, it is to make sure that an instance of the policy specification language model, meaning a concrete set of security policies can be deployed into the ORKA system architecture and be enforced at runtime. On the other hand, an instance of the policy specification language model has also to be transformed into formal language representation in terms of validation and consistency analysis of specified policies. This transformation capability of the policy specification language is achieved by defining model transformation description between the policy specification language and there related counterparts for the validation and enforcement of the security policies. Two possible approaches will be discussed in Section 4.

The following Figure 1 depicts the overall process we apply to synchronize the work efforts of the different work packages in order to create the overall ORKA model and the alignment with the model of the policy specification language within the context of work package 1.3.

To generate a model bases for the specification language, all partners of the ORKA project agree, on we describe the process we are implementing to develop the model. Therefore, we separate the overall process into three major steps. In the first step we collect all requirements identified in the various work packages completed so far. These work packages are namely work package 1.1, 2.1, 2.2, 2.3, 3.1, 4.1, 5.1, and 5.2. Based on these requirements an abstract meta-model of the overall ORKA security model will be developed. This model will be refined and iterated until all partners agree on it.

In the second step we define a mapping from the ORKA model including the RBAC specification [4, 8, 10, 1, 9] to the corresponding model of the security policy specification language that will be utilized in ORKA. One important aspect of policy language specification is its capability to be transformed into a formal model to validate and analyze security policy specification instances. This is directly related with the ongoing work of the work packages 2.3, 2.4, 4.2, and 4.3. A third parallel step targets the alignment between the ORKA model and the architecture and technical concepts of the ORKA infrastructure (cf. WP 3.2, 3.3, and 3.4). It will be assured that the developed architecture is suitable for the enforcement of security policies. In the course of the ENFORCE work packages a translation from thee ORKA security policy language to a technical representation will be defined. One possible translation could be towards the eXtensible Access Control Markup Language XACML [12, 7, 2, 13, 11].

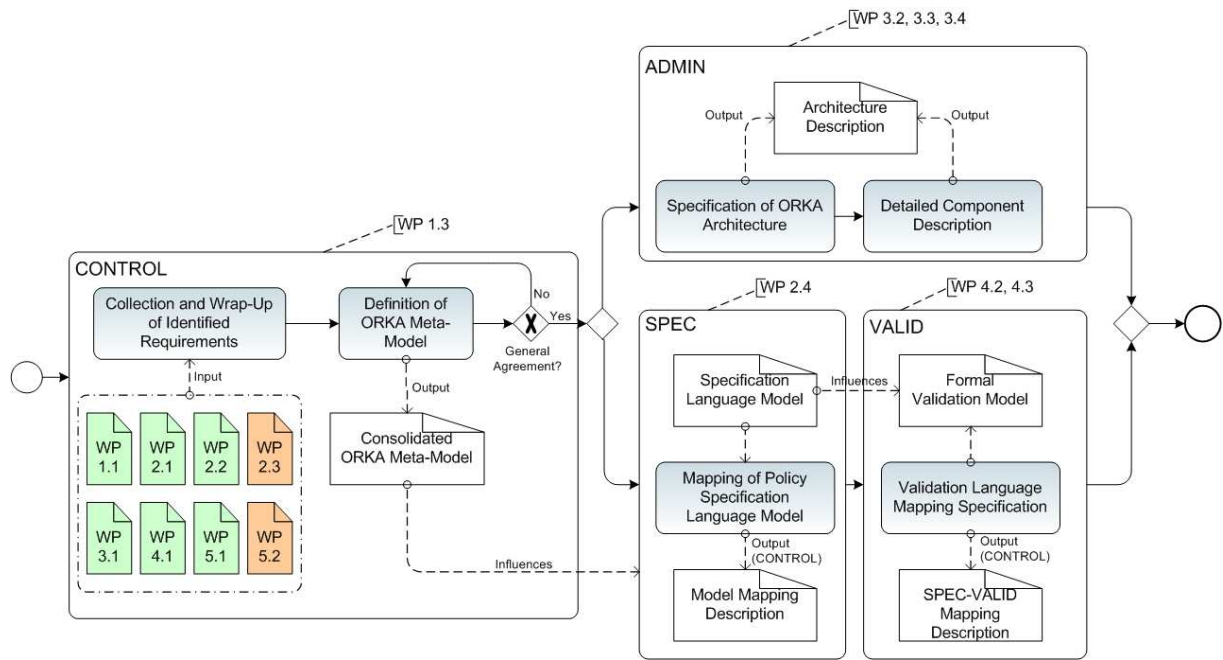


Figure 1: WP 1.3 Control Process

3 The ORKA Model

3.1 Subject Area Dependencies

A first collection of interfaces was compiled at the meeting of the consortium in April 2007. The following items already represent an extended collection of items. Every interface description is comprised of a title, a list of concerned fields, and a textual description.

3.1.1 Security Checks/Design Validation

Concerned fields:	ALL
Description:	Design and development of component architectures within the project which has potentials for security leaks should be announced to the partner EUROSEC which within the context of WP 1.9 will make a security analysis of the presented design.

3.1.2 Specification Language for Policy Enforcement

Concerned fields:	SPEC, ENFORCE
Interface description:	The policy specification language must be transferable into a language used for the policy evaluation by the policy decision point.

3.1.3 Specification Language for Validation

Concerned fields:	SPEC, VALID
Interface description:	The policy specification language must be transferable into a language used for the policy validation.

3.1.4 Administration constraint definition

Concerned fields:	ADMIN, SPEC, ENFORCE
Interface description:	The administration needs the possibility to specify administrative rights for users. This affects the specification language as it should provide such a possibility as well as possibly the PEP and PDP to enforce the specification.

3.1.5 Data-exchange interface between Validation and Administration components

Concerned fields:	ADMIN, VALID
Interface description:	There must be an interface specified between the validation component and the administration component, such that it is possible to initiate a policy validation as well as to get the results of a validation.

3.1.6 Policy storage interface

Concerned fields:	ADMIN, ENFORCE
Interface description:	There must be an interface specified between the storage component for policies specified for the PDP and the policy management component for administrators to manage policies.

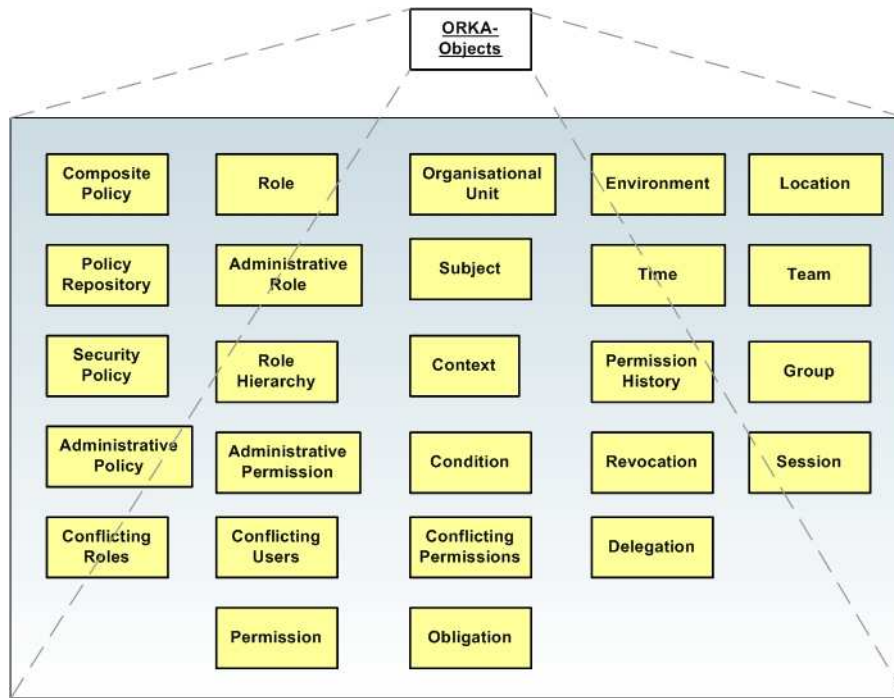


Figure 2: Entity Pin Board

3.2 ORKA Entities

To support the development process of the overall ORKA model we provide a first collection of brainstorming buzzwords (cf. Figure 2). This collection is the base of the ORKA security specification language model along with the RBAC specification and underlies some ongoing discussion with the whole consortium.

3.3 Consolidated ORKA Entities

Based on an initial discussion we defined a set of ORKA entities. In what follows we describe each entity and provide an entity relationship diagram describing the dependencies between the different entities.

- **Subject**
A subject is an entity (human or process) within ORKA that performs actions on resources. Subjects may be conflicting in case of common criteria. Conflicts can only be defined if there is semantic interpretation of subject attributes and relationships in place.
- **Subject Group**
A subject group is a collection of subjects. Subject groups may contain conflicting subjects. A subject group is the smallest collection of entities a single security policy can be applied to.
- **User**
User represent a specialization of subjects. User are human subjects interacting within the ORKA architecture.

- **Process**
They represent a specialization of subjects. A process impersonates a user. Meaning, a process interacts with the system on behalf of an existing user. Every process must impersonate a system user.
- **Permission**
A permission is one of the central building blocks of security policies within the ORKA model. Based on their attributes, permission may be conflicting with other permissions. Permissions comprise of an object, a related operation, optional constraints, and an assigned target subject group to that this permission applies to.
- **Object**
An object represents a concrete protected resource within the ORKA model. By protected we mean access to the object, in order to perform any kind of activity on it, is only granted if a security check is performed before. Objects can be found on different abstraction and semantic levels within the system. The different levels are represented by domains (cf. Policy Domain). For instance on file system level files and directories are considered objects. In a database systems tables, rows, columns, and cells are considered objects.
- **Operation**
An operation is an activity that can be performed on an object. Similar to objects, the operation that can be performed on a protected object depends on the domain of the objects. For instance, example operations for file system objects are *read*, *write*, *delete*. Each operation is assigned to exactly one object per permission.
- **Constraint**
A constraint is a conditional entity consisting of a single expression and arbitrary contextual information. Again, the contextual information is domain dependent. Depending on the domain there exists a huge set of different contextual information types, such as process history, session information, local time, or location. The number of context information types shown in Figure 3 is not complete, as indicated by the «*context*» stereotype entity. The supported kinds of context information depend on the supported domains of the ORKA system. A constraint returns either *true* or *false* depending on the evaluation of the expression.
- **Expression**
An expression evaluates a concrete contextual information against a given value. For instance an expression could validate if the current time is less than 6 p.m. and returns *true* in case the context information fulfills the expression or *false* in case it does not.
- **Policy Domain**
As described earlier a policy domain represents a specific aspect within a system environment. A policy domain may be considered a complete ontology description of a given system environment. For instance a policy domain ontology for business processes would describe what a business process is. For instance introducing the terms control-flow, data-flow, and resource perspective. In addition, it describes which objects can be found in this domain and what operations can be performed for each object of that domain.
- **Administrative Policy Domain**
Each administrative domain is part of a single policy domain. It defines a subject of the overall operations of the policy domain, by defining dedicated operations on objects

marked as administrative. For instance an administrative operation might be the assignment of process tasks to be executed by a subject holding a specific role.

- **Permission Group**
A permission group is a collection of permissions. A permission group may contain conflicting permissions that conflict with permissions in other permission groups. There must not be conflicts of permissions within the same permission group. Permissions groups are assigned to roles.
- **Role**
Roles represent organisational structures. Subject groups can be assigned to one or more roles. Therefore, different subject groups within a single role allow a horizontal fragmentation of a single organisational role. Role hierarchies are expressed by a partial order between the role entities. Role hierarchies are interpreted in terms of superiority.
- **Delegation Object**
A delegation object is an abstract representation of a role or permission group. That means it is nothing more than a collection of permissions that can be temporary assigned to another subject group.
- **Delegation**
A delegation is the temporary reassignment of permission collections (i.e. permission group, roles) to another subject group. There will be no "Mehrfach-Delegation", and no delegation for a permission which user does not possess.
- **Revocation**
A revocation depends on a previously occurred delegation of permissions. Each delegation can be revoked once. Revocation means, the deletion of a temporary permission collection after a specific condition is met.
- **Policy Object**
Within the ORKA architecture permissions are administered in terms of policy objects. A policy object contains an arbitrary collection of permission groups, whereby all permission groups must contain permissions related to the the same policy domain.
- **Composite Policy**
A composite policy combines several policy objects. Policy objects must not be assigned to the same policy domain. Meaning a mixture of different policy object domains is possible (e.g. a file system policy combined with a business process policy). Each policy object is evaluated separately. If the overall evaluation of all permissions of each policy object results in contradictions a conflict resolution strategy is used.
- **Conflict Resolution Strategy**
This entity is basically a rule describing how to deal with contradicting permission results (i.e. a combination of true and false values). For instance a potential strategy might be *false overrides*.
- **Policy Repository**
Policy objects and composite policies are stored in policy repositories that may be distributed over the ORKA architecture. A policy repository provides the functionality for handling policy requests, storing policies and managing policies (e.g., through an admin interface).

- Meta-Policy

A meta policy contains additional information for each composite policy, such as version number, author, last-modified date or other information necessary for policy authors.

The described entities and their relations to each other can be found in the following Figure 3:

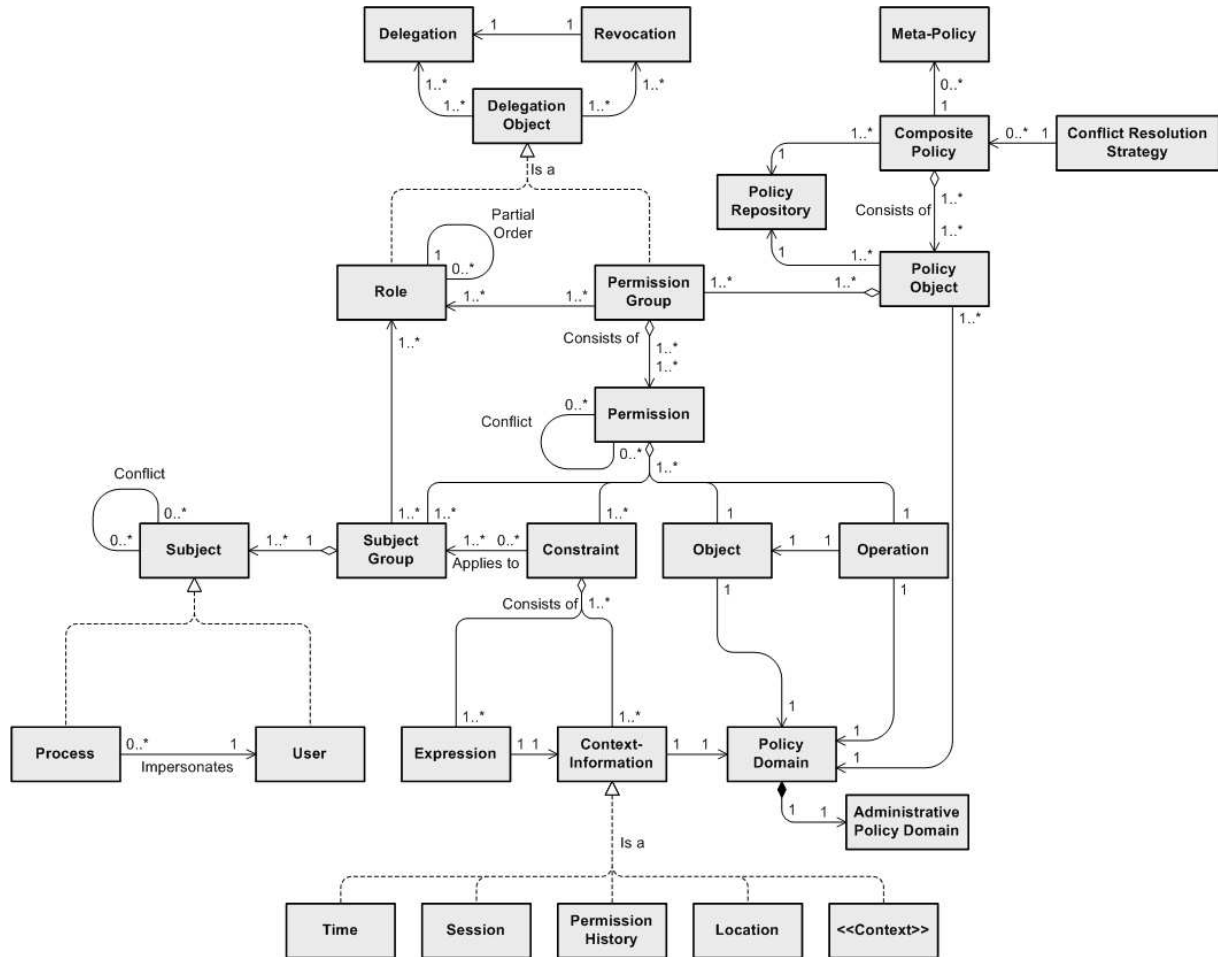


Figure 3: ORKA Model

3.4 SPECification of the ORKA Model

Based on the initial discussion about some of the entities and their relationships of the overall ORKA policy model (cf. 3) where all consortia partners were involved we developed a first conceptual model of the security policy language that is going to be defined in the work package 2.4. From this starting point, the subject area SPEC is going to further enhance this initial model with respect to the fundamental principles of the RBAC authorization model enhanced by the requirements identified and documented in the related ORKA work packages 1.2, 2.1, 4.1, 5.1, and 5.2. In the course of the work package 2.4 concepts will be developed to transform the ORKA model into suitable representations for the areas of administration, validation, and enforcement:

- **ADMIN**

Policy administration benefits from a volatile and lightweight memory or object model for the administration tools to enable an agile modification of security policies during the policy specification. The primary concern is to develop an approach to easily create and modify security policies without working directly creating the final enforcement or validation model policies. Using the enforcement policy model for instance would mean working with some kind of files (e.g., XML-based) or database persisted policies during the administration resulting in extensible parsing and writing activities through the whole administration process. Therefore, persisted policies will be read from a repository at administration startup into the memory and will only be written back to file if the administrator completes his work.

- **VALID**

For the validation of defined security policies a formal representation must be chosen to allow model checking and other formal methodologies to verify and validate ORKA policies with existing tools available. One potential target platform for the validation of ORKA policies is the Isabelle, a generic theorem proving environment [cite]. The model might not be suitable for administration or runtime enforcement purposes. Therefore, this formal representation will only be used during the validation of security policies.

- **ENFORCE**

Domain-independent and non-proprietary policy formats allow the deployment of the ORKA security architecture and the ORKA security model in loosely-coupled service-oriented architectures and legacy system integration scenarios. Meaning, the ORKA policies will be persisted, transported, and enforced using one or more de facto industry standards security policy communication enforcement languages, such as of XACML, WS-PolicyConstraints policies [3], SAML [5], or COPS [6]. This allows to later extend and integrate other XACML- and WS-* enabled policy enforcement and decision components into the ORKA security architecture landscape. This is achieved by relying on industrial standards rather than on self-made proprietary specifications.

Based on this idea of model transformation the following Figure 4 visualizes the identified dependencies between the different work packages and outlines their idealized temporal ordering. Starting from the discussion given in this document work package 2.4 defines the ORKA security policy specification languages containing the discussed entities and relationships of this document while integrating the major concepts of the RBAC authorization model. In parallel to this activity the specification of the other models in the area of ADMIN, ENFORCE, and VALID take place. As part of the work packages 2.5 and 2.6 model transformations will be defined to transform an ORKA security policy into the model of the administration, into an Isabelle Theory, and XACML policies for runtime enforcement. To achieve this policy translations two approaches are feasible that will be discussed in the successive section 4.

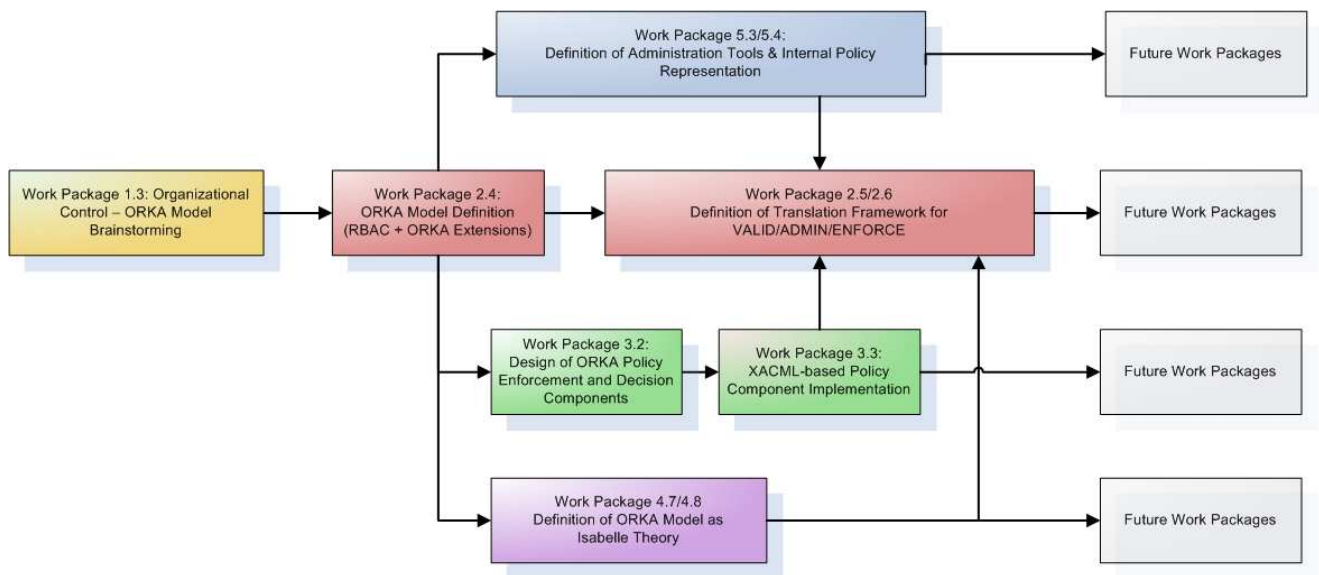


Figure 4: Work Package Dependencies

4 Policy Translation Approaches

In the last section we outlined the idea of model transformation to allow each subject area to create and use tools that utilize a specific domain-dependent model. A domain-dependent model is tailored for a specific problem and eases the creation of related tools. In the course of the ORKA project three different domain-dependent policy models and one domain-independent policy model are used to administrate, validate, and enforce security policies that are specified in the context of the domain-independent ORKA security policy specification language. Some background information about model transformations can be found in [14].

In this section we discuss two possible approaches to achieve the translation from the domain-independent ORKA security policies into the domain-dependent model for ADMIN, ENFORCE, and VALID.

4.1 Two-Step-Transformation

Using this transformation sequence we represent the ORKA model without complex transformation directly for instance as memory objects or some graphical objects within the administration tools. The concrete implementation of the administration model depends on which approach is the most suitable one. This should be possible because the administration tools are hand-tailored and therefore can be written to directly support the ORKA model. In the case policy administration finished and stored the policies are transformed into XACML++¹ distributed to the policy repositories and stored in a database or file system. To perform this transformation from the administration model to the enforcement model a specific model mapping is defined to map each administration entity onto an enforcement entity. In the two-step scenario we define a second mapping from the enforcement model onto the validation model. In this scenario the second transformation uses the XACML++ policies as input and generates a formal representation of the policies in order to validate them. The Two-Step-Translation, along with its related transformation mappings, is shown in Figure 5.

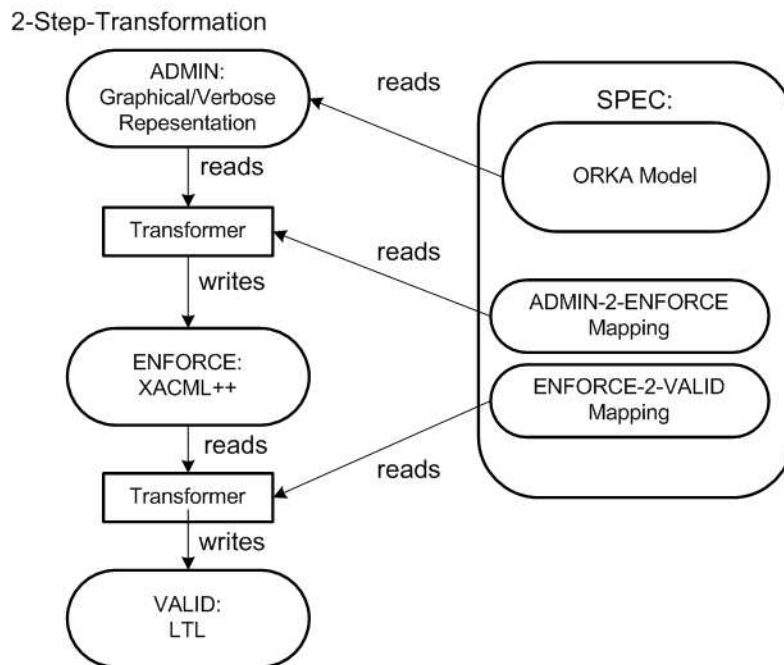


Figure 5: Two-Step-Transformation

This scenario while possible from a technological point of view, lacks two important aspects that are important for the overall success of the ORKA project:

- The transformation from the XACML++ model to the valid model requires a complete specification of the XACML++ model beforehand. On the other side it is not clear if the administration model can be translated completely into XACML++ policies. In this case only a subset of security policies could be validated.
- This temporal dependency would slow down the proceeding of the VALID work packages because they can only begin if the XACML++ policy representation is completely specified. In general, slowing down the complete ORKA project performance.

¹XACML++ is an ORKA specific profile for XACML

4.2 One-Step-Transformation

Because of the drawbacks of the first discussed approach we decided to follow an one-step scenario. In this scenario the administration model is still based directly on the general ORKA security policy model, but in this case the related enforcement and validation models are directly related from the administration model in contrast to the two-step approach. This approach is more suitable with respect to the time constraints of the ORKA project. Figure 6 depicts the discussed approach based on a one-step transformation originating from the administration model directly into the other two domain-dependent models.

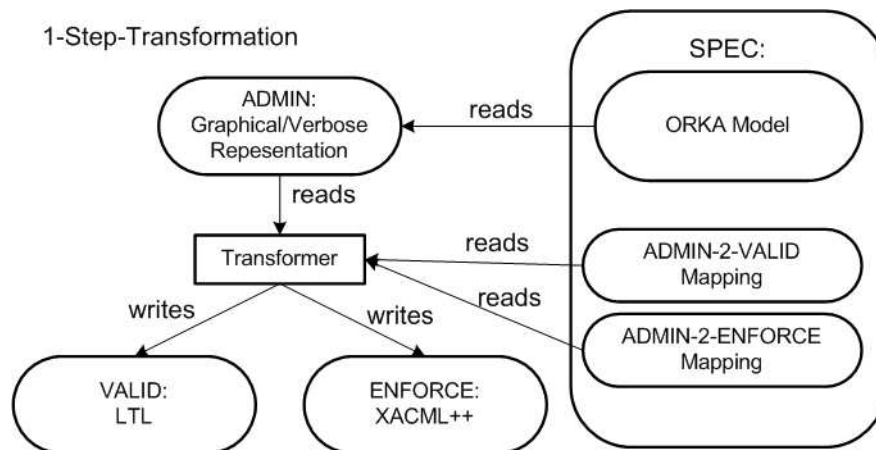


Figure 6: One-Step-Transformation

This approach has two major benefits:

- First, the fact that the enforcement and validation mapping depend from the administration model allows to specify both mappings in parallel, allowing to immediately start the work in both subject areas as soon as the administration model is completely specified.
- Second, the expressiveness of neither the enforcement model nor the validation model depend on each other. This means even if not all security policies defined by the ORKA model can be translated into related XACML++ policies, it might be still possible to express them all in the context of the validation model.

5 Enforcement Architecture

This section describes the architecture of the ORKA project in a general, high level perspective. The architecture presented is meant to be a reference for the concrete architectural design in work packages of the other subject areas, especially ENFORCE. Besides, this document also references the communication channels and gives a first draft according to the security goals to be reached for the data exchange between the single components.

5.1 High Level Architecture

This section presents a high level overview about the general architecture of ORKA. An accompanying illustration is given with Figure 7. We basically distinguish between 3 different layers: the User Interface Layer, the Business Layer, and the Application Layer. We will go through each of them in the following paragraphs, describing their role in the architecture.

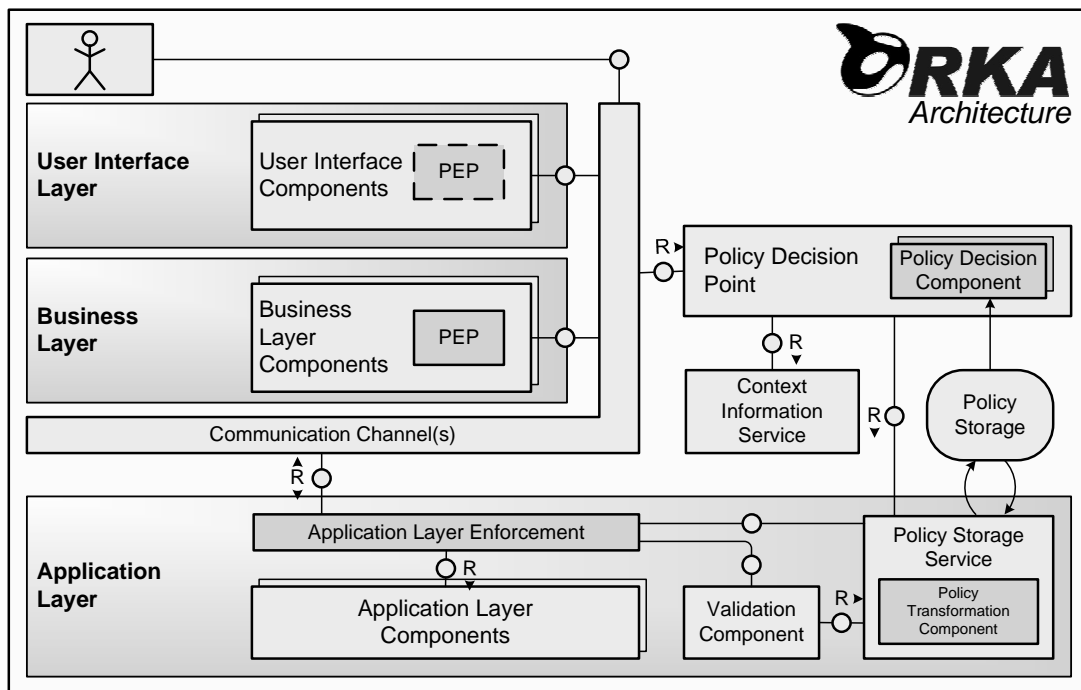


Figure 7: ORKA Overview Architecture

5.1.1 Three layers for interaction

The User Interface Layer comprises all components relevant for user interactions. In a business process environment, for instance, a user interface component could be the worklist which lists a user's tasks she is assigned to. Another example is the administration interface for managing the ORKA access control policies.

The **user interfaces** offer the functionality exposed by the underlying layers to the users. If the user requests to perform an action, these requests are directly communicated to the respective components of the Business Layer in case of a business process environment or to components of the Application Layer in case of actions without business process or workflow context. There is the possibility to implement enforcement techniques into the user interface components to prevent users from executing secured actions. In this case the user interfaces' enforcement component would send access requests to the evaluation component (Policy Decision Point (PDP)). Independent whether enforcement is done on the interface layer, there are additional enforcement components in the underlying layers.

The **Business Layer** comprises components introduced to manage and control related tasks and actions in a business process environment (e.g., order of event). A possible scenario such as shown in work package 3.2 is the usage of a workflow engine to take care a set of related tasks to be performed in a predefined order and, hence, realizing the execution of a business process. The components on this layer implement policy enforcement points to prevent unauthorized task executions.

Access requests (created by the policy enforcement points (PEP)) are sent to the PDP using the communication channel. The presented architecture leaves it open to the individual implementations how the communication between the PEP and the PDP is actually realized. This equally holds for the communication between components of the different layers. In Figure 7, the communication is therefore generally illustrated by a Communication Channel(s) component. A more specific architecture using an enterprise service bus (ESB) for communication between the components is, for instance, presented in workpackage 3.2.

The **Application Layer** comprises the components which provide the actual system's functionality. In other words, these components provide the functionality which is requested by either the business layer or the user interface layer. Examples are the internal functions of business objects for their manipulation, usually exposed by web services, or externally provided web service-based methods provided by other legacy systems.

Components of the Application Layer are either called by components of the User Interface Layer or by components of the Business Layer. Each call goes through an intermediate Application Layer Enforcement component and, hence, generates an access request. This ensures that only authorized components (on behalf of the interacting user) have the permission to call exposed functionality of the application layer components.

5.1.2 Dedicated ORKA components

Besides the more general components (user interface, business process realization, or legacy elements) the architecture also depicts several dedicated ORAKA components; namely Policy Decision Point (PDP), Policy Storage Service (PSS), Policy Storage, Validation Component, and Context Information Service (CIS).

Policy Decision Point: The PDP is the component responsible for evaluating access requests. It acts as central point of contact for any access request.

The evaluation of a request needs policies and policy rules. ORKA policies can be separated into different policy classes (see work package 3.2 for details). Each class has an individual set

of constraint types that can be expressed and each class also can be implemented in a different policy language. To support this flexibility the ORKA architecture introduces multiple Policy Decision Components (PDC). Each component belongs to the central PDP, but provides the evaluation logic for a certain policy class and therefore is capable to evaluate policies written in their respective policy language. The PDP coordinates the complete evaluation process, combines different results of its PDCs, and eventually sends a final decision back to the requesting PEP.

Policy Storage Service: The Policy Storage Service is responsible for managing the access control policies. This comprises two main tasks. The first one is, in case of an access request, the PDP needs the policy that specifies the rules to evaluate the request. The Policy Storage Service provides the policy by selecting it from the Policy Storage.

The second task concerns the policy administration. The Policy Storage Service is responsible for storing policies adequately. Generally, new policies are designed using an ORKA specific policy model. However, to store such a designed policy, the Policy Storage Service has to transform the policy into a storable format and, hence, into the policy class' specific policy language. The transformation task itself is performed by a Policy Transformation Component which is part of the Policy Storage Service.

One of the goals of ORKA is not only control access on executing actions on the application layer but also controlling administrators when administrating policies. The administration of policies is done via a user interface component and in this case, the Policy Storage Service is treated as being a component on the application layer offering a service to the user (e.g., "create new policy"). Hence, any action to be executed on the Policy Storage Service is restricted to access control policies as any other component on the application layer as well. This gives the possibility to apply the whole range of available access control policy restrictions used for regular applications (e.g., including SoD constraints) for the administration of policies, too.

Validation Component: For the Validation Component also applies that it is part of the Application Layer. It should mainly be called by the administration user interface, triggering the process of validating newly designed as well as already created policies. Policies are directly retrieved from the Policy Storage Service in a format the validation component expects it.

Context Information Service: The Context Information Service retrieves context information during runtime which is needed to process a policy request and evaluate corresponding policies. It should be connected to further information provider of the system to query the needed information (not shown in Figure 7).

Policy Storage: All policies are stored in their respective language according to the policy class they belong to in a separate location. The location is the policy storage which is accessed by the two type of components, the Policy Storage Service and policy decision components. The Policy Storage Service accesses it for administrative reasons; policy decision components retrieve the policies they need for the evaluation process.

5.2 Communication Channels

The components in the ORKA architecture are communicating. This section lists the communication channels and gives the main aspects for each of the interactions. Details about an exemplary implementation design can be found in workpackage 3.2.

- **PEP → PDP:** This channel transmits permission queries from the PEP to the PDP and returns the permission response from the PDP's access evaluation. A connection is initialized by the PEP.

Information exchange: Subject information, Information about the intended Action and intended Target for the request. The response contains the evaluated access decision (invalid or incomplete request, access granted, access denied, access decision failure (temporarily), access decision failure (permanent)).

- **PDP → PDC:** The PDP calls the PDC for evaluating an access request.

Information exchange: The PDP forwards the access request, the corresponding policy objects as well as a link to the Context Information Service (CIS). The response for the evaluated request comes from the PDC. It consists of one of the given response types listed above (for the PEP → PDP connection).

- **PDC → PEP:** For business context information a PDC will directly connect to the PEP to get the needed information.

Information exchange: Request for needed context specifying the type of the context information required. The respective response.

- **PDP → PSS:** The PDP retrieves policy objects from the Policy Storage Service.

Information exchange: The PDP transfers Target and Action values from the request to the storage service and receives the relevant policy objects.

- **PDC → CIS:** The PDC connects to the Context Information Service (CIS) for the retrieval of needed context value resolution.

Information exchange: The PDC sends a request for a context attribute value and receives the values.

- **Administration Interface → ORKA Components:** The specification for the interaction between the Administration interface and the ORKA components are currently under development (cf. workpackage 5.3) and will be summarized in workpackage 1.4.

5.3 Security Assumptions

Many ORKA components exchange messages on which eventually access control decisions and enforcements are based. Therefore it is important that certain security measures are applied to prevent intentional malicious or accidentally performed actions.

In the following sections we will provide a summary about the security assumptions applying for the ORKA architecture. As XACML has a similar request/response architecture for access control enforcement, we used [7] as starting point.

5.3.1 Authentication

With authentication we understand means to determine the identity of interacting components. It is important that ORKA dedicated components when interacting identify each other, considering the following reason on an exemplary connection between the PDP and the PEP.

When interacting, the PDP must be identified to the PEP and vice versa. The reason for a PDP identification is to limit the risk for false or invalid access decisions to be returned by an attacker which would lead to access enforcement based on an attacker's decision. The reason the PEP must be identified as requester to the PDP is to limit the risk that an attacker gets information from making (possibly unlimited) "fake" requests and receiving an evaluated response.

The same holds for interactions between PDP and its PDCs, PDP and Policy Storage Service, as well as Validation component interacting with the Policy Storage Service.

Also, it should be unambiguously clear that the stored policy objects have been implemented by an authorized subject, e.g. by using digital signatures.

5.3.2 Integrity

With integrity we understand means that messages transferred between different components can be recognized to be altered. Altered messages might happen by chance due to faulty communication channels or intentionally by an attacker (e.g., by a man-in-the-middle attack). Integrity is necessary to avoid acting based on unwanted and possibly unknown modified data. Otherwise, an attacker could, for instance, replace an access response which is denying a permission by a response which approves it without being discovered.

Integrity must also hold for stored policies such that it is possible to determine unintentionally altered or replaced, newly inserted, and deleted policies.

5.3.3 Confidentiality

Confidentiality provides measures to ensure that only intended subjects/components can actually read the content of a transferred message or the data stored in a dedicated place (e.g., policy storage).

In general, the information to be transferred between the components should be subject to confidentiality measures. Examples are the access requests and their corresponding access responses. Also, the context information itself requested by a subject from a context provider and transferred between components might be of interest to an attacker and should therefore be transferred in a confidential way.

The idea to keep such data confidential is to make it attackers as hard as possible to get information which they can use to analyze, finding a way to get unauthorized access. However, it is not recommended to solely rely on the secrecy of policies [7].

5.3.4 Privacy

Privacy means that user actions are kept confidential such that an attacker listening to component interactions may not draw conclusions with respect to the identity of the user or to any other access control relevant information.

5.3.5 Web of Trust

The trust model specifies which components in the ORKA architecture are depending on each other. For instance, the PDP relies on the Context Manager to provide the requested context information, whereas the Context Information Service itself relies on the information providers it queries for context information. Otherwise, a decision would be based on uncertain information. The following list gives the interdependencies of the components of our architecture. All (direct) dependencies are also depicted in Figure 8.

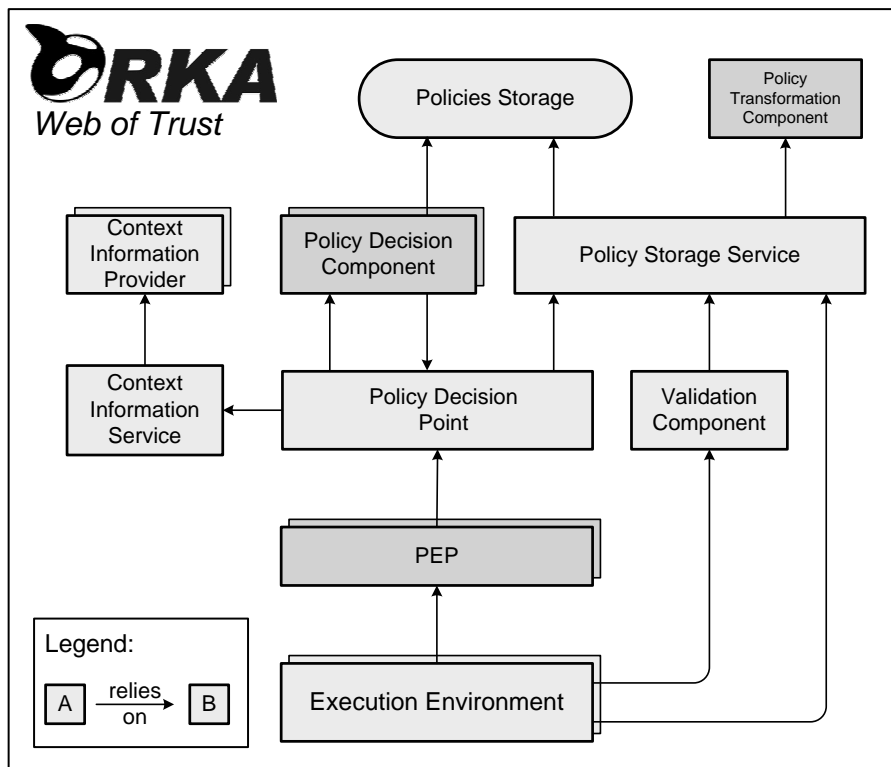


Figure 8: ORKA Web of Trust

Execution Environment: It is representing the complete system a subject is working with. Its correct functionality relies on the PEP's capability to enforce access control decisions. The execution environment is also used for policy administration. It therefore relies on the correct ability of the Policy Storage Service to store the policies as well as on the validation component to correct analyze and report invalid configured policies.

Validation Component: The validation component relies on the Policy Storage Service delivering the correct policies for the validation process.

Policy Enforcement Point (PEP): The PEP relies solely on the PDP to deliver a valid response for a sent access request by correct evaluating the necessary policies.

Policy Decision Point (PDP): The PDP's correct functionality is dependent on multiple other components, namely the PDCs to reliably process for evaluation forwarded policies, the Context Information Service to deliver the requested context information, and the Policy Storage Service to retrieve the appropriate policies needed to process an access request.

Context Information Service: The Context Information Service obviously must rely on the information given to it by the Context Information Providers. (Not shown in the graph: A PEP might also act as Context Information Provider).

Policy Decision Component (PDC): The PDCs rely, firstly, on the PDP forwarding correct information about the policies location (storage) and secondly on the policy storage to provide the requested policy objects.

Policy Storage Service: The Policy Storage Service also needs a reliable Policy Storage such that policies are appropriately stored for later use. It also depends on the Policy Transformation Component to appropriately transform the policies into their respective format.

5.3.6 Scope for ORKA

In section 5.3 we presented security assumptions and requirements, which apply for the ORKA architecture. Each of the items mainly describes what a final implementation is expected to address with respect to authentication, integrity, confidentiality, etc.

There are several different security measures available on a technical level to eventually realize the above mentioned security assumptions. Confidentiality, for instance, could be realized with applied cryptography, e.g. by using SSL-encrypted communication channels. The measures finally used for an implementation, however, are highly dependent on the environment where ORKA will be placed in.

Section 5.3 is meant to highlight that there are technical security requirements which have to be addressed when ORKA is integrated into a system landscape. We will further address this topic in workpackage 1.4 by giving examples of security measures for some of the above mentioned assumptions to realize an ORKA implementation. An explicit specification of security measures to address each of the above mentioned security assumptions, however, is for the ORKA project out of scope.

6 Conclusion

The primary goal of this work package is the overall alignment of all consortial partners. Starting from a general discussion about the topics of the different subject areas we outlined how the cooperation between all partners will be driven by our proposed synchronization process. Therefore, we showed the dependencies between the work packages of the near future and what kind of results of previous and parallel work packages they expect and what results they will produce.

As a starting point,altogether, we developed a first conceptual model of the ORKA security model. We defined a general terminology for the most important entities of our model with the goal to define a common language for our meetings and phone calls. In Section 3.3 and 3.4 we discussed the dependencies between the ORKA model and the domain-specific models of the different subject areas. Thus, we present a list of so called interfaces between the domain-specific models, such as validation of administered policies or the transformation of design-time policies into runtime policies.

The problem of domain-specific models and their suitability was discussed in section 3.4. We outlined that the driving force will be the subject area of SPEC. Based on their developed security model mappings and transformations between the domain-specific models of validation, administration, and enforcement will be defined. Further, we discussed potential model transformation strategies and finally chose a one-step transformation because of a beneficial independency between the three domain-specific model transformations.

In Section 5.1, we gave an abstract overview of the ORKA architecture, the contained components, what kind of communication channels exist, and potential information exchange that takes place between them (cf. Section 5.2). In Section 5.3, we gave a number of general security assumptions along with potential technologies that address these issues. This section is intended to outline various security topics that should be considered in ORKA and potential technological solutions. Nevertheless, we delay the decision which issues are within the scope of the ORKA project and which are not for a future work package.

In general, we consider this work package as an overall reference point for the course of the ORKA project and as a base for future CONTROL work packages.

References

- [1] American National Standard Institute (ANSI) for Information Technology. Role based access control. Technical Report ANSI INCITS 359-2004, ANSI, Feb. 2004.
- [2] Anne Anderson. *XACML Profile for Role Based Access Control (RBAC)*, 2004.
- [3] Anne H. Anderson. Domain-independent, composable web services policy assertions. *policy*, 0:149–152, 2006.
- [4] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control*. Computer Security Series. Artech House, Boston, 2003.
- [5] Harold Lockart. Demystifying saml. <http://dev2dev.bea.com/pub/a/2005/11/saml.html>, 2005.
- [6] Seppo Mallenius. The cops (common open policy service) protocol. *Research seminar on IP Quality of Service*, 2000.
- [7] Tim Moses. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. OASIS Standard.
- [8] Gustaf Neumann and Mark Strembeck. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM Transactions on Information and System Security*, 7(3):392–427, 2004.
- [9] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based access control models. *IEEE Computer*, 29(2):38–47, February 1996.
- [10] Mark Strembeck. <http://wi.wu-wien.ac.at/home/mark/xoRBAC/index.html> (2006-10-16).
- [11] Sun Microsystems, Inc. <http://sunxacml.sourceforge.net/> (2006-10-12).
- [12] Sun Microsystems, Inc. *A Brief Introduction to XACML*, 2003. <http://www.oasis-open.org/committees/download.php/2713/> (2006-10-05).
- [13] Sun Microsystems, Inc. *XACML: Access Control, Under Control*, 2005. http://research.sun.com/spotlight/2005_11_01-XACML.html (2006-09-14).
- [14] Torben Weis, Andreas Ulbrich, and Kurt Geihs. Model metamorphosis. *IEEE Software*, 20(5):46–51, 2003. September/October 2003.