# The Definition of
# The OPL Access Control Policy Language

Christopher Alm[1] and Ruben Wolf[2]

[1] Institute of IT-Security and Security Law, University of Passau
christopher.alm@uni-passau.de
[2] Fraunhofer Institute for Secure Information Technology (SIT), Darmstadt
ruben.wolf@sit.fraunhofer.de

UNIVERSITÄT
PASSAU
*Fakultät für Informatik und Mathematik*

Technical Report, Number MIP-0902
Department of Informatics and Mathematics
University of Passau, Germany
February 2009

# Contents

# The Definition of The OPL Access Control Policy Language[1]

OPL Version 1.2

Christopher Alm
christopher.alm@uni-passau.de
and
Ruben Wolf
rwolf@sit.fraunhofer.de

**Abstract**

Existing policy languages suffer from having a limited ability of directly and elegantly expressing high-level access control principles such as history-based separation of duty [24], binding of duty [17], context constraints [21], Chinese wall [7], and obligations [19]. Furthermore, it is often difficult to extend a language in order to retrofit these features once required or it is necessary to make use of complicated and complex language constructs to express a concept. In particular, the latter may cause human mistakes in the policy administration.

To address this problem, this report introduces a flexible, new policy language. The full language specification is given including a formal semantics written in Object Z and a formal syntax defined in XML. OPL can represent a wide range of access control principles directly by providing dedicated XML tags for each supported principle. It can be easily extended with further principles if necessary. Since OPL is based on a module concept, it can cope with the language complexity that usually comes with a growing expressiveness. Altogether OPL is suitable to be used in an enterprise environment: it combines the required expressiveness with the simplicity necessary for an appropriate administration. A considerable reference scenario is included in this report.

## 1 Introduction

Existing policy languages suffer from having a limited ability of directly and elegantly expressing high-level access control principles such as history-based separation of duty [24], binding of duty [17], context constraints [21], Chinese wall [7], and obligations [19]. Furthermore, it is often difficult to extend a language in order to retrofit these features once required or it is necessary to make use of complicated and complex language constructs to express a concept. In particular, the latter may cause human mistakes in the policy administration.

On the one hand, a reason for this problem can be that the underlying policy model is too focused on specific applications or scenarios as in the case of Ponder [9]. On the other hand, a reason can be that even though a language provides the necessary expressiveness, it requires the use of complicated and lengthy low-level language constructs in order to actually achieve the expressiveness as in the case of XACML [19]. Therefore, such a language makes policy administration error-prone and cumbersome.

As a solution to this, we present the flexible policy language *OPL*[2] that can represent a wide range of access control principles in XML directly by providing dedicated language constructs for each supported principle. It can be easily extended with further principles if necessary. OPL

[1] This work was supported by the German Ministry of Education and Research (BMBF) as part of the project ORKA, http://www.orka-projekt.de

[2] OPL stands for *ORKA Policy Language* because it has been defined as the policy language for the ORKA Project [1].

is under-pinned by a full formal semantic model specified in Object Z that is also given in this report. Since OPL is based on a module concept, it can cope with the language complexity that usually comes with a growing expressiveness. Altogether OPL is suitable to be used in an enterprise environment: it combines the required expressiveness with the simplicity necessary for an appropriate administration. Note that being able to express combinations of the mentioned access control principles in a policy language is necessary in real world scenarios such as in the financial sector or in the health care sector [1].

Our contributions are:

- We have developed the new policy language OPL which is based on the idea of modularizing the supported access control principles. Each concept is represented by a dedicated high-level language construct called *policy module* and can then be (re)used and combined with each other in order to create a policy. OPL is an XML application (cf. Sections 5 and 6).

  OPL is the policy language of the ORKA organizational control architecture developed in the ORKA research project [1]. The goal of ORKA is to develop a flexible and extensible authorization architecture that is able to enforce a wide range of organizational control principles and access control concepts in an enterprise environment. In ORKA, a rich tool set for OPL has been developed. This includes an administration tool, three different enforcement engines, and policy validation tools.

- We have developed a full formal semantic model of OPL (i.e. formal semantics or policy model), which is designed with a dedicated framework for the definition of access control models [3]. This framework, in turn, makes use of the systematic approach of formal object-oriented specification using Object Z [10] (cf. Sections 3 and 4).

  An important feature of the policy model is the fact that it clearly separates static policy data from other parts of the policy model such as dynamic session data, role activation data, and the policy decision logic.

  The policy model is a software model for an OPL policy engine. It is extensible so that new concepts can be defined and explored rapidly and concisely.

- We demonstrate how we can achieve the expressiveness required by an organization by using policy modules (cf. Sections 3 and 7).

  OPL directly supports advanced access control principles that cannot be expressed by means of dedicated high-level language constructs in any existing policy language to the best of our knowledge. Examples of such principles are history-based separation of duty and binding of duty on the task layer. We have developed a library of policy modules which is able to express the policies of our ORKA case studies, in particular the one from Schaad et al. [24] (cf. Sect. 7).

  This suite currently supports role hierarchy, separation of duty, context constraints, Chinese wall, object-based separation of duty, and workflow-related constraints including history-based separation of duty, prerequisites, binding of duty, and cardinality constraints.

- We explain why the modularization concept introduced by OPL is the right tool in order to cope with the complexity of an highly expressive policy language. The idea is to support for each application of OPL only what is really required by the organization and thus to keep the application of the policy language streamlined and focused (cf. Section 3).

- We explain how OPL can be extended in order to add new features to the language (cf. Section 3).

- We briefly demonstrate how to apply formal reasoning in order to provide an in-depth analysis of the policy modules Thereby we can, for example, prove important security properties as well as argue about design decisions (cf. Sections 4.1 and 4.6).

This report is organized as follows. Section 2 introduces some basic notions and motivates the need of a policy language. While Section 3 introduces the idea behind the way we define the semantics of OPL by means of a policy model, Section 4 introduces the library of currently available policy modules from the point of view of the semantics. Analogously, Section 5 and 6 introduce the idea behind how we define the syntax of OPL as well as the actual syntax of all available policy modules. In order to give an illustration of the power of OPL, we have a large example from the banking domain in Section 7.

# 2 Background and Motivation

## 2.1 Basic Definitions

### 2.1.1 Policy Model vs. Policy Object

An *(access control) policy* is the set of authorization requirements (i.e. authorizations) that a group of people agreed upon and thus it shall answer the question of who is allowed to perform which actions. An *(access control) policy object* is a document representing the policy. The policy object is based on a *policy model* or *access control model* defining a (formal or informal) semantic domain for the entities the policy object may consist of and thus giving a semantics to the policy object. In other words, while the policy model defines the structure and the properties of a policy and the rules according to which policies are created, a policy object is the representation of a concrete policy with real data for a concrete scenario. This is very similar to the distinction between classes and objects in object-oriented programming.

If the policy object is written in a language with a formal syntax (i.e. formal grammar), we will call this language *policy language* or *policy representation format*. Such a format makes the policy manageable and interpretable by the access decision and enforcement environment. In particular, it can be used for storing the policy persistently.

An *authorization architecture* is the design of a set of components and their communication relationships which are necessary to specify, manage, validate, and enforce an authorization policy. The *access control mechanism* or *policy engine* or *policy decision point* or *authorization engine* is an implementation of the component of an authorization architecture that decides access requests. Hence, the policy engine implements the policy model. The policy engine is configured by loading a machine readable version of the policy (i.e. the policy object written in a policy language) and it can then make decisions based on the policy represented by the policy object. The policy engine knows how to interpret the policy object because it is implemented according to the policy model giving the semantics to the policy object.

### 2.1.2 Role-Based Access Control

Since RBAC plays a central role to OPL, we briefly describe the key aspects of it. For further information confer to the literature [13, 27, 12].

In RBAC, users and permissions are not directly linked to each other, instead *roles* add a level of indirection between them in that users are assigned to roles and roles are assigned, in turn, to permissions. Roles are assumed to be related to jobs or functions of persons in organizations and to be associated to a semantics which expresses authority and responsibility.

RBAC is known for its various extensions, sometimes called *authorization constraints.* In OPL these authorization constraints are regarded as separated access control concepts and introduced through the mentioned policy modules. Authorization constraints can be regarded as restrictions on the sets, functions, and relations representing the RBAC model. There are many examples for the practical usage of authorization constraints, in particular for the modeling of higher-level organizational rules as needed, for example, in hospital or banking applications. In order to express such organizational rules, various types of authorization constraints have been identified in the literature such as several types of static and dynamic separation of duty constraints [20, 8, 15, 25, 2], constraints on delegation [26], cardinality constraints [23], context constraints [14, 18], and workflow constraints [4].

Finally, it should be noted that RBAC is assumed to be policy neutral in that it can simulate access control models based on completely different paradigms. Sandhu, for example, has shown that it is possible to simulate lattice-based access control by using RBAC plus constraints [22].

### 2.2 The Need for a Policy Language

A policy language is a common vehicle to make the policies that are run by an access control engine (which in turn is based on some access control model) manageable with regard to administration and enforcement. Therefore, it is necessary to develop a policy language for an access control model which is able to represent the policies of the model. There are, for example, approaches adding such a representation format for policies to an RBAC-based model [16, 6, 5].

The policy model which we use for OPL has already incorporated all necessary data modeling facilities for the definition of a policy language [3]. This is achieved in such a way that our access control model clearly separates the static data stored in a policy object from the dynamic data such as session data and role activations.

For OPL, also the flexibility and extensibility of the underlying policy model is inevitable because it aims to incorporate access control principles for various application scenarios including banking and health care. In addition to that, it strives for being open for extensions so that it can still be used when the requirements and the application environment change.

Note that our approach of developing a policy language is driven by an underlying policy model (i.e. access control model). There are also more *language-driven* approaches where the development of the access control model is driven by the syntax of a policy language [19, 9].

## 3 The OPL Policy Model

As we explained in Section 2.1.1, the policy model gives the semantics to the policy language. In this section we introduce the idea behind the policy model and explain how the formal semantics is established. Since the OPL Policy Model is an instance of the extensible framework for the specification of complex RBAC-models defined by Alm [3], we only introduce the key aspects necessary to understand the structure of OPL.

The idea behind the OPL policy model is that it is defined as a set of *policy modules*. Each policy module is focusing on a certain access control principle and defines all the functionality about the access control principle. This includes all aspects of policy management such as necessary data structures, administrative rules, and the access decision logic.

The policy model is specified by using the Object Z notation, a method which is both object-oriented and formal. In general, the object-orientation provides us with the means to cope with complexity, extensibility, flexibility, and feature reuse. Being a formal method, the Object Z specification leaves no room for ambiguities and it can be subject to formal reasoning.

Extensibility is achieved in such a way that new policy modules may be introduced or existing ones may be refined. The expressiveness of OPL is only limited by the expressiveness of Object Z which is based on first-order logic and set theory. Furthermore, the complexity of the policy model is also reduced, since modularization is one of the key countermeasures against complexity.

Every policy module is introduced by means of a set of Object Z classes (i.e. a classes ending with the words "Engine", "Policy", "Request", and "Env"). These classes are associated to

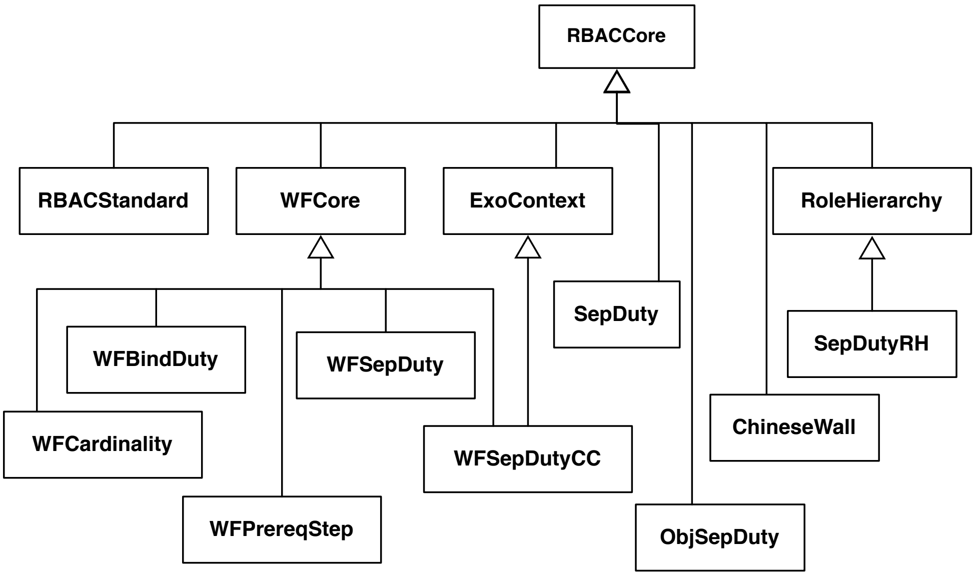


Figure 1: Dependency Hierarchy for Modules (Class Inheritance in UML)

each other as illustrated by Figure 2. The "Engine" class specifies an access control concept (such as separation of duty) including its access decision evaluation logic, its dynamic data structures such as session data, and its interface to the system raising access requests such as the *CheckAccess* function. Policy modules may depend on each other in such a way that a module can add or change some functionality to another one while leaving the rest as it is. The dependences between the modules is realized through class inheritance. Figure 1 shows the current class hierarchy in UML.

The purpose of the "Policy" classes classes is to provide the data structures for the policy language (including administrative rules, administrative functions and consistency rules). Hence these *policy modules* define which information is supposed to be stored within a policy. The policy modules are arranged in the same class inheritance relation as the policy modules so that we actually have the inheritance tree of Figure 1 twice.

The "Request" classes specify the format for of the requests for each module. The "Env" classes specify the information provided through the environment to the policy engine such as context information and data from the workflow management system.

Finally, a policy object can be created by simply selecting the required modules and by filling the data structures with the required data. During the selection of policy modules the dependencies imposed by Figure 1 has to be taken into account. With regard to the theory (i.e. the policy model) behind it, this means that the features of all selected Object Z classes are inherited by using *multiple inheritance*. Multiple inheritance is a rather unique Object Z feature that allows us to merge the state spaces and operations of different modules and therefore to include the functionality they provide. The complexity of policy creation and administration is reduced since an administrator needs only to focus on the policy modules that he or she has selected for the current scenario. Thereby it is possible to ignore whole parts of the policy language and to take into account only the small subset of the language that is relevant for the current scenario.

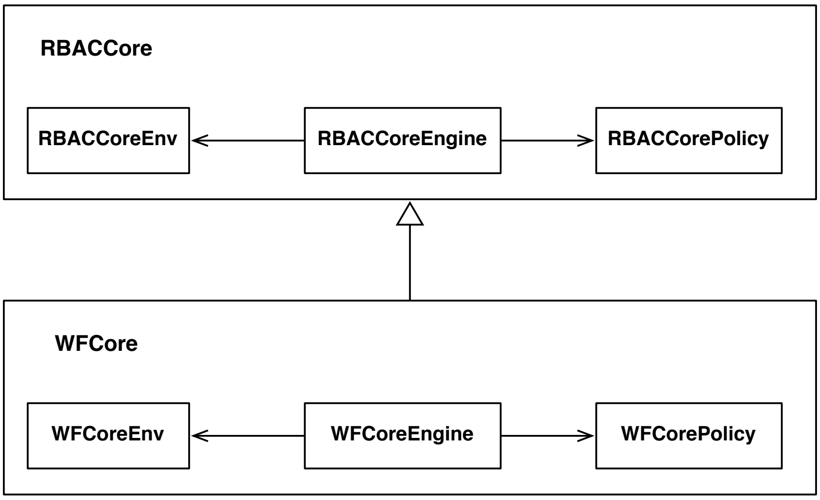


Figure 2: Relationship between the classes of a module (in UML)

# 4 Library of Modules: Formal Semantics

Each module is introduced in a separate subsection. The structure of such a subsection is always the same. It consists of the following four mandatory parts and one optional part at the end:

1. **Formal Definition:** in this part the module is defined formally by using Object Z. A concatenation of all formal definition parts of all modules leads to the full self-contained formal definition of the whole policy model.

2. **Interpretation of basic types:** in this part all introduced Object Z basic types are given an informal semantics. Here the reader is told how the model can be interpreted. The model gets connected to the real world at this point.

3. **Notes on operation invocation:** each module states a clear interface it provides to the environment (i.e. to the applications invoking the operations of the policy engine). The definition of the interface is done by means of the Object Z visibility list at the beginning of each module. The "notes on operation invocation" part contains an informal description on when or under what circumstances the operations of the interface are invoked. The question for what purpose an operation is invoked is answered by its formal definition. This part provides additional information on how the model can be interpreted.

4. **Informal Summary:** this part summarizes the functionality of a module and helps to understand the formal definition. The main purpose is to give a quick overview of the module. We recommend to read this paragraph first. We still keep it almost at the end because it is there only for convenience and does not have any authoritative function.

5. **Remarks:** the purpose of this optional part is to state remarks such as design decisions and interesting properties of a module.

The following aspects of an access control model are covered by each policy module:

- **Policy Data Structures:** the state space of a *Policy class defines which information of a policy module is supposed to be stored in a policy object. Such a policy object can, for example, be made persistent and load/unload by the policy engine. According to Strembeck's categorization of access control data [21], all data structures of a module that are not part of the policy data structures are part of the so-called *endogenous context.*

- **Administrative rules and functions:** the class invariants of a *Policy class define the *administrative constraints* that need to be fulfilled at anytime in order to keep the policy consistent. The operations defined in a *Policy class are the administrative functions that are supposed to be used for policy administration and management at runtime. This may include several functions for policy review such as a query function for a data structure.
- **Evaluation logics and policy engine operations:** the state space of *Engine class defines the endogenous context of the policy engine including, for example, session data or role activations. The class invariants define certain consistency properties that need to be fulfilled at anytime in order to keep the policy engine in a valid state. The operations defined in a *Engine class are the ones invoked while processing user requests such as access requests, subject creation requests, and role activation requests. Mainly they operate on the endogenous context, however, they may also affect policy data.

## 4.1 Role-Based Access Control

### Formal Definition

$$[USER, ROLE, SUBJECT, OBJECT, OBJECTINSTANCE, OPERATION]$$

$$PERMISSION == OPERATION \times OBJECT$$

$$DECISION ::= grant \mid deny$$

*RBACCoreRequest*

$$\begin{array}{l} s : SUBJECT \\ op : OPERATION \\ obi : OBJECTINSTANCE \end{array}$$

*RBACCoreEnv*

$$\begin{array}{l} OI : \mathbb{P}\, OBJECTINSTANCE \\ O : \mathbb{P}\, OBJECT \\ instanceof_{obj} : OI \rightarrow O \end{array}$$

*RBACCorePolicy*

$$\begin{array}{l} U : \mathbb{P}\, USER \\ R : \mathbb{P}\, ROLE \\ P : \mathbb{P}\, PERMISSION \\ UA : U \leftrightarrow R \\ PA : R \leftrightarrow P \end{array}$$

*AddUser*

$$\begin{array}{l} \Delta(U) \\ u? : USER \\ \hline u? \notin U \\ U' = U \cup \{u?\} \end{array}$$

*DeleteUser*

$$\begin{array}{l} \Delta(U, UA) \\ u? : USER \\ \hline u? \in U \\ U' = U \setminus \{u?\} \\ UA' = UA \setminus \{r : ROLE \bullet u? \mapsto r\} \end{array}$$

*AddRole*

$$\begin{array}{l} \Delta(R) \\ r? : ROLE \\ \hline r? \notin R \\ R' = R \cup \{r?\} \end{array}$$

*DeleteRole*

$$\begin{array}{l} \Delta(R, UA, PA) \\ r? : ROLE \\ \hline r? \in R \\ UA'^{\sim}(\!|\ \{r?\}\ |\!) = \varnothing \\ PA'(\!|\ \{r?\}\ |\!) = \varnothing \\ R' = R \setminus \{r?\} \end{array}$$

*AddPermission*

$$\begin{array}{l} \Delta(P) \\ p? : PERMISSION \\ \hline p? \notin P \\ P' = P \cup \{p?\} \end{array}$$

*DeletePermission*

$$\begin{array}{l} \Delta(P, PA) \\ p? : PERMISSION \\ \hline p? \in P \\ PA'^{\sim}(\!|\ \{p?\}\ |\!) = \varnothing \\ P' = P \setminus \{p?\} \end{array}$$

*AddUA*

$$\begin{array}{l} \Delta(UA) \\ u? : USER \\ r? : ROLE \\ \hline u? \in U;\ r? \in R \\ u? \mapsto r? \notin UA \\ UA' = UA \cup \{u? \mapsto r?\} \end{array}$$

*DeleteUA*

$$\begin{array}{l} \Delta(UA) \\ u? : USER \\ r? : ROLE \\ \hline u? \mapsto r? \in UA \\ UA' = UA \setminus \{u? \mapsto r?\} \end{array}$$

*AddPA*

$$\begin{array}{l} \Delta(PA) \\ r? : ROLE \\ p? : PERMISSION \\ \hline r? \in R;\ p? \in P \\ r? \mapsto p? \notin PA \\ PA' = PA \cup \{r? \mapsto p?\} \end{array}$$

*DeletePA*

$$\begin{array}{l} \Delta(PA) \\ r? : ROLE \\ p? : PERMISSION \\ \hline r? \mapsto p? \in PA \\ PA' = PA \setminus \{r? \mapsto p?\} \end{array}$$

$RBACCoreEngine$

For all modules boolean variables are defined here.

$RBACStandardInv : \mathbb{B}$

$RoleHierarchyInv : \mathbb{B}$

Policy:

$Policy : \downarrow RBACCorePolicy_{\copyright}$

Environment:

$Env : \downarrow RBACCoreEnv_{\copyright}$

Endogeneous Context:

$S : \mathbb{P}\, SUBJECT$

$SR : S \leftrightarrow Policy.R$

$SU : S \rightarrow Policy.U$

$ImportPolicy$

$\Delta(Policy)$

$Policy? : \downarrow RBACCorePolicy$

$Policy? \in \downarrow RBACCorePolicy$

$Policy' = Policy?$

$ExportPolicy$

$Policy! : \downarrow RBACStandardPolicy$

$Policy! \in \downarrow RBACStandardPolicy$

$Policy! = Policy$

$Grant$

$d! : DECISION$

$d! = grant$

$Deny$

$d! : DECISION$

$d! = deny$

$Authorization$

$req? : \downarrow RBACCoreRequest$

$RBACStandardPolicy$

$RBACCorePolicy$

*RBACStandardEngine*

$$\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject},$$
$$\mathit{ActivateRole}, \mathit{DeactivateRole})$$

$$\mathit{RBACCoreEngine}$$

$$\begin{array}{l}
\mathit{RBACStandardInv} \\
\neg\ \mathit{RoleHierarchyInv} \\
\mathit{Policy} \in \downarrow \mathit{RBACStandardPolicy} \\
\forall\, s : S \bullet \mathit{SR}(\!|\ \{s\}\ |\!) \subseteq \mathit{Policy}.\mathit{UA}(\!|\ \{\mathit{SU}(s)\}\ |\!)
\end{array}$$

*ImportPolicy*

$$\mathit{Policy}? \in \downarrow \mathit{RBACStandardPolicy}$$

*ExportPolicy*

$$\mathit{Policy}! \in \downarrow \mathit{RBACStandardPolicy}$$

*Authorization*

$$\begin{array}{l}
\exists\, r : \mathit{ROLE};\ p : \mathit{PERMISSION} \bullet \\
\quad (\mathit{req}?.s, r) \in \mathit{SR} \quad \wedge \\
\quad (r, p) \in \mathit{Policy}.\mathit{PA} \quad \wedge \\
\quad (\mathit{req}?.\mathit{op}, \mathit{Env}.\mathit{instanceof}_{\mathit{obj}}(\mathit{req}?.\mathit{obi})) = p
\end{array}$$

$$\mathit{CheckAccess} \mathrel{\widehat=} (\mathit{Grant} \wedge \mathit{Authorization}) \vee (\mathit{Deny} \wedge \neg\ \mathit{Authorization})$$

*CreateSubject*

$$\begin{array}{l}
\Delta(S, \mathit{SR}, \mathit{SU}) \\
s? : \mathit{SUBJECT} \\
u? : \mathit{USER} \\
\mathit{rs}? : \mathbb{P}\, \mathit{ROLE} \\
\hline
u? \in \mathit{Policy}.U; \\
\mathit{rs}? \subseteq \mathit{Policy}.\mathit{UA}(\!|\ \{u?\}\ |\!); \\
s? \notin S \\
S' = S \cup \{s?\} \\
\mathit{SR}' = \mathit{SR} \cup \{r : \mathit{rs}? \bullet s? \mapsto r\} \\
\mathit{SU}' = \mathit{SU} \cup \{s? \mapsto u?\}
\end{array}$$

*DestroySubject*

$$\begin{array}{l}
\Delta(S, \mathit{SR}, \mathit{SU}) \\
s? : \mathit{SUBJECT} \\
u? : \mathit{USER} \\
\hline
s? \in S;\ u? = \mathit{SU}(s?) \\
S' = S \setminus \{s?\} \\
\mathit{SR}' = \mathit{SR} \setminus \{r : \mathit{ROLE} \bullet s? \mapsto r\} \\
\mathit{SU}' = \mathit{SU} \setminus \{s? \mapsto u?\}
\end{array}$$

*ActivateRole*

$$\begin{array}{l}
\Delta(\mathit{SR}) \\
r? : \mathit{ROLE} \\
s? : \mathit{SUBJECT} \\
u? : \mathit{USER} \\
\hline
u? \in \mathit{Policy}.U;\ s? \in S;\ r? \in \mathit{Policy}.R \\
u? = \mathit{SU}(s?) \\
r? \in \mathit{Policy}.\mathit{UA}(\!|\ \{u?\}\ |\!) \\
r? \notin \mathit{SR}(\!|\ \{s?\}\ |\!) \\
\mathit{SR}' = \mathit{SR} \cup \{s? \mapsto r?\}
\end{array}$$

*DeactivateRole*

$$\begin{array}{l}
\Delta(\mathit{SR}) \\
r? : \mathit{ROLE} \\
s? : \mathit{SUBJECT} \\
u? : \mathit{USER} \\
\hline
u? \in \mathit{Policy}.U;\ s? \in S;\ r? \in \mathit{Policy}.R \\
u? = \mathit{SU}(s?) \\
r? \in \mathit{SR}(\!|\ \{s?\}\ |\!) \\
\mathit{SR}' = \mathit{SR} \setminus \{s? \mapsto r?\}
\end{array}$$

### Interpretation of basic types

- **Subjects:** we interpret the elements of *SUBJECT* as identifiers to the active entities of the system such as processes in an operating system.
- **Objects/Object-Instances:** we interpret the elements of *OBJECTINSTANCE* as identifiers to the entities of the system where subjects perform actions on. They are assigned to a type in *OBJECT*. Note that in the policy permissions are based on objects rather than on object-instances. This is because object-instances are considered to be more transient entities and they are created and deleted outside of the scope of the policy engine. For example the concrete elements of a file system can be regarded as object instances. Note that processes (i.e. subjects) might also be made available through the file system to provide, for example, inter process communication channels. Another example are business objects such as credit-objects.
- **Operations:** we interpret the elements of *OPERATION* as identifiers to the actions that subjects (request to) perform on objects. In some cases it might be necessary to pass additional parameters to an operation when it is invoked. Such parameters, however, are not part of the formal model and it is up the interpretation of this basic type how to handle them.
- **Users:** we interpret the elements of *USER* as identifiers to the digital representation of human users of the system, as they are provided by, for example, a directory service.
- **Roles:** we interpret the elements of *ROLE* as identifiers to the digital representation of the roles representing the structure of an organization. In contrast to the previously defined basic types, roles are a concept that is internal to authorization. Hence, the only occurrence of the just mentioned "digital representation" can be the name of the role as it is defined in a policy. In distributed or more complex systems, however, roles may be managed separately. In this case they can be understood in the same way as the other basic types in this paragraph.

### Notes on Operation Invocation

- *ImportPolicy* and *ExportPolicy* appear in all modules. They are called by a PAP in order to load or unload a policy, respectively.
- Each time a subject requests to perform an operation on an object the operation *CheckAccess* is called and answers whether or not to grant this request.
- The further engine operations *CreateSubject*, *DestroySubject*, *ActivateRole*, *DeactivateRole* are invoked by the PEP whenever a user creates/destroys a subject or (de)activates a role for a subject.

**Informal Summary** The basic functionality of role-based access control has already been defined through two separate modules. While *RBACCore* introduces only data structures and heads of operations that are shared with all existing modules, *RBACStandard* defines the necessary constraints for RBAC. In particular, *RBACCore* is the root element of our module hierarchy.

The state space of *RBACCore* can be visualized by the following figure. This figure further supports the informal descriptions of this paragraph.

$$\begin{tikzcd}
U \arrow[rr, leftrightarrow, "UA" description] & & R \arrow[rr, leftrightarrow, "PA" description] & & P \\
& S \arrow[ul, "SU" description] \arrow[ur, leftrightarrow, "SR" description] & & &
\end{tikzcd}$$

The *RBACStandard* module is mostly derived from the core model of the ANSI RBAC standard [27]. The class invariant

$$\forall\, s : S \bullet SR(\!|\ \{s\}\ |\!) \subseteq UA(\!|\ \{SU(s)\}\ |\!)$$

says that the set of activated roles of a subject needs to be a subset of the set of assigned roles of the user corresponding to the subject. Hence, a user can only activate a role if he or she is assigned to it.

**Remarks**

- The corresponding OPL/XML data structures of this policy module are defined in Section 6.1.
- Note that it would have been possible to write *CheckAccess* by means one single schema. The reason why we decided not to do so is that this way we are prepared for more elaborate modules connecting, for example, obligations or logging events with access decisions. These can be added by simply extending the schema definitions of *Grant* and *Deny*.
- One of the design decisions made for *RBACCoreEngine* is to decide whether to chose

  $$PERMISSION == \mathbb{P}(OPERATION \times OBJECT)$$

  or

  $$PERMISSION == OPERATION \times OBJECT$$

  as a definition of a permission. While in the first case a permission is a set of tuples, in the second case a permission is one tuple. We decided to chose the second variant (in contrast to the ANSI standard) because the first variant adds an unnecessary complexity to the model. While in the first place it seemed appropriate to use the first version, particularly because a workflow task could have been directly mapped to one permission instead of a group of permissions, it turned out on the one hand that it is not necessary to have this permission aggregation for this purpose. On the other hand, having a permission as a set of tuples (i.e. first variant) makes it difficult when it comes to selecting the "requested permission" (e.g. in order to check for associated context constraints) out of a object/operation pair given by a request. This is because such a pair can be an element of more than one permission in this case.
- In the version of role-based access control published in 1992 [11] there was a rule called *role assignment rule* stated as follows "a subject can execute a transaction only if the subject has selected or been assigned to a role". This rule is redundant as we show here.

**Proposition 1** For any $s : S$ and $(op, ob) : Policy.P$ within the scope of $RBACStandardEngine$ such that $Authorization[s/s?, op/op?, ob/ob?]$ it follows that $SR(\!|\ \{s\}\ |\!) \neq \varnothing$.

Proof: Assume that there exist $s : S$ and $(op, ob) : Policy.P$ such that

$$Authorization[s/s?, op/op?, ob/ob?]$$

but $SR(\!|\ \{s\}\ |\!) = \varnothing$. From $Authorization[s/s?, op/op?, ob/ob?]$ it follows that there exists $r : R$ such that $(s, r) \in SR$ which yields a contradiction. □

## 4.2 Role Hierarchy

### Formal Definition

$$PartialOrder[T] == \{R : T \leftrightarrow T \mid R^* = R;\ R^{-1} \cap R = \mathrm{id}\ T\}$$

*RoleHierarchyPolicy*

$RBACCorePolicy$

$$\begin{array}{l}
RH : R \leftrightarrow R \\
\hline
RH^* \in \{PO : PartialOrder[ROLE]\}
\end{array}$$

*AddInheritance*

$$\begin{array}{l}
\Delta(RH) \\
senior? : R \\
junior? : R \\
\hline
\neg(junior? \succeq_{RH} senior?) \\
\neg(senior? \succeq_{RH} junior?) \\
RH' = RH \cup \{senior? \mapsto junior?\}
\end{array}$$

*DeleteInheritance*

$$\begin{array}{l}
\Delta(RH) \\
senior? : R \\
junior? : R \\
\hline
RH' = RH \setminus \{senior? \mapsto junior?\}
\end{array}$$

*RoleHierarchyEngine*

$$\begin{array}{l}
\upharpoonright(ImportPolicy, ExportPolicy, CheckAccess, CreateSubject, DestroySubject,\\
\quad ActivateRole, DeactivateRole)
\end{array}$$

$RBACCoreEngine$

$$\begin{array}{l}
\_ \succeq_{RH} \_ : PartialOrder[ROLE]\\
\hline
RoleHierarchyInv\\
\neg\, RBACStandardInv\\
Policy \in \downarrow RoleHierarchyPolicy\\
\_ \succeq_{RH} \_ = Policy.RH^{*}\\
\forall\, s : S \bullet (SR\llparenthesis \{s\} \rrparenthesis) \subseteq (\_ \succeq_{RH} \_\llparenthesis\, Policy.UA\llparenthesis \{SU(s)\} \rrparenthesis \rrparenthesis)
\end{array}$$

*ImportPolicy*

$$Policy? \in \downarrow RoleHierarchyPolicy$$

*ExportPolicy*

$$Policy! \in \downarrow RoleHierarchyPolicy$$

*Authorization*

$$\begin{array}{l}
\exists\, r, r_j : ROLE \mid r \succeq_{RH} r_j;\ p : PERMISSION;\ \bullet\\
\quad (req?.s, r) \in SR \quad \wedge\\
\quad (r_j, p) \in Policy.PA \quad \wedge\\
\quad (req?.op, Env.instanceof_{obj}(req?.obi)) = p
\end{array}$$

*CreateSubject*

$$\begin{array}{l}
\Delta(S, SR, SU)\\
s? : SUBJECT\\
u? : USER\\
rs? : \mathbb{P}\, ROLE\\
\hline
u? \in U;\ s? \notin S\\
rs? \subseteq (\_ \succeq_{RH} \_\llparenthesis\, Policy.UA\llparenthesis \{u?\} \rrparenthesis \rrparenthesis)\\
S' = S \cup \{s?\}\\
SR' = SR \cup \{r : rs? \bullet s? \mapsto r\}\\
SU' = SU \cup \{s? \mapsto u?\}
\end{array}$$

*ActivateRole*

$$\begin{array}{l}
\Delta(SR)\\
r? : ROLE\\
s? : SUBJECT\\
u? : USER\\
\hline
u? \in U;\ s? \in S;\ r? \in Policy.R\\
u? = SU(s?)\\
r? \in (\_ \succeq_{RH} \_\llparenthesis\, Policy.UA\llparenthesis \{u?\} \rrparenthesis \rrparenthesis)\\
r? \notin SR\llparenthesis \{s?\} \rrparenthesis\\
SR' = SR \cup \{s? \mapsto r?\}
\end{array}$$

*DestroySubject*

$$\begin{array}{l}
\Delta(S, SR, SU)\\
s? : SUBJECT\\
u? : USER\\
\hline
s? \in S;\ u? = SU(s?)\\
S' = S \setminus \{s?\}\\
SR' = SR \setminus \{r : ROLE \bullet s? \mapsto r\}\\
SU' = SU \setminus \{s? \mapsto u?\}
\end{array}$$

*DeactivateRole*

$$\begin{array}{l}
\Delta(SR)\\
r? : ROLE\\
s? : SUBJECT\\
u? : USER\\
\hline
u? \in Policy.U;\ s? \in S;\ r? \in Policy.R\\
u? = SU(s?)\\
r? \in SR\llparenthesis \{s?\} \rrparenthesis\\
SR' = SR \setminus \{s? \mapsto r?\}
\end{array}$$

**Interpretation of basic types** This module does not introduce any new basic types.

**Notes on Operation Invocation** The same notes apply as in the case of *RBACCore* and *RBACStandard*.

**Informal Summary** The *RoleHierarchy* module adds (as the name implies) a hierarchy to the set of roles. The hierarchy needs to be a partial order. It has mainly two consequences:

1. The class invariant of *RoleHierarchyEngine* is different to the invariant of *RBACStandardEngine* in such a way that in addition to roles directly assigned to a user, a user is now able to activate roles junior to his/her assigned roles (i.e. downward the hierarchy). The class invariant

   $$\forall\, s : S \bullet (SR \llparenthesis\, \{s\} \,\rrparenthesis) \subseteq (\_ \succeq_{RH} \_ \llparenthesis\, UA \llparenthesis\, \{SU(s)\} \,\rrparenthesis \,\rrparenthesis)$$

   reads as follows. Any subject is only allowed to activate a role that its corresponding user is either assigned to directly or that is a role that is junior to a role the user is assigned to.

2. The *Authorization* operation is different to *RBACStandard* in such a way that now access is also granted to any permission that is assigned to a junior role of an activated role of a user.

The last question to answer in this paragraph is why we need two versions of the role hierarchy—namely $RH$ and $\succeq_{RH}$. While $RH$ is the role hierarchy as it is represented in the policy, $\succeq_{RH}$ is only existent at runtime. $RH$ is just an antisymmetric relation because it would not make sense to ensure transitivity and reflexivity in the policy. Otherwise, for example, each time a role inheritance is added to the policy, it would be necessary to add all indirect junior-senior-relationships in order to keep the relation transitive. In contrast, $\succeq_{RH}$ is the transitive reflexive closure of $RH$, and it is used when evaluating access decisions and role activations based on the role hierarchy as a partial order.

### Remarks

- The corresponding OPL/XML data structures of this policy module are defined in Section 6.2.

- It should be noted that it would have been possible to introduce role hierarchies as a child module of *RBACStandard*. In this case the Object Z definition principle “cancellation and redefinition” needs to be used [10]. This principle makes it possible for a subclass to cancel some features of its superclass in order to make new definitions with the same name but a (slightly) different semantics. Cancellation and redefinition is carried out by simply renaming the features which are supposed to be canceled upon inheritance. E.g.

  $$RBACCoreEngine[oldS/S, \cdots, oldCreateSubject/CreateSubject]$$

  This way also the class invariant of *RBACCoreEngine* gets canceled and a new one can be introduced for *RoleHierarchyEngine*. For *RoleHierarchyEngine* the whole subject concept of *RBACCoreEngine* needs to be replaced because the semantics of role activation needs to be changed (i.e. the class invariant).

## 4.3 Context Constraints

### Formal Definition

$[ContextFunctionName, CFParamKey, CFParamValue]$

$CFParamContext : \mathbb{B}$

$CFParamType ::= String \mid Date \mid Time \mid Int$

$ContextFunctionParam == CFParamKey \times CFParamValue \times CFParamType \times CFParamContext$

$ContextConstraint == ContextFunctionName \times \mathbb{P}\ ContextFunctionParam$

*ExoContextEnv*

$RBACCoreEnv$

$$\begin{array}{l}
CFNAME : \mathbb{P}\ ContextFunctionName \\
MandatoryKeys : CFNAME \leftrightarrow CFParamKey \\
OptionalKeys : CFNAME \leftrightarrow CFParamKey \\
\hline
\forall\ cfname : CFNAME \bullet MandatoryKeys \llparenthesis \{cfname\} \rrparenthesis \cap OptionalKeys \llparenthesis \{cfname\} \rrparenthesis = \varnothing
\end{array}$$

*ΦEVAL*

$$\begin{array}{l}
cfname? : ContextFunctionName \\
cfps? : \mathbb{P}\ ContextFunctionParam
\end{array}$$

The requested context function name must be known to the environment for context evaluation.

$cfname? \in CFNAME$

Are the given parameter keys correct?

$$\begin{array}{l}
\mathrm{dom}\ cfps? \subseteq MandatoryKeys \llparenthesis \{cfname?\} \rrparenthesis \cup OptionalKeys \llparenthesis \{cfname?\} \rrparenthesis \\
MandatoryKeys \llparenthesis \{cfname?\} \rrparenthesis \subseteq \mathrm{dom}\ cfps?
\end{array}$$

It is only specified that context functions have a boolean return value. It is not specified in which way context functions are evaluated, because this depends on the set of available context functions.

*ExoContextPolicy*

$$\begin{array}{l} RBACCorePolicy \\ \hline CC : \mathbb{P}\ ContextConstraint \\ PCC : P \leftrightarrow CC \\ PACC : PA \leftrightarrow CC \\ RCC : R \leftrightarrow CC \end{array}$$

*AddCC*

$$\begin{array}{l} \Delta(CC) \\ cc? : ContextConstraint \\ \hline cc? \notin CC \\ CC' = CC \cup \{cc?\} \end{array}$$

*DeleteCC*

$$\begin{array}{l} \Delta(CC) \\ cc? : ContextConstraint \\ \hline cc? \in CC \\ PCC^{\sim}(\!|\ \{cc?\}\ |\!) = \varnothing \\ RCC^{\sim}(\!|\ \{cc?\}\ |\!) = \varnothing \\ PACC^{\sim}(\!|\ \{cc?\}\ |\!) = \varnothing \\ CC' = CC \setminus \{cc?\} \end{array}$$

*AddPCC*

$$\begin{array}{l} \Delta(PCC) \\ p? : PERMISSION \\ cc? : ContextConstraint \\ \hline (p?, cc?) \notin PCC \\ p? \in P;\ \ cc? \in CC \\ PCC' = PCC \cup \{p? \mapsto cc?\} \end{array}$$

*DeletePCC*

$$\begin{array}{l} \Delta(PCC) \\ p? : PERMISSION \\ cc? : ContextConstraint \\ \hline (p?, cc?) \in PCC \\ PCC' = PCC \setminus \{p? \mapsto cc?\} \end{array}$$

*AddPACC*

$$\begin{array}{l} \Delta(PACC) \\ p? : PERMISSION \\ r? : ROLE \\ cc? : ContextConstraint \\ \hline (r?, p?) \in PA;\ \ cc? \in CC \\ ((r?, p?), cc?) \notin PACC \\ PACC' = PACC \cup \{(r?, p?) \mapsto cc?\} \end{array}$$

*DeletePACC*

$$\begin{array}{l} \Delta(PACC) \\ p? : PERMISSION \\ r? : ROLE \\ cc? : ContextConstraint \\ \hline ((r?, p?), cc?) \in PACC \\ PACC' = PACC \setminus \{(r?, p?) \mapsto cc?\} \end{array}$$

*AddRCC*

$$\begin{array}{l} \Delta(RCC) \\ r? : ROLE \\ cc? : ContextConstraint \\ \hline (r?, cc?) \notin RCC \\ r? \in P;\ \ cc? \in CC \\ RCC' = RCC \cup \{r? \mapsto cc?\} \end{array}$$

*DeleteRCC*

$$\begin{array}{l} \Delta(RCC) \\ r? : ROLE \\ cc? : ContextConstraint \\ \hline (r?, cc?) \in RCC \\ RCC' = RCC \setminus \{r? \mapsto cc?\} \end{array}$$

$DeletePermission \mathrel{\widehat{=}}$
$\quad \forall r? : ROLE;\ cc? : ContextConstraint \mid ((r?, p?), cc?) \in PACC \bullet DeletePACC \quad \land$
$\quad \forall cc? : ContextConstraint \mid (p?, cc?) \in PCC \bullet DeletePCC$
$DeletePA \mathrel{\widehat{=}}$
$\quad \forall cc? : ContextConstraint \mid ((r?, p?), cc?) \in PACC \bullet DeletePACC$
$DeleteRole \mathrel{\widehat{=}}$
$\quad \forall p? : ROLE;\ cc? : ContextConstraint \mid ((r?, p?), cc?) \in PACC \bullet DeletePACC \quad \land$
$\quad \forall cc? : ContextConstraint \mid (r?, cc?) \in RCC \bullet DeleteRCC$

*ExoContextEngine*

$\upharpoonright(ImportPolicy, ExportPolicy, CheckAccess, CreateSubject, DestroySubject,$
$\quad ActivateRole, DeactivateRole)$

$RBACCoreEngine$

$Policy \in \downarrow ExoContextPolicy$

*ImportPolicy*

$Policy? \in \downarrow ExoContextPolicy$

*ExportPolicy*

$Policy! \in \downarrow ExoContextPolicy$

*EvalContextConstraints*

$ccs? : \mathbb{P}\ ContextConstraint$

$\bigwedge (cfname, cfps) : ccs? \bullet Env.\Phi EVAL[cfname/cfname?, cfps/cfps?]$

*ReturnAuthorizationCC*

$req? : \downarrow Request$
$ccs! : \mathbb{P}\ ContextConstraint$

$ccs! =$
$\quad Policy.PCC \llparenthesis \{p : Policy.P \mid p = (req?.op, Env.instanceof_{obj}(req?.obi))\} \rrparenthesis \quad \cup$
$\quad Policy.PACC \llparenthesis \{(r, p) : Policy.PA \mid p = (req?.op, Env.instanceof_{obj}(req?.obi));$
$\qquad r \in SR\llparenthesis \{req?.s\} \rrparenthesis\} \rrparenthesis \quad \cup$
$\quad Policy.RCC \llparenthesis \{(r, p) : Policy.PA \mid p = (req?.op, Env.instanceof_{obj}(req?.obi));$
$\qquad r \in SR\llparenthesis \{req?.s\} \rrparenthesis \bullet r\} \rrparenthesis$

$Authorization \mathrel{\widehat{=}} ReturnAuthorizationCC \fatsemi EvalContextConstraints$

*ReturnActivateRoleCC*

$s? : SUBJECT$
$u? : USER$
$r? : ROLE$
$ccs! : \mathbb{P}\ ContextConstraint$

$ccs! = Policy.RCC \llparenthesis \{r?\} \rrparenthesis$

$ActivateRole \mathrel{\widehat{=}} ReturnActivateRoleCC \fatsemi EvalContextConstraints$

$$
\begin{array}{|l}
\textit{ReturnCreateSubjectCC} \\
\hline
s? : \mathit{SUBJECT} \\
u? : \mathit{USER} \\
rs? : \mathbb{P}\,\mathit{ROLE} \\
ccs! : \mathbb{P}\,\mathit{ContextConstraint} \\
\hline
ccs! = \mathit{Policy}.\mathit{RCC}(\!|\ rs?\ |\!) \\
\hline
\end{array}
$$

$$\mathit{CreateSubject} \mathrel{\widehat=} \mathit{ReturnCreateSubjectCC} \mathbin{⨟} \mathit{EvalContextConstraints}$$

### Interpretation of basic types

- **Context Function Name:** A context function name is an identifier for a boolean operator which is used by the context evaluator in order to deal with context information represented through context function parameters. An example for a context function are comparison functions such as "time-in-between" or "greater-or-equal".

- **Context Function Parameters:** A context function parameter represents a context information. There are two cases depending on the value of *CFParamContext*.

  - If *CFParamContext* is true, then the value *CFParamValue* needs to be resolved by an appropriate context provider before it can be processed by the context function. E.g. *CFParamValue* could be something like "current-time" or "current-subject" or "current-location".
  - If *CFParamContext* is false, then the value *CFParamValue* is like a constant that can directly be processed by the context function.

  The content of *CFParamType* states the data type of the value of *CFParamValue*. In case that the value of *CFParamValue* needs to be resolved first, it refers to the type after the resolution.

  Finally *CFParamKey* is a helper parameter that makes it possible to use different types of parameter handling (cf. Section 6.5).

**Notes on Operation Invocation** This module does not introduce any operations other than administrative operations.

**Informal Summary** The *ExoContext* module is based on the idea that context constraints can be assigned to permissions, roles, and permission assignments.

- In case of the assignment to permissions, the permissions can only be granted if the context constraint is satisfied. Therefore, context constraints can be assigned to permissions via the *PCC* relation. In particular, whenever an access decision is made, the renewed *Authorization* operation firstly checks which of the constraints apply for the request (*ReturnAuthorizationCC*). Secondly, it evaluates all the context constraints (*EvalContextConstraints*).

- In case of the assignment to a role, the permissions that a user acquires via this role are only available if all assigned context constraints are satisfied. Furthermore, it is not possible to activate a role if its assigned context constraints are not satisfied. Therefore, context constraints can be assigned to permissions via the *RCC* relation.

- In case of the assignment of a context constraint to a permission assignment, an authorization that requires a certain permission assignment cannot be granted if the assigned context constraint is not satisfied. Therefore, context constraints can be assigned to permissions via the *PACC* relation.

**Remarks**

- The corresponding OPL/XML data structures of this policy module are defined in Section 6.5.
- It should be noted that our approach is a superset of Strembeck's and Neumann's approach [21] who only consider the case of assigning context constraints to permissions (*PCC*). This is a drawback because, firstly, *PACC* is useful if a certain permission is supposed to be restricted only for some roles but not for all roles. Secondly, having the ability to constrain whole roles (via *RCC*) is a convenient policy modeling feature that cannot be expressed in their approach.
- It should be noted that with our approach the GT-RBAC model [18] can be imitated. For this purpose, we make use of the features realized through the *RCC* relation.

## 4.4 Basic Separation of Duty

### Formal Definition

*SepDutyPolicy*

*RBACCorePolicy*

Static separation of duty:

$SSoD : (\mathbb{P}\, ROLE) \pfun \mathbb{N}$

Static separation of duty for permissions:

$SSoDP : (\mathbb{P}\, PERMISSION) \pfun \mathbb{N}$

Strict static separation of duty:

$SSSoD : (\mathbb{P}\, ROLE) \pfun \mathbb{N}$

Dynamic separation of duty:

$DSoD : (\mathbb{P}\, ROLE) \pfun \mathbb{N}$

$\forall\, u : U;\ (rs, n) : SSoD \bullet \#(rs \cap UA \llparenthesis \{u\} \rrparenthesis) \leq n$
$\forall\, r : R;\ (ps, n) : SSoDP \bullet \#(ps \cap PA \llparenthesis \{r\} \rrparenthesis) \leq n$

Additionally, minimal forbidden subsets (mfrs) of a critical role set do not have any common permission. I.e. they are (not necessarily pairwise) disjoint.

$$\begin{aligned}
&\forall\, u : U;\ (rs, n) : SSSoD \bullet \\
&\quad \#(rs \cap UA \llparenthesis \{u\} \rrparenthesis) \leq n \land \\
&\quad \bigcap \{r : ROLE;\ mfrs : \mathbb{P}\, ROLE;\ ps : \mathbb{P}\, PERMISSION \mid \\
&\qquad mfrs \subseteq rs;\ \#mfrs = n + 1;\ r \in mfrs;\ ps = PA \llparenthesis \{r\} \rrparenthesis \bullet ps\} = \varnothing
\end{aligned}$$

Otherwise the constraints do not have any effect:

$\forall (rs, n) : SSoD \bullet \#(rs) > n$
$\forall (rs, n) : SSSoD \bullet \#(rs) > n$
$\forall (ps, n) : SSoDP \bullet \#(ps) > n$
$\forall (rs, n) : DSoD \bullet \#(rs) > n$

*AddUA*

$$\begin{array}{l}
\forall (rs, n) : \mathbb{P}(\mathbb{P}\, ROLE \times \mathbb{N}) \mid \\
\quad (rs, n) \in SSoD \cup SSSoD \bullet \\
\quad \#(rs \cap UA' \llparenthesis \{u?\} \rrparenthesis) \leq n
\end{array}$$

*AddPA*

$$\begin{array}{l}
\forall (ps, n) : SSoDP \bullet \\
\quad \#(ps \cap PA' \llparenthesis \{r?\} \rrparenthesis) \leq n \\
\forall (rs, n) : SSSoD \bullet \\
\quad \bigcap \{ r : rs;\ ps : \mathbb{P}\, PERMISSION \mid \\
\quad ps = PA' \llparenthesis \{r\} \rrparenthesis \bullet ps \} = \varnothing
\end{array}$$

*AddSSoD*

$$\begin{array}{l}
rs? : \mathbb{P}\, ROLE \\
n? : \mathbb{N} \\
\hline
rs? \notin \operatorname{dom} SSoD \\
\#(rs?) > n? \\
SSoD' = SSoD \cup \{ rs? \mapsto n? \}
\end{array}$$

*DeleteSSoD*

$$\begin{array}{l}
rs? : \mathbb{P}\, ROLE \\
\hline
rs? \in \operatorname{dom} SSoD \\
SSoD' = SSoD \setminus \{ rs? \mapsto SSoD(rs?) \}
\end{array}$$

*AddSSoDP*

$$\begin{array}{l}
ps? : \mathbb{P}\, PERMISSION \\
n? : \mathbb{N} \\
\hline
ps? \notin \operatorname{dom} SSoDP \\
\#(ps?) > n? \\
SSoDP' = SSoDP \cup \{ ps? \mapsto n? \}
\end{array}$$

*DeleteSSoDP*

$$\begin{array}{l}
ps? : \mathbb{P}\, PERMISSION \\
\hline
ps? \in \operatorname{dom} SSoDP \\
SSoDP' = SSoDP \setminus \{ rs? \mapsto SSoDP(rs?) \}
\end{array}$$

*AddSSSoD*

$$\begin{array}{l}
rs? : \mathbb{P}\, ROLE \\
n? : \mathbb{N} \\
\hline
rs? \notin \operatorname{dom} SSSoD \\
\#(rs?) > n? \\
SSSoD' = SSSoD \cup \{ rs? \mapsto n? \}
\end{array}$$

*DeleteSSSoD*

$$\begin{array}{l}
rs? : \mathbb{P}\, ROLE \\
\hline
rs? \in \operatorname{dom} SSSoD \\
SSSoD' = SSSoD \setminus \{ rs? \mapsto SSSoD(rs?) \}
\end{array}$$

*AddDSoD*

$$\begin{array}{l}
rs? : \mathbb{P}\, ROLE \\
n? : \mathbb{N} \\
\hline
rs? \notin \operatorname{dom} DSoD \\
\#(rs?) > n? \\
DSoD' = DSoD \cup \{ rs? \mapsto n? \}
\end{array}$$

*DeleteDSoD*

$$\begin{array}{l}
rs? : \mathbb{P}\, ROLE \\
\hline
rs? \in \operatorname{dom} DSoD \\
DSoD' = DSoD \setminus \{ rs? \mapsto DSoD(rs?) \}
\end{array}$$

*SepDutyEngine*

$\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject},$
$\quad \mathit{ActivateRole}, \mathit{DeactivateRole})$

$\mathit{RBACCoreEngine}$

Role activation history for a subject:

$\mathit{History}_{SR} : S \leftrightarrow R$

$\mathit{Policy} \in \downarrow \mathit{SepDutyPolicy}$

$\forall\, u : \mathit{Policy}.U;\ (rs, n) : \mathit{Policy}.\mathit{DSoD} \bullet \#(rs \cap \mathit{History}_{SR}(\!|\ SU^{-1}(\!|\ \{u\}\ |\!)\ |\!)) \leq n$

$SR \subseteq \mathit{History}_{SR}$

*ImportPolicy*

$\mathit{Policy}? \in \downarrow \mathit{SepDutyPolicy}$

*ExportPolicy*

$\mathit{Policy}! \in \downarrow \mathit{SepDutyPolicy}$

*CreateSubject*

$\Delta(\mathit{History}_{SR})$

$s? : \mathit{SUBJECT}$

$u? : \mathit{USER}$

$rs? : \mathbb{P}\, \mathit{ROLE}$

$\forall (rs, n) : \mathit{Policy}.\mathit{DSoD} \bullet \#(rs \cap \mathit{History}'_{SR}(\!|\ SU^{-1}(\!|\ SU(s?)\ |\!)\ |\!)) \leq n$

$\mathit{History}'_{SR} = \mathit{History}_{SR} \cup \{r : rs? \bullet s? \mapsto r\}$

*ActivateRole*

$\Delta(\mathit{History}_{SR})$

$r? : \mathit{ROLE}$

$s? : \mathit{SUBJECT}$

$u? : \mathit{USER}$

$\forall (rs, n) : \mathit{Policy}.\mathit{DSoD} \bullet \#(rs \cap \mathit{History}'_{SR}(\!|\ SU^{-1}(\!|\ SU(s?)\ |\!)\ |\!)) \leq n$

$\mathit{History}'_{SR} = \mathit{History}_{SR} \cup \{s? \mapsto r?\}$

*DestroySubject*

$\Delta(\mathit{History}_{SR})$

$s? : \mathit{SUBJECT}$

$u? : \mathit{USER}$

$\mathit{History}'_{SR} = \mathit{History}_{SR} \setminus \{r : \mathit{ROLE} \mid r \in SR(\!|\ \{s?\}\ |\!) \bullet s? \mapsto r\}$

*SepDutyRHPolicy*

*RoleHierarchyPolicy*

$$SSoD : (\mathbb{P}\, ROLE) \mathrel{⇻} \mathbb{N}$$
$$DSoD : (\mathbb{P}\, ROLE) \mathrel{⇻} \mathbb{N}$$

$$\forall\, u : U;\ (rs, n) : SSoD \bullet \#(rs \cap \_ \succeq_{RH} \_ \llparenthesis\ UA \llparenthesis\ \{u\}\ \rrparenthesis\ \rrparenthesis) \leq n$$

Otherwise the constraints do not have any effect:

$$\forall (rs, n) : SSoD \bullet \#(rs) > n$$
$$\forall (rs, n) : DSoD \bullet \#(rs) > n$$

*AddUA*

$$\forall (rs, n) : SSoD \bullet \#(rs \cap \_ \succeq_{RH} \_ \llparenthesis\ UA' \llparenthesis\ \{u?\}\ \rrparenthesis\ \rrparenthesis) \leq n$$

*AddSSoD*

$$rs? : \mathbb{P}\, ROLE$$
$$n? : \mathbb{N}$$

$$rs? \notin \operatorname{dom} SSoD$$
$$\#(rs?) > n?$$
$$SSoD' = SSoD \cup \{rs? \mapsto n?\}$$

*DeleteSSoD*

$$rs? : \mathbb{P}\, ROLE$$

$$rs? \in \operatorname{dom} SSoD$$
$$SSoD' = SSoD \setminus \{rs? \mapsto SSoD(rs?)\}$$

*AddDSoD*

$$rs? : \mathbb{P}\, ROLE$$
$$n? : \mathbb{N}$$

$$rs? \notin \operatorname{dom} DSoD$$
$$\#(rs?) > n?$$
$$DSoD' = DSoD \cup \{rs? \mapsto n?\}$$

*DeleteDSoD*

$$rs? : \mathbb{P}\, ROLE$$

$$rs? \in \operatorname{dom} DSoD$$
$$DSoD' = DSoD \setminus \{rs? \mapsto DSoD(rs?)\}$$

*SepDutyRHEngine*

$$\upharpoonright(ImportPolicy, ExportPolicy, CheckAccess, CreateSubject, DestroySubject,$$
$$ActivateRole, DeactivateRole)$$

*RoleHierarchyEngine*

$$Policy \in \downarrow SepDutyRHPolicy$$
$$\forall\, u : Policy.U;\ (rs, n) : Policy.DSoD \bullet \#(rs \cap \_ \succeq_{RH} \_ \llparenthesis\ SR \llparenthesis\ SU^{-1}(u)\ \rrparenthesis\ \rrparenthesis) \leq n$$

*ImportPolicy*

$$Policy? \in \downarrow SepDutyRHPolicy$$

*ExportPolicy*

$$Policy! \in \downarrow SepDutyRHPolicy$$

*CreateSubject*

$$\forall (rs, n) : Policy.DSoD \bullet$$
$$\#(rs \cap \_ \succeq_{RH} \_ \llparenthesis\ SR' \llparenthesis\ \{s?\}\ \rrparenthesis\ \rrparenthesis \leq n$$

*ActivateRole*

$$\forall (rs, n) : Policy.DSoD \bullet$$
$$\#(rs \cap \_ \succeq_{RH} \_ \llparenthesis\ SR' \llparenthesis\ \{s?\}\ \rrparenthesis\ \rrparenthesis \leq n$$

**Interpretation of basic types** This module does not introduce any new basic types.

**Notes on Operation Invocation** This module does not introduce any new operations other than administrative operations.

**Informal Summary** The constraints introduced by *SepDuty* has the following informal semantics:

- **Static Separation of Duty:** The function *SSoD* assigns a natural number $n$ to a set of critical roles. The class invariant of *SepDutyEngine* and the additional constraints posed on *AddUA* ensure that at most $n$ critical roles may be assigned to user via *UA*.
- **Static Separation of Duty for Permissions:** The function *SSoDP* assigns a natural number $n$ to a set of critical permissions. The class invariant of *SepDutyEngine* and the additional constraints posed on *AddPA* ensure that at most $n$ critical permissions may be assigned to user via *PA*.
- **Strict Static Separation of Duty:** The function *SSSoD* assigns a natural number $n$ to a set of critical roles. In addition to the constraints posed by the static separation of duty principles (as described above) it ensures that critical roles are not allowed to have overlapping permissions.
- **Dynamic Separation of Duty:** The function *DSoD* assigns a natural number $n$ to a set of critical roles. The class invariant of *SepDutyEngine* and the additional constraints posed on *CreateSubject* and *ActivateRole* ensure that at most $n$ critical roles may be activated at a time by a user (not even by means of two different subjects). During the life time of a subject it is not possible to activate critical roles one after another.

In contrast, *SepDutyRH* supports currently only two variants of separation of duty:

- **Static Separation of Duty:** The function *SSoD* assigns a natural number to a set of critical roles. The class invariant of *SepDutyRH* and the additional constraints posed on *AddUA* ensure that the assigned roles of a user together with their junior roles have only $n$ members in common with the critical roles.
- **Dynamic Separation of Duty:** The function *DSoD* assigns a natural number to a set of critical roles. The class invariant of *SepDutyRH* and the additional constraints posed on *ActivateRole* and *CreateSubject* ensure that the active roles of a user together with their junior roles have only $n$ members in common with the critical roles.

**Remarks**

- The corresponding OPL/XML data structures of this policy modules are defined in Sections 6.3 and 6.4.
- The separation of duty properties for this module are partially derived from Gligor et. al. [15] and from the ANSI Standard [27]. Further variants of separation of duty properties can be easily added if necessary.

## 4.5 Chinese Wall and Object-based Separation of Duty

### Formal Definition

$\mathit{ChineseWallPolicy}$

$\mathit{RBACCorePolicy}$

Set of chinese wall partitionings over objects:

$CW : \mathbb{P}\,\mathrm{seq}\,\mathbb{P}\,\mathit{OBJECT}$

Assignment of objects to users indicating which chinese wall partitions the user is associated with:

$UO_{CW} : U \leftrightarrow \mathit{OBJECT}$

We have indeed a partitioning:

$\forall\, cw : CW \bullet \exists\, os : \mathbb{P}\,\mathit{OBJECT} \bullet cw \,\mathrm{partitions}\, os$

All objects in UOcw are in some partition:

$$\begin{array}{l}\forall\, ob : \mathit{OBJECT} \mid ob \in \mathrm{ran}\, UO_{CW} \bullet \\ \quad \exists\, cw : CW;\; obpt : \mathbb{P}\,\mathit{OBJECT} \mid \langle obpt \rangle \,\mathrm{in}\, cw \bullet ob \in obpt\end{array}$$

$\mathit{ChineseWallEngine}$

$$\begin{array}{l}\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject}, \\ \quad \mathit{ActivateRole}, \mathit{DeactivateRole})\end{array}$$

$\mathit{RBACCoreEngine}$

$\mathit{Policy} \in {\downarrow}\mathit{ChineseWallPolicy}$

$\mathit{ImportPolicy}$

$\mathit{Policy}? \in {\downarrow}\mathit{ChineseWallPolicy}$

$\mathit{ExportPolicy}$

$\mathit{Policy}! \in {\downarrow}\mathit{ChineseWallPolicy}$

$\mathit{Authorization}$

If a user is assigned to a certain chinese wall partion, then the requested object must be in this partition:

$$\begin{array}{l}\forall\, cw : \mathit{Policy}.CW;\; obpt : \mathbb{P}\,\mathit{OBJECT} \mid \langle obpt \rangle \,\mathrm{in}\, cw \bullet \\ \quad \mathit{Policy}.UO_{CW}(\!|\; SU(req?.s) \;|\!) \cap pt \neq \varnothing \\ \qquad \Rightarrow \mathit{Env}.\mathit{instanceof}_{obj}(req?.obi) \in obpt\end{array}$$

$\mathit{CommitAccess}$

$\Delta(\mathit{Policy})$

$req? : {\downarrow}\mathit{Request}$

If the given object is in a partition of a certain chinese wall partitioning and this partition is not yet assigned to the user (i.e. first request), then the object is assigned to the user in order to remember the partition for future requests.

$$\begin{array}{l}\forall\, cw : \mathit{Policy}.CW;\; obpt : \mathbb{P}\,\mathit{OBJECT} \mid \langle obpt \rangle \,\mathrm{in}\, cw \bullet \\ \quad \mathit{Policy}.UO_{CW}(\!|\; SU(req?.s) \;|\!) \cap obpt = \varnothing \\ \quad \wedge (req?.op, \mathit{Env}.\mathit{instanceof}_{obj}(req?.obi)) \in obpt \\ \quad \Rightarrow \mathit{Policy}.UO'_{CW} = \mathit{Policy}.UO_{CW} \cup \{SU(req?.s) \mapsto \mathit{Env}.\mathit{instanceof}_{obj}(req?.obi)\}\end{array}$$

*ObjSepDutyEnv*

History which users have accessed which object instance by which operation.

$UOO_{ObjSoD} : USER \leftrightarrow OBJECTINSTANCE \leftrightarrow OPERATION$

*ObjSepDutyPolicy*

For these objects the ObjSepDuty property is supposed to be fulfilled.

$ObjSoD : \mathbb{P}\ OBJECT$

*ObjSepDutyEngine*

$RBACCoreEngine$

*Authorization*

$$\begin{aligned}&\exists\, op : OPERATION \mid (SU(req?.s), req?.obi, op) \in Env.UOO_{ObjSoD} \bullet\\ &\quad Env.instanceof_{obj}(req?.obi) \in Policy.ObjSoD \Rightarrow\\ &\quad req?.op = op\end{aligned}$$

**Interpretation of basic types** This module does not introduce any new basic types.

### Notes on Operation Invocation

- **CommitAccess:** while CheckAccess only checks whether a request can be granted, CommitAccess is invoked by the PEP in order to inform the policy engine that a certain access request has actually been performed. Since the PEP is trusted, any access requests can be committed.

### Informal Summary

- Chinese Wall is a partitioning of objects in such a way that once a user accessed an object that is in some partition, he or she is bound to that partition. I.e. the user cannot access any objects included in other partitions of the same partitioning but he or she is able to access object outside of the whole partitioning.

- Informally, object-based separation of duty means that for all objects (i.e. object types) $T$ in *ObjSoD* the following shall be true. If a user $u$ has accessed an object instance $oi$ of the type $T$ by means of the operation $op$, the from that point in time the user is only allowed to access $oi$ be means of the operation $op$.

### Remarks

- The corresponding OPL/XML data structures of this policy modules are defined in Sections 6.6 and 6.7.

- Special about the Chinese Wall module is that it contains policy changing engine operations. Hence, a granted request may entail a policy change.

## 4.6 Workflow-Related Constraints

### Formal Definition

$$[\mathit{WFTEMPLATE}, \mathit{WFINSTANCE}, \mathit{TASK}, \mathit{TASKINSTANCE}]$$

*WFCoreRequest*

$$\mathit{RBACCoreRequest}$$

$$\mathit{ti} : \mathit{TASKINSTANCE}$$

*WFCoreEnv*

$$\begin{array}{l}
\mathit{wftemplates} : \mathbb{P}\,\mathit{WFTEMPLATE} \\
\mathit{tasks} : \mathbb{P}\,\mathit{TASK} \\
\mathit{wfinstances} : \mathbb{P}\,\mathit{WFINSTANCE} \\
\mathit{taskinstances} : \mathbb{P}\,\mathit{TASKINSTANCE} \\
\mathit{instanceof}_{\mathit{wf}} : \mathit{wfinstances} \rightarrow \mathit{wftemplates} \\
\mathit{instanceof}_{\mathit{task}} : \mathit{taskinstances} \rightarrow \mathit{tasks} \\
\mathit{belongsto}_{\mathit{inst}} : \mathit{taskinstances} \rightarrow \mathit{wfinstances} \\
\mathit{belongsto}_{\mathit{tmpl}} : \mathit{tasks} \rightarrow \mathit{wftemplates} \\
\mathit{History}_{\mathit{WFI}} : \mathrm{seq}(\mathit{WFINSTANCE} \times \mathit{USER} \times \mathit{TASK})
\end{array}$$

$$\begin{array}{l}
\mathit{instanceof}_{\mathit{wf}} \circ \mathit{belongsto}_{\mathit{inst}} = \mathit{belongsto}_{\mathit{tmpl}} \circ \mathit{instanceof}_{\mathit{task}} \\
\forall (\mathit{wfi}, u, t) : \mathit{WFINSTANCE} \times \mathit{USER} \times \mathit{TASK} \mid \langle (\mathit{wfi}, u, t) \rangle \;\mathrm{in}\; \mathit{History}_{\mathit{WFI}} \bullet \\
\quad \mathit{belongsto}_{\mathit{tmpl}}(t) = \mathit{instanceof}_{\mathit{wf}}(\mathit{wfi})
\end{array}$$

*WFCorePolicy*

*RBACCorePolicy*

$TPA : TASK \leftrightarrow P$

$TRA : TASK \leftrightarrow R$

The synchronization problem:

$\mathrm{dom}\ TPA \subseteq Env.tasks$

$\mathrm{dom}\ TRA \subseteq Env.tasks$

Every role needs the rights necessary to execute the operations associated with the task (vertical/horizontal alignment):

$\forall (t, r) : TRA \bullet TPA(\!|\ \{t\}\ |\!) \subseteq PA(\!|\ \{r\}\ |\!)$

*AddTPA*

$\Delta(TPA)$

$task? : TASK$

$p? : PERMISSION$

$p? \in P;\ task? \in Env.tasks$

$(task?, p?) \notin TPA$

$\forall r : ROLE \mid (task?, r) \in TRA \bullet$

$\quad (r, p?) \in PA$

$TPA' = TPA \cup \{task? \mapsto p?\}$

*DeleteTPA*

$\Delta(TPA)$

$task? : TASK$

$p? : PERMISSION$

$(task?, p?) \in TPA$

$TPA' = TPA \setminus \{task? \mapsto p?\}$

*AddTRA*

$\Delta(TRA)$

$task? : TASK$

$r? : ROLE$

$r? \in R;\ task? \in Env.tasks$

$(task?, r?) \notin TRA$

$\forall p : PERMISSION \mid (task?, p) \in TPA \bullet$

$\quad (r?, p) \in PA$

$TRA' = TRA \cup \{task? \mapsto r?\}$

*DeleteTRA*

$\Delta(TRA)$

$task? : TASK$

$r? : ROLE$

$(task?, r?) \in TRA$

$TRA' = TRA \setminus \{task? \mapsto r?\}$

*WFCoreEngine*

$\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject},$
$\quad \mathit{ClaimTI}, \mathit{ReleaseTI}, \mathit{ActivateRole}, \mathit{DeactivateRole})$

$\mathit{RBACCoreEngine}$

$\mathit{claimedby} : \mathit{TASKINSTANCES} \nrightarrow S$

$\mathit{Policy} \in \downarrow \mathit{WFCorePolicy}$
$\mathrm{dom}\ \mathit{claimedby} \subseteq \mathit{Env.taskinstances}$

*ImportPolicy*

$\mathit{Policy}? \in \downarrow \mathit{WFCorePolicy}$

*ExportPolicy*

$\mathit{Policy}! \in \downarrow \mathit{WFCorePolicy}$

*ClaimTI*

$\Delta(\mathit{claimedby})$
$s? : \mathit{SUBJECT}$
$ti? : \mathit{TASKINSTANCE}$

$\mathit{claimedby}' = \mathit{claimedby} \cup \{ti? \mapsto s?\}$
$\exists\, r : \mathit{ROLE} \mid (s?, r) \in \mathit{SR} \bullet (\mathit{Env.instanceof}_{\mathit{task}}(ti?), r) \in \mathit{Policy.TRA}$

*ReleaseTI*

$s? : \mathit{SUBJECT}$
$ti? : \mathit{TASKINSTANCE}$

$\mathit{claimedby}' = \mathit{claimedby} \setminus \{ti? \mapsto s?\}$

*Authorization*

$req? : \downarrow \mathit{RBACCoreRequest}$

$req? \in \mathit{WFCoreRequest}$
$req?.ti\ \underline{\mathit{claimedby}}\ req?.s$
$(req?.op, req?.ob) \in \mathit{Policy.TPA}\llparenthesis\, \{\mathit{Env.instanceof}_{\mathit{task}}(req?.ti)\}\, \rrparenthesis$

$WFSepDutyPolicy$

$WFCorePolicy$

History-based Dynamic Separation of Duty (Syncless):

$HDSoDSL : \mathbb{P}\ WFTEMPLATE$

History-based Dynamic Separation of Duty (Simple):

$HDSoD : (\mathbb{P}\ TASK) \nrightarrow \mathbb{N}$

History-based Dynamic Separation of Duty (Task Partions):

$HDSoDTP : \mathbb{P}\,\mathrm{seq}\,\mathbb{P}\ TASK$

All tasks in a critical task set must belong to exactly one workflow template. Futhermore, the number of tasks in a critical task set must exceed the number of allowed task claims from this set:

$\forall (ts, n) : HDSoD \bullet \#(Env.belongsto_{tmpl} \llparenthesis\ ts\ \rrparenthesis) = 1 \land \#(ts) > n$

All task partions partion a task set that belongs to one workflow template:

$$\forall\, tp : HDSoDTP \bullet \exists\, ts : \mathbb{P}\ TASK \bullet \\ \quad tp \text{ partitions } ts \land \#(Env.belongsto_{tmpl} \llparenthesis\ ts\ \rrparenthesis) = 1$$

$\mathit{WFSepDutyEngine}$

$\upharpoonleft(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject},$
$\quad \mathit{ClaimTI}, \mathit{ReleaseTI}, \mathit{ActivateRole}, \mathit{DeactivateRole})$

$\mathit{WFCoreEngine}$

$\mathit{Policy} \in \downarrow \mathit{WFSepDutyPolicy}$

$\mathit{ImportPolicy}$

$\mathit{Policy}? \in \downarrow \mathit{WFSepDutyPolicy}$

$\mathit{ExportPolicy}$

$\mathit{Policy}! \in \downarrow \mathit{WFSepDutyPolicy}$

$\mathit{ClaimTI}$

$s? : \mathit{SUBJECT}$
$ti? : \mathit{TASKINSTANCE}$

$$\begin{array}{l}
\mathit{Env.belongsto}_{tmpl} \circ \mathit{Env.instanceof}_{task}(ti?) \in \mathit{Policy.HDSoDSL} \Rightarrow \\
\quad \exists\, t : \mathit{TASK} \bullet \\
\qquad \neg(\langle(\mathit{Env.belongsto}_{inst}(ti?), \mathit{SU}(s?), t)\rangle \mathrel{\mathrm{in}} \mathit{Env.History}_{WFI}) \land \\
\qquad t \neq \mathit{Env.instanceof}_{task}(ti?) \land \\
\qquad \mathit{Env.belongsto}_{tmpl}(t) = \mathit{Env.belongsto}_{tmpl} \circ \mathit{Env.instanceof}_{task}(ti?) \\
\forall (ts, n) : \mathit{Policy.HDSoD} \mid \mathit{Env.instanceof}_{task}(ti?) \in ts \bullet \\
\quad \#\{t : \mathit{TASK} \mid \\
\qquad t \in ts; \\
\qquad t \neq \mathit{Env.instanceof}_{task}(ti?); \\
\qquad \langle(\mathit{Env.belongsto}_{inst}(ti?), \mathit{SU}(s?), t)\rangle \mathrel{\mathrm{in}} \mathit{Env.History}_{WFI} \\
\quad \} < n
\end{array}$$

If there exists a task in the history that is included in a task set which is part of any critical task partition, then the requested task must also be included in this task set.

$$\begin{array}{l}
\forall\, tp : \mathit{Policy.HDSoDTP};\ ts : \mathbb{P}\,\mathit{TASK} \mid \langle ts\rangle \mathrel{\mathrm{in}} tp \bullet \exists\, t : \mathit{TASK} \bullet \\
\quad t \in ts \land \langle(\mathit{Env.belongsto}_{inst}(ti?), \mathit{SU}(s?), t)\rangle \mathrel{\mathrm{in}} \mathit{Env.History}_{WFI} \Rightarrow \\
\quad \mathit{Env.instanceof}_{task}(ti?) \in ts
\end{array}$$

$\mathit{WFSepDutyCCPolicy}$

$\mathit{WFCorePolicy}$

$\mathit{ExoContextPolicy}$

$\mathit{HDSoDTPCC} : \mathbb{P}(\mathrm{seq}\,\mathbb{P}\,\mathit{TASK} \times \mathit{CC})$

All task partions partion a task set that belongs to one workflow template:

$$\begin{array}{l}
\forall (tp, cc) : \mathit{HDSoDTPCC} \bullet \exists\, ts : \mathbb{P}\,\mathit{TASK} \bullet \\
\quad tp \mathrel{\mathrm{partitions}} ts \land \#(\mathit{Env.belongsto}_{tmpl}(\!|\ ts\ |\!)) = 1
\end{array}$$

*WFSepDutyCCEngine*

$\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject},$
$\quad \mathit{ClaimTI}, \mathit{ReleaseTI}, \mathit{ActivateRole}, \mathit{DeactivateRole})$

$\mathit{WFCoreEngine}[\mathit{oldClaimTI} / \mathit{ClaimTI}]$

$\mathit{ExoContextEngine}$

$\mathit{Policy} \in \downarrow \mathit{WFSepDutyCCPolicy}$

*ImportPolicy*

$\mathit{Policy}? \in \downarrow \mathit{WFSepDutyCCPolicy}$

*ExportPolicy*

$\mathit{Policy}! \in \downarrow \mathit{WFSepDutyCCPolicy}$

*RetHDSoDTPCC*

$s? : \mathit{SUBJECT}$
$ti? : \mathit{TASKINSTANCE}$
$ccs! : \mathbb{P}\ \mathit{ContextConstraint}$

A context constraint is selected for evaluation whenever a task is requested that might be subject to an HDSoD constraint associated to the context constraint:

$ccs! = \{(tp, cc) : \mathit{Policy.HDSoDTPCC};\ ts : \mathbb{P}\ \mathit{TASK} \mid$
$\quad \langle ts \rangle \text{ in } tp \land \mathit{Env.instanceof}_{\mathit{task}}(ti?) \in ts \bullet cc\}$

*EnforceHDSoDTP*

$s? : \mathit{SUBJECT}$
$ti? : \mathit{TASKINSTANCE}$

If there exists a task in the history that is included in a task set which is part of any critical task partition, then the requested task must also be included in this task set.

$\forall (tp, cc) : \mathit{Policy.HDSoDTPCC};\ ts : \mathbb{P}\ \mathit{TASK} \mid \langle ts \rangle \text{ in } tp \bullet \exists\, t : \mathit{TASK} \bullet$
$\quad t \in ts \land \langle(\mathit{Env.belongsto}_{\mathit{inst}}(ti?), \mathit{SU}(s?), t)\rangle \text{ in } \mathit{Env.History}_{\mathit{WFI}} \Rightarrow$
$\quad \mathit{Env.instanceof}_{\mathit{task}}(ti?) \in ts$

$\mathit{ClaimTI} \mathrel{\widehat=} \mathit{oldClaimTI}$
$\quad \land$
$\quad (\mathit{RetHDSoDTPCC} \mathbin{⨾} \mathit{EvalContextConstraints}) \Rightarrow \mathit{EnforceHDSoDTP}$

*WFCardinalityPolicy*

$\mathit{WFCorePolicy}$

$\mathit{Card} : \mathit{TASK} \rightarrowtail\!\!\!\rightarrow \mathbb{N}$

*WFCardinalityEngine*

$$\begin{array}{l}
\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject}, \\
\quad \mathit{ClaimTI}, \mathit{ReleaseTI}, \mathit{ActivateRole}, \mathit{DeactivateRole}) \\
\mathit{WFCoreEngine}
\end{array}$$

$$\mathit{Policy} : \downarrow \mathit{RBACCorePolicy}$$

*ImportPolicy*

$$\mathit{Policy}? \in \downarrow \mathit{WFCardinalityPolicy}$$

*ExportPolicy*

$$\mathit{Policy}! \in \downarrow \mathit{WFCardinalityPolicy}$$

*ClaimTI*

$$\begin{array}{l}
s? : \mathit{SUBJECT} \\
ti? : \mathit{TASKINSTANCE} \\
\hline
\exists\, n : \mathbb{N} \bullet \\
\quad (\mathit{Env.instanceof}_{\mathit{task}}(ti?) \mapsto n) \in \mathit{Policy.Card} \Rightarrow \\
\quad \#( \\
\qquad \mathit{Env.History}_{\mathit{WFI}} \upharpoonright \\
\qquad \{(\mathit{Env.belongsto}_{\mathit{inst}}(ti?), \mathit{SU}(s?), \mathit{Env.instanceof}_{\mathit{task}}(ti?))\} \\
\quad ) < n
\end{array}$$

*WFBindDutyPolicy*

$$\mathit{WFCorePolicy}$$

$$\mathit{BoD} : \mathit{TASK} \rightarrowtail\!\!\!\!\!\rightarrow \mathit{TASK}$$

*WFBindDutyEngine*

$$\begin{array}{l}
\upharpoonright(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject}, \\
\quad \mathit{ClaimTI}, \mathit{ReleaseTI}, \mathit{ActivateRole}, \mathit{DeactivateRole}) \\
\mathit{WFCoreEngine}
\end{array}$$

$$\mathit{Policy} \in \downarrow \mathit{WFBindDutyPolicy}$$

*ImportPolicy*

$$\mathit{Policy}? \in \downarrow \mathit{WFBindDutyPolicy}$$

*ExportPolicy*

$$\mathit{Policy}! \in \downarrow \mathit{WFBindDutyPolicy}$$

*ClaimTI*

$$\begin{array}{l}
s? : \mathit{SUBJECT} \\
ti? : \mathit{TASKINSTANCE} \\
\hline
\exists\, t : \mathit{TASK} \mid (t, \mathit{Env.instanceof}_{\mathit{task}}(ti?)) \in \mathit{Policy.BoD} \bullet \\
\quad \exists\, u : \mathit{USER};\ \mathit{wfi} : \mathit{WFINSTANCE} \mid \langle(\mathit{wfi}, u, t)\rangle \text{ in } \mathit{Env.History}_{\mathit{WFI}} \bullet \\
\qquad u = \mathit{SU}(s?)
\end{array}$$

*WFPrereqStepPolicy*

*WFCorePolicy*

$\mathit{PrereqStep} : \mathit{TASK} \leftrightarrow \mathit{TASK}$

*WFPrereqStepEngine*

$\upharpoonleft(\mathit{ImportPolicy}, \mathit{ExportPolicy}, \mathit{CheckAccess}, \mathit{CreateSubject}, \mathit{DestroySubject},$
$\quad \mathit{ClaimTI}, \mathit{ReleaseTI}, \mathit{ActivateRole}, \mathit{DeactivateRole})$

*WFCoreEngine*

$\mathit{Policy}? \in \downarrow \mathit{WFPrereqStepPolicy}$

*ImportPolicy*

$\mathit{Policy}? \in \downarrow \mathit{WFPrereqStepPolicy}$

*ExportPolicy*

$\mathit{Policy}! \in \downarrow \mathit{WFPrereqStepPolicy}$

*ClaimTI*

$s? : \mathit{SUBJECT}$
$ti? : \mathit{TASKINSTANCE}$

$\forall\, t : \mathit{TASK} \mid (t, \mathit{Env.instanceof}_{task}(ti?)) \in \mathit{Policy.PrereqStep} \bullet \exists\, u : \mathit{USER} \bullet$
$\quad \langle(\mathit{Env.belongsto}_{inst}(ti?), u, t)\rangle \text{ in } \mathit{Env.History}_{WFI}$

**Interpretation of Basic Types** This class of modules introduces four new basic types:

- **Workflow Template:** the elements of *WFTEMPLATE* can be interpreted as identifiers to the workflow schema definitions (also called process models) of the workflow management system (WFMS).
- **Workflow Instance:** the elements of *WFINSTANCE* can be interpreted as identifiers to the actual instances of the workflows run by the WFMS. Thus, they are instances of the workflow templates of the WFMS. There may be multiple workflow instances of one workflow template.
- **Task:** the elements of *TASK* can be interpreted as identifiers to the steps of a workflow template.
- **Task Instance:** the elements of *TASKINSTANCE* can be interpreted as identifiers to the actual instances of workflow steps run by the WFMS. They are instances of the tasks. Again, there may be multiple task instances of one task.

**Notes on Operation Invocation** This class of modules introduces two new operations in its Engine section:

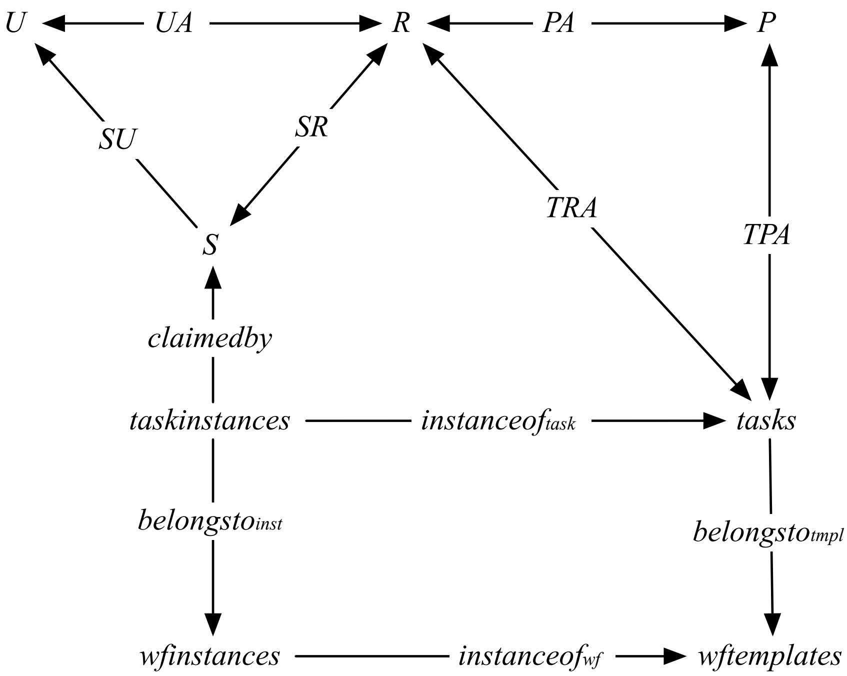
- **ClaimTI:** in order to be able to execute a task instance, a subject needs to claim the task instance first. This association between the task instances and subjects is necessary because authorizations can then be put in the context of a workflow instance. Furthermore, history-based separation of duty and related concepts can be specified.

  ClaimTI usually is invoked when a work item is selected from the work-list handler.

- **ReleaseTI:** A task instance is released whenever it is completed or aborted. There is no need to make a distinction between these two cases, because completed task instances appear in the history of the WFMS.

**Informal Summary** In this section we introduce a class of modules related to the topic workflows, instead of just one module. While the core module *WFCore* provides the basic concepts of workflow authorization the additional modules add workflow-related authorization constraints such as history-based separation of duty.

The following figure illustrates the state space of the core module *WFCore* and its relationship to a WFMS.



The WFCoreEnv environment consists of the relationships and the states of the task instances, the tasks, the workflow instances, and the workflow templates. These sets are linked with the authorization engine by means of three relations—namely *claimedby*, *TRA*, and *TPA*.

The *claimedby* relation contains the association of task instances and subjects which is necessary because access decisions for a subject can then be put in the context of a workflow instance. Furthermore, principles such as history-based separation of duty work in such a way that a subject might not be able to "claim" a workflow instance if this violates an history-based separation of duty constraint.

The relations *TRA* (task-role-assignment) and *TPA* (task-permission-assignment), which are particularly part of the policy, specify the relationship between tasks, roles, and permissions. Every task is assigned a set of permissions (via *TPA*) that are necessary in order to carry out this task. For example, for an "approve contract" task the permissions "read document" and "sign document" might be necessary. Once a task instance is claimed by a subject, the policy engine ensures that within the context of a task instance only the permissions associated with the task are granted (cf. *Authorization*). Via *TRA* a task is assigned to a set of roles that are allowed to claim the task (cf. *ClaimTI*). The class invariant $\forall(t,r) : \mathit{TRA} \bullet \mathit{TPA}(\!|\ \{t\}\ |\!) \subseteq \mathit{PA}(\!|\ \{r\}\ |\!)$ of *WFCoreEngine* ensures that every role associated with a certain task has as least the privileges (assigned via *PA*) necessary to execute the task.

The modules *WFSepDuty*, *WFCardinality*, *WFBindDuty*, and *WFPrereqStep* work in such a way that they augment *ClaimTI* with the constraints necessary for the particular workflow authorization principle:

- *WFSepDuty* and *WFSepDutyCC* add history-based separation of duty for workflows in four different versions:
  - **HDSoDSL:** for all $wftempl : WFTemplates$ in the data structure *HDSoDSL* the following shall be true: no user is able to execute all tasks belonging to *wftempl* within one workflow instance.
  - **HDSoD:** from a given set of critical tasks a user is only allowed to execute $N$ different tasks within a workflow instance.
  - **HDSoDTP:** note that *HDSoDTP* is a set of partitionings. Whenever a user has executed a task which is included in some partition of such a partitioning, he or she is bound to this partition in the future with regard to the specific workflow instance. I.e. the user is not allowed to executed any tasks which are included in other partitions of the same partitioning. However, the user is still allowed to execute tasks outside of the whole partitioning.
  - **HDSoDTPCC:** this data structure imposes the same constraint as *HDSoDTP* with the following difference. The history-based separation of duty constraint must only be fulfilled if the assigned context constraint is fulfilled. E.g. if we have an industrial customer, the workflow must be executed according to the mentioned history-based separation of duty property.

  by checking if for a workflow instance there is still another task instance left once the requested task instance is claimed.

- With *WFCardinality* it is possible to confine the number of task instances for a certain task in a workflow instance: in such a case, a task instance can only be claimed if the history does not already contain a certain number of entries for this task in the context of a certain workflow instance.

- By means of *WFBindDuty* the policy engine can force a certain user to be responsible for a certain task if he or she has completed a certain other task: only this user is then allowed to claim the task with the binding duty. This can be seen as a certain form of obligation in a workflow sense.

- Finally, with *WFPrereqStep* a constraint can be placed in such a way that a certain task can only be claimed if a certain other task has already be completed by some user.

One may think of further modules and it will probably be necessary when modeling real world authorization requirements. This set of modules serves as a good starting point for such an application.

Finally, we prove some properties of the workflow modules.

**Proposition 2** A subject who claimed a task instance has all permissions necessary to execute the operations associated (over *TPA*) with the corresponding task.

PROOF: Let $s : S$ and $ti : TASKINSTANCE$ such that $ti \ \underline{claimedby} \ s$. Let the corresponding task be $task : TASK \mid task = Env.instanceof_{task}(ti)$. We need to show that $s$ has the authorization to perform any permission $(op, ob) = p : PERMISSION$ that is assigned to $task$, i.e. $(task, p) \in TPA$. Hence, we need to show

$Authorization[s/s?, op/op?, ob/ob?]$

The only way to claim a task-instance (see $\Delta$-list) is by means of $ClaimTI$. Therefore, we have

$$ClaimTI[s/s?, ti/ti?]$$

from which we can follow that there exists a role $r : ROLE$ that has been activated by $s$, i.e. $(s, r) \in SR$, and that is assigned to $task$, i.e. $(task, r) \in TRA$. The class invariant of $WFCoreEngine$ ensures that for every such assignment $(task, r) \in TRA$ it holds that if $(task, p) \in TPA$ then $(r, p) \in PA$. Hence, we have $(s, r) \in SR$ and $(r, p) \in PA$ which concludes the proof. □

Finally, we argue about the following design decision formally:

$$\forall (t, r) : TRA \bullet TPA(\!|\ \{t\}\ |\!) \subseteq PA(\!|\ \{r\}\ |\!)$$

instead of

$$\forall (t, r) : TRA \bullet TPA(\!|\ \{t\}\ |\!) = PA(\!|\ \{r\}\ |\!)$$

because of the following undesirable consequence:

**Proposition 3** If

$$\forall (t, r) : TRA \bullet TPA(\!|\ \{t\}\ |\!) = PA(\!|\ \{r\}\ |\!)$$

then one $role : ROLE$ cannot be assigned to two tasks $t_1, t_2 : TASK$ with different assigned permissions (over $TPA$).

Proof: Let $p : PERMISSION$ be w.l.o.g. such that $(t_1, p) \in TPA$ but $(t_2, p) \notin TPA$. Assume that $(t_1, role) \in TRA$ and $(t_2, role) \in TRA$. By using $(t_1, p) \in TPA$ and $(t_1, role) \in TRA$ as well as the proposed formula we get

$$p \in PA(\!|\ \{role\}\ |\!)$$

By using $(t_2, role) \in TRA$ together with the proposed formula we get

$$PA(\!|\ \{role\}\ |\!) = TPA(\!|\ \{t_2\}\ |\!)$$

Hence, $p \in TPA(\!|\ \{t_2\}\ |\!)$, i.e. $(t_2, p) \in TPA$, which yields a contradiction. □

**Remarks**

- The corresponding OPL/XML data structures of this policy modules are defined in Sections 6.8, 6.9, 6.10, 6.11, 6.12, and 6.13.
- Note that $instanceof_{wf}$, $instanceof_{task}$, $belongsto_{inst}$, and $belongsto_{tmpl}$ commute as is specified by the class invariant of $WFCoreEnv$.
- Our workflow concept distinguishes between *task-level* and *operation-level*. Authorizations on operation-level include everything that can be handled by the *Authorization* function. In contrast, a task may be a set of multiple operations. Authorizations on this level are handled by the function *ClaimTI*. For example, workflow-based (history-based) separation of duty is handled on this level.

- In theory it would have been also possible to define the history (in *WFCoreEnv*) through

  $$History : \mathrm{seq}(\mathit{TASKINSTANCE} \times \mathit{USER})$$

  instead of

  $$History : \mathrm{seq}(\mathit{WFINSTANCE} \times \mathit{USER} \times \mathit{TASK})$$

  The reason is that from a task instance one can conclude the associated workflow instance and the associated task via $\mathit{instanceof}_{task}$ and $\mathit{belongsto}_{inst}$, respectively. However, from an implementation point of view we think the history should be organized as multiple histories each of which is associated with the appropriate workflow instances. Hence, each workflow instance carries its own history. Therefore, on the data modeling level, it is reasonable to organize the data accordingly.

# 5 Definition of the Policy Representation Format

The concrete syntax for policies is specified in the OPL/XML language. This section introduces the OPL/XML language and explains how to use it in order to compose ORKA policies. In contrast to the authorization model, the OPL/XML language only defines a subset of the Object Z definitions (cf. Section 4). For each policy module the OPL/XML specification only contains the elements of the corresponding Object Z entity class ending with the word "Policy". These entities define which part of the policy need to be made persistent within the OPL/XML policy specification. Persistent policy descriptions may be used, e.g to reinitialize an authorization platform with a given policy, transfer a policy to another system or backup and restore policy snapshots. The OPL/XML specification does not contain any endogenous or exogenous context definition itself[3]. However, context information may be referenced by means of external identifiers (see below), e.g., in order to compose constraints.

Some of the OPL/XML policy modules below may not be used isolated, since they make use of definitions from other modules within the same policy class (e.g. the role hierarchy model does not introduce new roles but only relations between existing roles, that have to be defined in an RBAC core module). The dependencies between the various modules have been outlined in Figure 1. There are policy modules that do not add any data to the policy specification, e.g. the RBACStandard module. In order to use such modules they need to be defined as active modules (cf. Section 5.2.

The authorization model constitutes a particular PDP being responsible for all policy decisions with respect to a single policy object. From that we derived the design principle, that policy objects must be equipped with all the necessary policy information in order to make the associated policy decisions. However, this also implies that different policy objects cannot share policy module instances (i.e. the hierarchical RBAC module of policy object A cannot reuse the role definitions of the RBAC core module of policy object B).

The design of OPL/XML is based on the following assumptions. First of all, we separate syntax from semantic. The OPL/XML definitions within this chapter specify the syntax of the policy, while the semantic has already been specified in Section 4. An ORKA policy object is one single XML file containing all the necessary module definitions. The DTD (document type definition) for the policy object constitutes the syntax of the policy, not the semantic. The overall syntax definition has been separated into several hierarchically structured DTD files, one DTD file per policy module and one DTD for the policy object, where the latter one inherits all modules' DTDs.

[3] With two exceptions in the Chinese Wall and Object-based Separation of Duty module, see Section 6.6.

Since the DTD specification language is somehow limited, there may be policy specifications that conform to the DTD but constitute invalid policies with respect to the authorization model. That means the DTD definitions provide a framework for definition of syntactically correct policies that not necessarily need to be reasonable policies with respect to the authorization model. Additionally, the DTD for policy object does not restrict the combination of policy modules within, however not all module combinations are meaningful.

The XML language allows to specify unique identifiers and references to those identifiers. Though, we have decided not to make use of them for the following reason. Since all policy modules are within the same policy object's XML file, the specifications of users, roles, permissions and all other identifiers would have to be unique not only within a policy module but within the whole policy object when using that feature. This would have been to restrictive.

Below we differentiate between *internal* and *external* identifiers. The scope of internal identifiers is the authorization model, i.e. they are assigned in order to be references within the policy specification (e.g. user or permission identifier). External identifiers are used to reference external resources such as operations or objects.

## 5.1 Naming and Namespace Conventions

The XML language is string-based and not typed. Therefore and for the sake of consistency we propose to keep with the following naming and namespace conventions.

In general, even if XML supports UTF-8 encoding, the OPL/XML filenames and specifications should only use the characters of the 7-bit ASCII charset.

### Filenames

All examples and fragments of policy modules within this manual start with the prefix "`module`" and end with "`policy-fragment`" followed by the suffix "`.xml`". The corresponding OPL/XML DTD files start with the prefix "`module`" and end with "`policy`" followed by the suffix "`.dtd`".

To name filenames the minus symbol is used as a separator for composite names, while to name the elements and attributes of the XML files and the corresponding DTD specifications the underscore symbol is used.

**Examples:**

- module-exo-context-policy.dtd
- module-role-hierarchy-policy-fragment.xml
- policy-object.dtd
- policy-object-domain-a.xml

### Element Names

All OPL/XML element names use lower case letters. Composite names use the underscore symbol "_" as a separator.

**Examples:**

- users, user, roles, role
- permission_assignment
- policy_object_modules

### External and Internal Identifiers

Since XML does not support typed element values, we use prefixes in order to specify the type of the particular internal identifier. This is intended to facilitate the readability of the OPL/XML statement. When possible, external identifiers should use the same naming scheme.

**Examples:**

- User “user:jochen_mueller”
- Permission “permission:read_some_file”
- Context constraint “cc:ssl_status_on”

The following sections introduce the OPL/XML syntax description for all the modules of the authorization model (cf. Section 4). We will explain, which parts of the particular module specification have to be specified within the OPL/XML representation of the policy an which not. Within each subsection we outline the syntax by means of a figure containing the main elements of the module or policy object. After that, we present an example fragment of the particular OPL/XML specification (displayed with a yellow background) and the corresponding DTD (displayed with a cyan background).

## 5.2 ORKA Policy Object in OPL/XML

The policy object in OPL/XML is the container for all policy rules of a particular application domain. There may be different policy objects for the various application domains. Each policy object is specified within a single OPL/XML file. Each policy object needs to contain all policy rules in order to make an access decision for the respective policy domain (i.e. a policy decision point cannot bring the policy rules of different policy domains together). An OPL/XML file cannot reference the internal identifiers specified within another OPL/XML file.

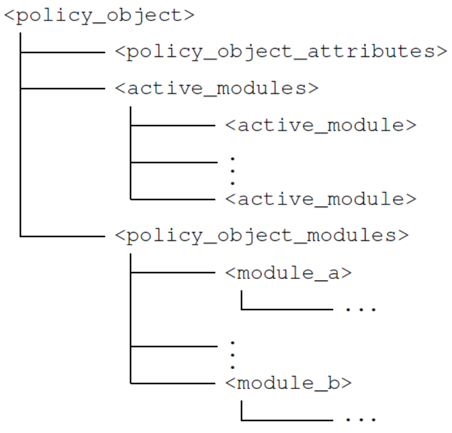


Figure 3: Outline of a Policy Object

Figure 3 outlines the main structure of the policy object in OPL/XML. The particular module definitions are omitted for the sake of simplicity. See Section 7.3 for a full policy object example. Some further aspects are pointed out below.

- The element `<policy_object>` includes a generic list of attributes, active policy modules and the actual policy modules.

- The element `<policy_object_attributes>` provides a generic attribute list for policy object related attributes, such as the name, the version, the description, or the creator of the policy object.

- The element `<active_modules>`, which specifies the active policy modules that are used within the policy object, is necessary besides the subsequent `<policy_object_modules>` element. Policy modules may be used even if they do not any persistent data to the policy. For example, the module RBACStandard is used for non-hierarchical role-based policies. However, the RBACStandard module itself does not add any policy data within the XML. All necessary data has to be stored using the RBACCore module. Nevertheless, it needs to be configured that the RBACStandard module is part of the policy object. That's why we need the list of active modules. Additionally, think of a second module without any additional policy data, e.g. RBACStandard2. If we would not specify which modules are active, we could not decide whether to use RBACStandard or RBACStandard 2 (or none of both).

- The element `<policy_object_modules>` is a container element for the particular policy module specifications. There must be at least one policy module within a policy object.

The following XML listing is an example for a policy object in OPL/XML. However, the specification of the policy object's modules is omitted here.

Listing 1: OPL/XML: Example Policy Object Fragment

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policy_object SYSTEM "policy-object.dtd">
<policy_object>

  <policy_object_attributes>
    <attribute key="name" value="po:domain-a"/>
    <attribute key="version" value="1.2"/>
    <attribute key="date" value="2008-03-05"/>
    <attribute key="description" value="XML␣syntax␣of␣a␣policy␣object"/>
    <attribute key="creator" value="user:admin-abc"/>
    <attribute key="domain" value="domain:a"/>
  </policy_object_attributes>

  <active_modules>
    <active_module name="module_rbac_core_policy" />
    <active_module name="module_role_hierarchy_policy" />
  </active_modules>

  <policy_object_modules>
    <module_rbac_core_policy>
      <!-- The particular module specification -->
    </module_rbac_core_policy>

    <module_role_hierarchy_policy>
      <!-- The particular module specification -->
    </module_role_hierarchy_policy>

  </policy_object_modules>

</policy_object>
```

The DTD `policy-object.dtd` specifies the syntax of the ORKA policy object. It inherits all the modules' DTD specifications, that will be detailed in the subsequent sections. By importing those DTDs the corresponding element and attribute definitions are available in the DTD of the policy object.

Listing 2: OPL/XML: policy-object.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- policy_object -->
<!ELEMENT policy_object (policy_object_attributes, active_modules,
     policy_object_modules)>

<!-- generic policy object attribute list -->
<!ELEMENT policy_object_attributes (attribute*)>
<!ELEMENT attribute EMPTY>
<!ATTLIST attribute
  key CDATA #REQUIRED
  value CDATA #REQUIRED
>

<!-- list of active modules -->
<!ELEMENT active_modules (active_module*)>
<!ELEMENT active_module EMPTY>
<!ATTLIST active_module
  name CDATA #REQUIRED
>

<!-- policy object modules -->
<!ELEMENT policy_object_modules (module_rbac_core_policy |
                                 module_role_hierarchy_policy |
                                 module_sep_duty_policy |
                                 module_sep_duty_rh_policy |
                                 module_exo_context_policy |
                                 module_chinese_wall_policy |
                                 module_obj_sep_duty_policy |
                                 module_wf_core_policy |
                                 module_wf_sep_duty_policy |
                                 module_wf_sep_duty_cc_policy|
                                 module_wf_cardinality_policy |
                                 module_wf_bind_duty_policy |
                                 module_wf_prereq_step_policy
                                 )+>

<!-- include external dtd for rbac core -->
<!ENTITY % module_rbac_core_policy SYSTEM "module-rbac-core-policy.dtd">
%module_rbac_core_policy;

<!-- include external dtd for role hierarchies -->
<!ENTITY % module_role_hierarchy_policy SYSTEM "module-role-hierarchy-policy.dtd">
%module_role_hierarchy_policy;

<!-- include the external dtd for non-hierachical separation of duty -->
<!ENTITY % module_sep_duty_policy SYSTEM "module-sep-duty-policy.dtd">
%module_sep_duty_policy;

<!-- include the external dtd for hierachical separation of duty (and
     reuse the non-hierarchical element definitions) -->
<!ENTITY % module_sep_rh_duty_policy SYSTEM "module-sep-duty-rh-policy.dtd">
```

```
%module_sep_rh_duty_policy;

<!-- include external dtd for exogenous context constraints -->
<!ENTITY % module_exo_context_policy SYSTEM "module-exo-context-policy.dtd">
%module_exo_context_policy;

<!-- include external dtd for chinese wall constraints -->
<!ENTITY % module_chinese_wall_policy SYSTEM "module-chinese-wall-policy.dtd">
%module_chinese_wall_policy;

<!-- include external dtd for object-based separation of duty constraints -->
<!ENTITY % module_obj_sep_duty_policy SYSTEM "module-obj-sep-duty-policy.dtd">
%module_obj_sep_duty_policy;

<!-- include external dtd for workflow core -->
<!ENTITY % module_wf_core_policy SYSTEM "module-wf-core-policy.dtd">
%module_wf_core_policy;

<!-- include external dtd for workflow-based separation of duty constraints -->
<!ENTITY % module_wf_sep_duty_policy SYSTEM "module-wf-sep-duty-policy.dtd">
%module_wf_sep_duty_policy;

<!-- include external dtd for workflow-based separation of duty with
     context constraints -->
<!ENTITY % module_wf_sep_duty_cc_policy SYSTEM "module-wf-sep-duty-cc-policy.dtd">
%module_wf_sep_duty_cc_policy;

<!-- include external dtd for workflow-based cardinality constraints -->
<!ENTITY % module_wf_cardinality_policy SYSTEM "module-wf-cardinality-policy.dtd">
%module_wf_cardinality_policy;

<!-- include external dtd for workflow-based bind of duty constraints -->
<!ENTITY % module_wf_bind_duty_policy SYSTEM "module-wf-bind-duty-policy.dtd">
%module_wf_bind_duty_policy;

<!-- include external dtd for workflow-based prerequisite step constraints -->
<!ENTITY % module_wf_prereq_step_policy SYSTEM "module-wf-prereq-step-policy.dtd">
%module_wf_prereq_step_policy;
```

# 6 Library of Modules: XML Syntax

## 6.1 RBAC Core Module in OPL/XML

The OPL/XML specifications within this section correspond to the Object Z definitions in Section 4.1. This section defines all the sets and relations of the policy module *RBACCorePolicy*, i.e. the users, the roles, the permissions, the user assignments, and the permission assignments. Figure 4 outlines the structure of this module.

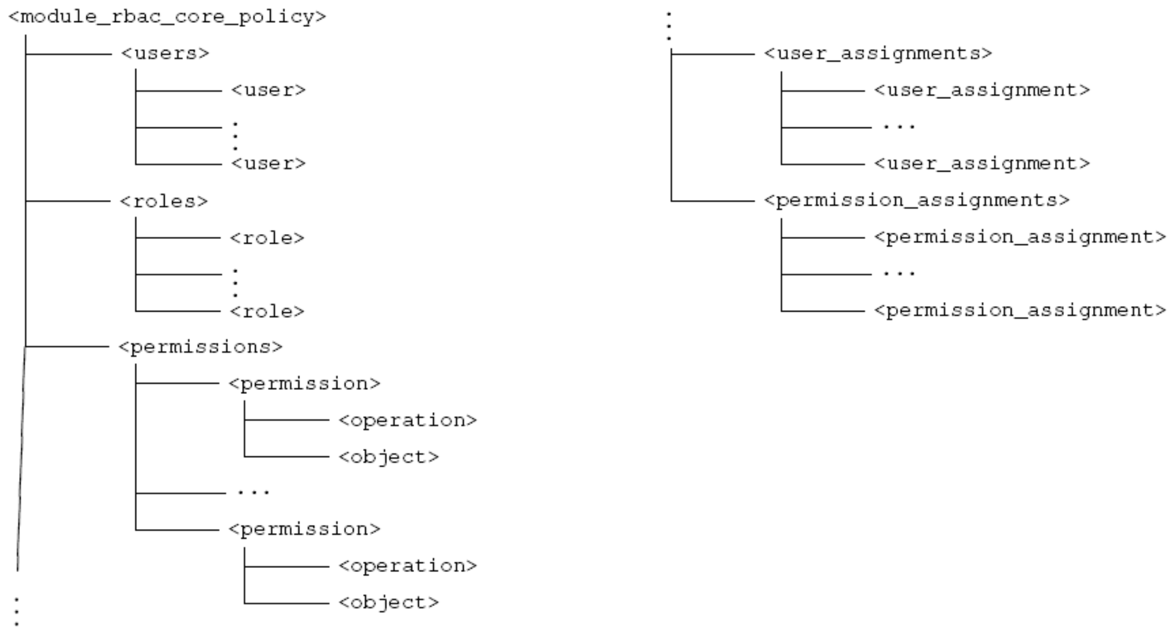


Figure 4: Outline of RBAC Core Module

All users, roles, and permissions are labeled by identifiers. The users are identified by an *external* identifier, that may be, for example, a distinguished name in an LDAP repository or a login name for a web server. Roles and permissions have an *internal* authorization model based identifier. This identifier is used in subsequent definitions of the policy, such as role hierarchies. Roles may have an additional description. Permissions consist of an operation (specified by an *external* operation identifier) and an object (also specified by an *external* object identifier).

Listing 3: OPL/XML: Example RBAC Core Module Fragment

```
<module_rbac_core_policy>
  <users>
    <!-- The element users (plural)as a container for all users is mandatory,
         however, the number of user elements (singular) within may
         be 0 to n. -->
    <user user_id="user:klaus_meier"/>
    <user user_id="user:jochen_schmidt"/>
  </users>
  <roles>
    <!-- The element roles (plural) as a container for all roles is mandatory,
         however, the number of role elements (singular) within may
         be 0 to n. -->
    <role role_id="role:employee" role_description="Employees"/>
    <role role_id="role:manager" role_description="Managers"/>
  </roles>
  <permissions>
    <!-- The element permissions (plural) as a container for all permissions
         is mandatory, however, the number of permissions elements (singular)
         within may be 0 to n. Operations and object elements are mandatory
         any permission -->
    <permission permission_id="permission:read_some_file">
      <operation operation_id="read"/>
      <object object_id="C:\SomeFile.txt"/>
    </permission>
    <permission permission_id="permission:read_confidential_file">
      <operation operation_id="read"/>
      <object object_id="C:\Strategy\Secrets.txt"/>
    </permission>
  </permissions>
  <user_assignments>
    <!-- The element user_assigments (plural) as a container for all user
         assignments is mandatory, however, the number of user_assignment
         elements (singular) within may be 0 to n. -->
    <user_assignment user_id="user:jochen_schmidt" role_id="role:employee"/>
    <user_assignment user_id="user:jochen_schmidt" role_id="role:manager"/>
    <user_assignment user_id="user:klaus_meier" role_id="role:employee"/>
  </user_assignments>
  <permission_assignments>
    <!-- The element permissions_assignments (plural) as a container for all
         permission assignments is mandatory, however, the number of
         permission_assignment elements (singular) within may be 0 to n. -->
    <permission_assignment permission_id="permission:read_some_file"
                           role_id="role:employee"/>
    <permission_assignment permission_id="permission:read_confidential_file"
                           role_id="role:manager"/>
  </permission_assignments>
</module_rbac_core_policy>
```

The DTD `module-rbac-core-policy.dtd` below defines the syntax of the RBAC core module. There is one aspect that holds for nearly all cases in that a multitude of similar elements are defined one after each other. Here, we usually have a container element which is named after the name of the elements to hold using the plural form (e.g. we have a required element `<users>` which contains zero or more elements `<user>`).

Listing 4: OPL/XML: DTD for RBAC Core Module

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_rbac_core_policy (users, roles, permissions, user_assignments,
     permission_assignments)>
<!-- Users -->
<!ELEMENT users (user)*>
<!ELEMENT user EMPTY>
<!ATTLIST user
        user_id CDATA #REQUIRED
>
<!-- Roles -->
<!ELEMENT roles (role)*>
<!ELEMENT role EMPTY>
<!ATTLIST role
        role_id CDATA #REQUIRED
        role_description CDATA #IMPLIED
>
<!-- Permissions -->
<!ELEMENT permissions (permission)*>
<!ELEMENT permission (operation, object)>
<!ATTLIST permission
        permission_id CDATA #REQUIRED
>
<!ELEMENT operation EMPTY>
```

```
<!ATTLIST operation
        operation_id CDATA #REQUIRED
 >
<!ELEMENT object EMPTY>
<!ATTLIST object
        object_id CDATA #REQUIRED
 >
<!-- User Assignments -->
<!ELEMENT user_assignments (user_assignment)*>
<!ELEMENT user_assignment EMPTY>
<!ATTLIST user_assignment
        role_id CDATA #REQUIRED
        user_id CDATA #REQUIRED
>
<!-- Permission Assignments -->
<!ELEMENT permission_assignments (permission_assignment)*>
<!ELEMENT permission_assignment EMPTY>
<!ATTLIST permission_assignment
        permission_id CDATA #REQUIRED
        role_id CDATA #REQUIRED
>
```

Please note that there is no XML specification for the module RBACStandard since it does not have any data to be stored persistently. However, if the module RBACStandard is used, it needs to be configured in the `<active_modules>` element of the policy object (cf. Section 5.2).

## 6.2 Role Hierarchy Module in OPL/XML

The OPL/XML specifications within this section are derived from the Object Z definitions in Section 4.2. This section defines all the role-role relations of the policy module *RoleHierarchy-Policy*. Figure 5 outlines the structure of this module.

The role hierarchy module cannot be used isolated, since it makes use of the role definitions of the rbac core modules. I.e. the policy class which uses the role hierarchy module must also contain an rbac core module.

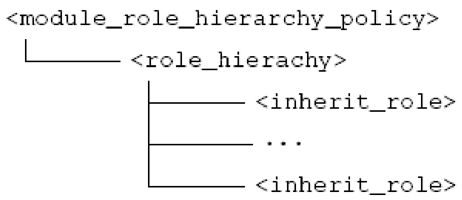


Figure 5: Outline of Role Hierarchy Module

The role hierarchy is specified as a set of tuples, where each tuple constitutes an edge in a role hierarchy graph. Each edge is a tuple of two roles, an upper and a lower role, where the upper role inherits all the permissions of the lower role. The role identifiers have to be previously defined in an rbac core module. In OPL/XML, the `<role_hierarchy>` contains `<inherit_role>` elements that represent such edges. Each `<inherit_role>` element has two mandatory attributes `upper_role` and `lower_role` that hold the internal role identifiers of the roles to be connected. The semantic of a role hierarchy requires that the role hierarchy is a partially ordered set, i.e. it is reflexive, antisymmetric, and transitive. In particular, it does not contains any cycles. The syntax definition must reflect these requirements, of course.

Listing 5: OPL/XML: Example Role Hierarchy Module Fragment

```
<module_role_hierarchy_policy>
  <!-- Specifies all edges between roles of the role hierarchy assumes that
       all roles have been previously defined in the RBACCoreM module -->
  <role_hierarchy>
    <!-- The upper_role inherits all permissions from the lower_role -->
    <inherit_role upper_role="role:a" lower_role="role:b"/>
    <inherit_role upper_role="role:a" lower_role="role:c"/>
    <inherit_role upper_role="role:b" lower_role="role:d"/>
    <inherit_role upper_role="role:e" lower_role="role:f"/>
  </role_hierarchy>
</module_role_hierarchy_policy>
```

The DTD `module-role-hierarchy-policy.dtd` below defines the OPL/XML syntax of the role hierarchy module.

Listing 6: OPL/XML: module-role-hierarchy-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_role_hierarchy_policy (role_hierarchy)>
<!ELEMENT role_hierarchy (inherit_role)*>
<!ELEMENT inherit_role EMPTY>
<!ATTLIST inherit_role
        lower_role CDATA #REQUIRED
        upper_role CDATA #REQUIRED
>
```

### 6.3 Separation of Duty Module in OPL/XML

The OPL/XML specifications within this section correspond to the Object Z definitions in Section 4.4. There are two different separation of duty modules in our authorization model, one for non-hierarchical role-based policy and one for hierarchical role-based policies (see Section 6.4). This section refers to the formal specifications of *SepDutyPolicy*, which supports the following types of separation of duty constraints

1. static separation of duty (SSoD),
2. strict static separation of duty (SSSoD),
3. static separation of duty attached to permissions (SSoDP), and
4. dynamic separation of duty (DSoD).

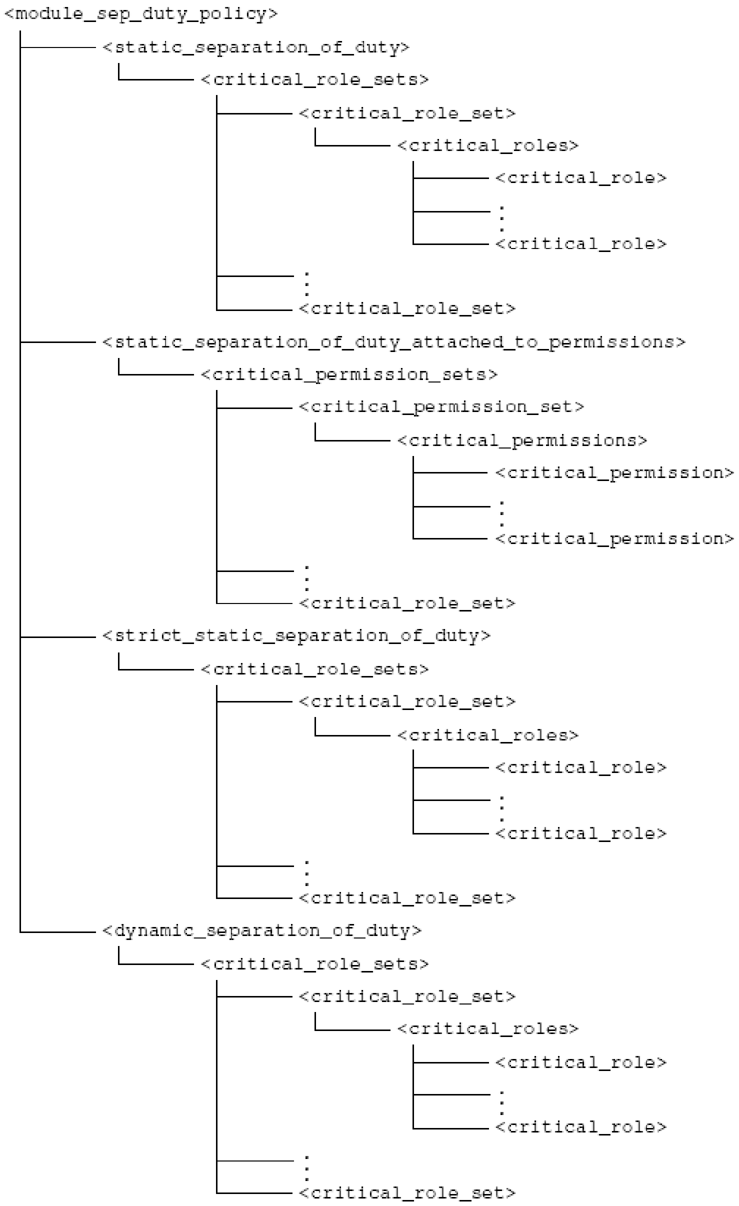


Figure 6: Outline of Separation of Duty Module

All types of separation of duty constraints are specified in the same module, as outlined by Figure 6. For the types SSoD, SSSoD and DSoD, we have an element `<critical_role_sets>` which holds all the `<critical_role_set>` elements of the particular type.

A `<critical_role_set>` element contains the element `<critical_roles>` which includes the critical roles as `<critical_role>` elements. Additionally, the `<critical_ role_set>` has a mandatory attribute `cardinality` which specifies the cardinality of the respective critical role set.

For the SSoDP constraint, we have

- `<critical_permission_sets>` instead of `<critical_role_sets>`,
- `<critical_permission_set>` instead of `<critical_role_set>`,
- `<critical_permissions>` instead of `<critical_roles>`, and
- `<critical_permission>` instead of `<critical_role>`.

Listing 7: OPL/XML: Example Separation of Duty Module Fragment

```xml
<module_sep_duty_policy>
  <static_separation_of_duty>
    <critical_role_sets>
      <!-- The element critical_role_sets (plural) as a container for all critical
           role sets is mandatory, however, the number of critical_role_set
           elements (singular) within may be 0 to n. -->
      <critical_role_set cardinality="1">
        <!-- The value of the attribute cardinality must be less than or equal to
             the number of roles -->
        <critical_roles>
          <!-- The number of critical roles may be 1 to n -->
          <critical_role role_id="role:b"/>
          <critical_role role_id="role:e"/>
        </critical_roles>
      </critical_role_set>
      <critical_role_set cardinality="2">
        <!-- The value of the attribute cardinality must be less than or equal to
             the number of roles -->
        <critical_roles>
          <!-- The number of critical roles may be 1 to n -->
          <critical_role role_id="role:a"/>
          <critical_role role_id="role:d"/>
          <critical_role role_id="role:e"/>
          <critical_role role_id="role:g"/>
        </critical_roles>
      </critical_role_set>
    </critical_role_sets>
  </static_separation_of_duty>
  <static_separation_of_duty_attached_to_permissions>
    <critical_permission_sets>
      <!-- The element critical_permission_sets (plural) as a container for all
           critical permission sets is mandatory, however, the number of
           critical_permission_set elements (singular) within may be 0 to n. -->
      <critical_permission_set cardinality="2">
        <critical_permissions>
          <critical_permission permission_id="permission:read_folder_a"/>
          <critical_permission permission_id="permission:read_folder_b"/>
          <critical_permission permission_id="permission:read_folder_c"/>
          <critical_permission permission_id="permission:read_folder_d"/>
        </critical_permissions>
      </critical_permission_set>
    </critical_permission_sets>
  </static_separation_of_duty_attached_to_permissions>
  <dynamic_separation_of_duty>
    <critical_role_sets>
      <!-- The element critical_role_sets (plural) as a container for all critical
           role sets is mandatory, however, the number of critical_role_set
           elements (singular) within may be 0 to n. -->
      <critical_role_set cardinality="2">
        <!-- The value of the attribute cardinality must be less than or equal to
             the number of roles -->
        <critical_roles>
          <!-- The number of critical roles may be 1 to n -->
          <critical_role role_id="role:a"/>
          <critical_role role_id="role:b"/>
          <critical_role role_id="role:h"/>
        </critical_roles>
      </critical_role_set>
    </critical_role_sets>
  </dynamic_separation_of_duty>
</module_sep_duty_policy>
```

The DTD `module-sep-duty-policy.dtd` below defines the OPL/XML syntax of the non-hierarchical separation of duty module.

Listing 8: OPL/XML: module-sep-duty-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Non-hierarchical policies may have four different types of separation of
     duties: static separation of duty (SSoD), strict static separation of duty
     (SSSoD), static separation of duty attached to permissions (SSoDP), and
     dynamic separation of duty (DSoD). -->
<!ELEMENT module_sep_duty_policy (static_separation_of_duty?,
    static_separation_of_duty_attached_to_permissions?,
     strict_static_separation_of_duty?, dynamic_separation_of_duty?)>

<!-- static and dynamic sods -->
<!ELEMENT static_separation_of_duty (critical_role_sets)>
<!ELEMENT static_separation_of_duty_attached_to_permissions (
    critical_permission_sets)>
<!ELEMENT strict_static_separation_of_duty (critical_role_sets)>
<!ELEMENT dynamic_separation_of_duty (critical_role_sets)>

<!-- critical role sets and cardinality -->
<!ELEMENT critical_role_sets (critical_role_set)*>
<!ELEMENT critical_role_set (critical_roles)>
<!ATTLIST critical_role_set
        cardinality CDATA #REQUIRED
        name CDATA #IMPLIED
        description CDATA #IMPLIED
>
<!ELEMENT critical_roles (critical_role)+>
<!ELEMENT critical_role EMPTY>
<!ATTLIST critical_role
        role_id CDATA #REQUIRED
>
<!-- critical permission sets and cardinality -->
<!ELEMENT critical_permission_sets (critical_permission_set)*>
<!ELEMENT critical_permission_set (critical_permissions)>
<!ATTLIST critical_permission_set
        cardinality CDATA #REQUIRED
        name CDATA #IMPLIED
        description CDATA #IMPLIED
>
<!ELEMENT critical_permissions (critical_permission)+>
<!ELEMENT critical_permission EMPTY>
<!ATTLIST critical_permission
        permission_id CDATA #REQUIRED
>
```

## 6.4 Separation of Duty in Role Hierarchies Module in OPL/XML

This section refers to the formal specifications of *SepDutyRHPolicy* (cf. Section 4.4) for hierarchical separation of duty constraints. In contrast to the non-hierarchical variant, only two types of separation of duty constraints are supported:

1. static separation of duty (SSoD) and
2. dynamic separation of duty (DSoD).

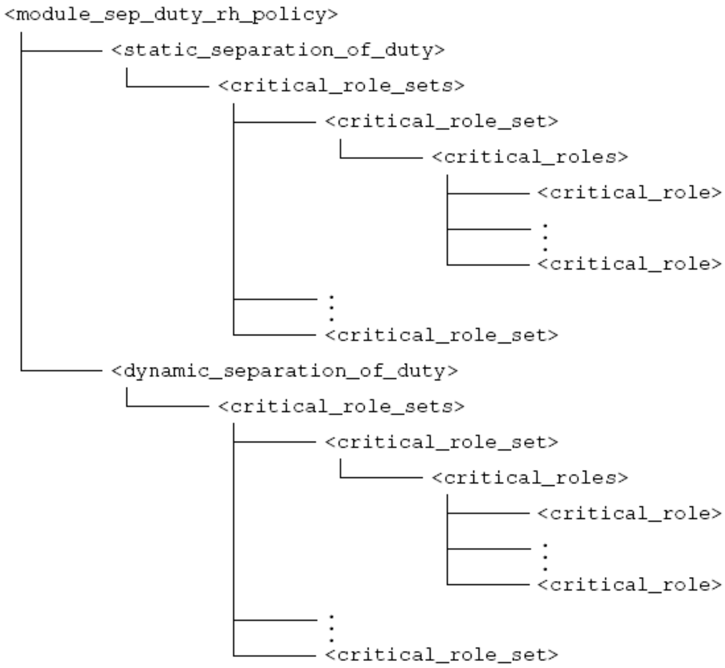


Figure 7: Outline of Hierarchical Separation of Duty Module

Since the syntax of the hierarchical separation of duty constraints modules is almost the same as the non-hierarchical variant, please refer to Section 6.3 for further information about the XML elements.

Listing 9: OPL/XML: Example Hierarchical Separation of Duty Fragment

```
<module_sep_duty_rh_policy>
  <static_separation_of_duty>
    <critical_role_sets>
      <!-- The element critical_role_sets (plural) as a container for all critical
           role sets is mandatory, however, the number of critical_role_set
           elements (singular) within may be 0 to n. -->
      <critical_role_set cardinality="1">
        <!-- The value of the attribute cardinality must be less than or equal to
             the number of roles -->
        <critical_roles>
          <!-- The number of critical roles may be 1 to n -->
          <critical_role role_id="role:a"/>
          <critical_role role_id="role:b"/>
        </critical_roles>
      </critical_role_set>
    </critical_role_sets>
  </static_separation_of_duty>
  <dynamic_separation_of_duty>
    <critical_role_sets>
      <!-- The element critical_role_sets (plural) as a container for all critical
           role sets is mandatory, however, the number of critical_role_set
           elements (singular) within may be 0 to n. -->
      <critical_role_set cardinality="2">
        <!-- The value of the attribute cardinality must be less than or equal to
             the number of roles -->
        <critical_roles>
          <!-- The number of critical roles may be 1 to n -->
          <critical_role role_id="role:c"/>
          <critical_role role_id="role:d"/>
          <critical_role role_id="role:e"/>
          <critical_role role_id="role:f"/>
        </critical_roles>
      </critical_role_set>
      <critical_role_set cardinality="1">
        <!-- The value of the attribute cardinality must be less than or equal to
             the number of roles -->
        <critical_roles>
          <!-- The number of critical roles may be 1 to n -->
          <critical_role role_id="role:a"/>
          <critical_role role_id="role:c"/>
        </critical_roles>
      </critical_role_set>
    </critical_role_sets>
  </dynamic_separation_of_duty>
</module_sep_duty_rh_policy>
```

The DTD module-sep-duty-rh-policy.dtd below defines the OPL/XML syntax of the hierarchical separation of duty module. As you can see, we reuse the document type definitions from the non-hierarchical variant i.e. we inherit the definitions and make them available in the hierarchical variant.

Listing 10: OPL/XML: module-sep-duty-rh-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Hierarchical policies may only have the two types of separation of duties:
     static separation of duty (SSoD), and dynamic separation of duty (DSoD). -->
<!ELEMENT module_sep_duty_rh_policy (static_separation_of_duty?,
     dynamic_separation_of_duty?)>
```

## 6.5 Exogenous Context Constraints Module in OPL/XML

This sections refers to the formal specification of exogenous context constraints as introduced in Section 4.3. Context constraints use external context provider to check information the policy decision is based on. Therefor, each context provider has defined various context functions that are referenced by its context function IDs. Optionally, context functions may require parameters to be provided by the caller.

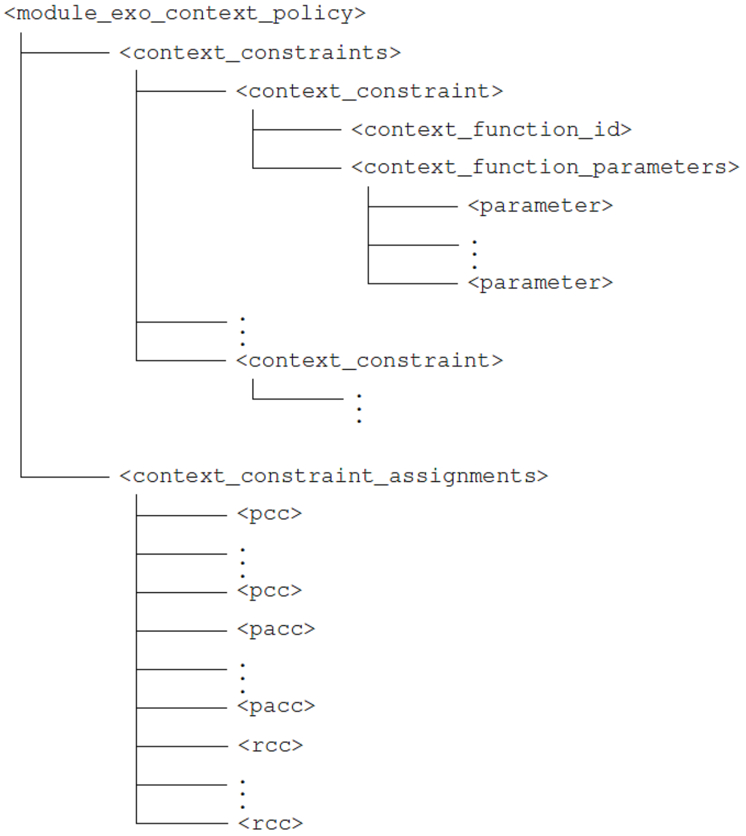


Figure 8: Outline of Exogenous Context Constraints Module

The OPL/XML specification of exogenous context constraints consists of two parts. First, all context constraints are defined and labeled within the element `<context_constraints>`, and second, the previously defined context constraints are assigned. There are currently three types of assignments supported: to permissions (`<pcc>`), to permission assignments (`<pacc>`) and to roles (`<rcc>`). Figure 8 outlines the main elements of the exogenous context module.

Listing 11: OPL/XML: Example Exogenous Context Constraints Fragment

```
<module_exo_context_policy>
  <!-- The module contains of two parts: definition of context constraints and
       assignment of permission to the previously defined context constraints -->

  <context_constraints>
    <!-- The element context_constraints (plural) as a container for all
         context_constraint elements is mandatory, however, the number of
         context_constraint elements (singular) within may be 0 to n. -->
    <context_constraint cc_id="cc:on_opening_hours_only">
      <context_function_id id="in_between_for_two_timestamps" />
      <context_function_parameters>
        <!-- a parameter must have a value and may have an optional key -->
        <parameter key="time" value="DateTimeContextProvider.current-time"
                   type="time" context="yes" />
        <parameter key="begin" value="08:00" type="time" context="no" />
        <parameter key="end" value="20:00" type="time" context="no" />
      </context_function_parameters>
    </context_constraint>
    <context_constraint cc_id="cc:ssl_status_on">
      <context_function_id id="equals" />
      <context_function_parameters>
        <parameter key="left" value="ConnectionContextProvider.check_ssl"
                   type="string" context="yes" />
        <parameter key="right" value="true" type="string" context="no" />
      </context_function_parameters>
    </context_constraint>
  </context_constraints>

  <context_constraint_assignments>
    <!-- The element context_constraint_assignments is a container
         for all context constraint assignments. There are currently three
         types of assignments supported: to permissions (pcc), to permission
         assignments (pacc) and to roles (rcc). -->
    <pcc permission_id="permission:write_accounting_entry"
         cc_id="cc:on_opening_hours_only" />
    <pcc permission_id="permission:read_confidential_data"
         cc_id="cc:ssl_status_on" />
    <pacc role_id="role:manager"
          permission_id="permission:update_confidential_data"
          cc_id="cc:ssl_status_on" />
    <rcc  role_id="role:clerk"
          cc_id="cc:on_opening_hours" />
  </context_constraint_assignments>

</module_exo_context_policy>
```

With respect to the OPL/XML representation, the following aspects are of interest:

- The mandatory attribute `cc_id` of the element `<context_constraint>` is an internal identifier and used to assign the context constraint to a permission.
- The mandatory attribute `id` of the element `<context_function_id>` is an external identifier representing a function that can process context information.
- The element `<context_function_parameters>` is optional. It contains a set of elements `<parameter>` representing the parameters passed to the context function.
- Each `<parameter>` element must have an attribute `value`, an attribute `type`, and an attribute `context`. It may have the optional attribute `key`.
  - **Value:** A parameter contains in its `value` attribute either a constant or a reference to a context information that must be retrieved from a context provider before it can be processed by the context function. In the latter case the notation is a follows:

    *`contextprovider.function(parameter, ...)`*

    Note that the specification of the appropriate external context provider that is responsible to resolve a reference to a context information is included in the `value` attribute.
    In particular, nesting within the `value` attribute is allowed. For example

    `wfms.getTasks(parameters.workflowinstance)`

  - **Type:** Independent from whether the content of the `value` attribute is a constant or a reference to external context information, the `type` attribute is one of the following: {`date|time|int|string`}. It specifies the data type of the constant or the data type of the resolved context information.
  - **Context:** The content of `context` is either `yes` or `no`:
    - In case of `yes`, the value of `value` needs to be resolved by an external context provider before it can be processed by the context function.
    - In case of `no`, the value of `value` is like a constant that can directly be processed by the context function.
  - **Key:** By allowing an optional `key` attribute, we can support different types of parameter handling:
    1. Assignment of parameters to the function in the provided order (such as with Java method invocations), see Listing 11 lines 21–27 which corresponds to the function call:
       `ConnectionContextProvider.check_ssl_status(on)`
    2. Assignment of parameter in arbitrary order, where the mapping of parameters is specified by keys (such as for most of Unix shell commands), see Listing 11 lines 12–20 which corresponds to the function call:
       `DateTimeContextProvider.time_interval_function`
       `--begin=08:00 --end=20:00`

- Within the element `<pcc>` an already existing permission (specified by the internal identifier `permission_id`) is assigned to the internal identifier `cc_id`.

- Within the element `<pacc>` an already existing permission assignment (specified by the two internal identifiers `permission_id` and `role_id`) is assigned to the internal identifier `cc_id`.

- Within the element `<rcc>` an already existing role (specified by the internal identifier `role_id`) is assigned to the internal identifier `cc_id`.

The DTD `module-exo-context-policy.dtd` below defines the OPL/XML syntax of the exogenous context constraints module.

Listing 12: OPL/XML: module-exo-context-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_exo_context_policy (context_constraints,
                                    context_constraint_assignments)>

<!-- context constraints -->
<!ELEMENT context_constraints (context_constraint)*>
<!ELEMENT context_constraint (context_function_id, context_function_parameters*)>
<!ATTLIST context_constraint
  cc_id CDATA #REQUIRED
>

<!ELEMENT context_function_id EMPTY>
<!ATTLIST context_function_id
  id CDATA #REQUIRED
>
<!ELEMENT context_function_parameters (parameter*)>
```

```
<!ELEMENT parameter EMPTY>
<!ATTLIST parameter
  value CDATA #REQUIRED
  type (date|time|int|string) #REQUIRED
  context (yes|no) #REQUIRED
  key CDATA #IMPLIED
>

<!-- context constraint assigments -->
<!ELEMENT context_constraint_assignments (pcc*,pacc*, rcc*)*>
<!ELEMENT pcc EMPTY>
<!ATTLIST pcc
  permission_id CDATA #REQUIRED
  cc_id CDATA #REQUIRED
>
<!ELEMENT pacc EMPTY>
<!ATTLIST pacc
  role_id CDATA #REQUIRED
  permission_id CDATA #REQUIRED
  cc_id CDATA #REQUIRED
>
<!ELEMENT rcc EMPTY>
<!ATTLIST rcc
  role_id CDATA #REQUIRED
  cc_id CDATA #REQUIRED
>
```

## 6.6 Chinese Wall Module in OPL/XML

This sections refers to the formal specification of the Chinese Wall model as introduced in Section 4.5. The following figure outlines the main elements of the Chinese Wall's OPL/XML representation format.

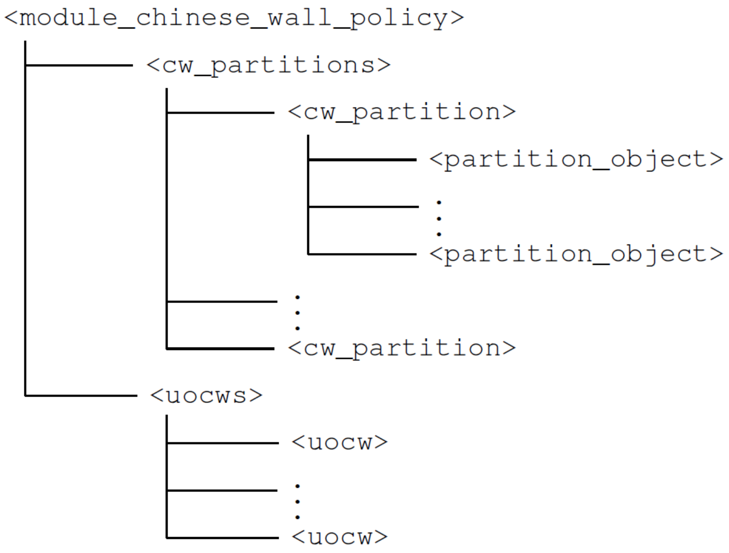


Figure 9: Outline of Chinese Wall Module

The module consists of two parts. First, the definition of object partitions. And second, the set of user object assignments. For the semantics of the particular elements, please refer to the formal specification in Section 4.5.

Listing 13: OPL/XML: Example Chinese Wall Fragment

```xml
<module_chinese_wall_policy>
<!-- This module consists of two parts. First, the definition of object
     partitions. And second, the set of user-object tuples specifying the history
     of objects by accessed by the particular users -->
  <cw_partitions>
    <cw_partition>
      <partition_object object_id="object:files_company_a" />
      <partition_object object_id="object:emails_company_a" />
    </cw_partition>
    <cw_partition>
      <partition_object object_id="object:files_company_b" />
      <partition_object object_id="object:emails_company_b" />
    </cw_partition>
  </cw_partitions>

  <!-- uocws are initially empty and are filled in while users working
       with the system. -->
  <uocws>
    <uocw user_id="user:mueller" object_id="object:files_company_a" />
    <uocw user_id="user:schmidt" object_id="object:emails_company_b" />
  </uocws>
</module_chinese_wall_policy>
```

The DTD `module-chinese-wall-policy.dtd` below defines the OPL/XML syntax of the chinese wall module.

Listing 14: OPL/XML: module-chinese-wall-policy.dtd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_chinese_wall_policy (cw_partitions, uocws)>

<!ELEMENT cw_partitions (cw_partition)*>

<!ELEMENT cw_partition (partition_object)*>

<!ELEMENT partition_object EMPTY>
<!ATTLIST partition_object
  object_id CDATA #REQUIRED
>

<!ELEMENT uocws (uocw)*>

<!ELEMENT uocw EMPTY>
<!ATTLIST uocw
  user_id CDATA #REQUIRED
  object_id CDATA #REQUIRED
>
```

## 6.7 Object-based Separation of Duty in OPL/XML

This module defines the OPL/XML representation of the ObjSepDuty module as formally specified in Section 4.5. The main OPL/XML elements of the module are outlined in the figure below.

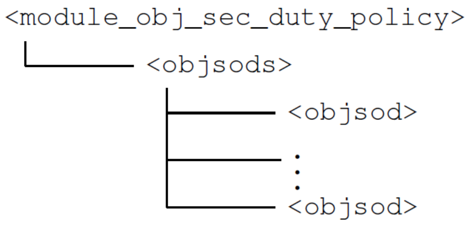


Figure 10: Outline of Object-based Separation of Duty Module

The following OPL/XML fragment outlines the use of the representation format.

Listing 15: OPL/XML: Object-based Separation of Duty Fragment

```
<module_obj_sep_duty_policy>
  <!-- Object-based Separation of Duty Constraints -->
  <objsods>
    <objsod object_id="object:product_bundle" />
    <objsod object_id="object:contract_a" />
  </objsods>
</module_obj_sep_duty_policy>
```

The DTD `module-obj-sep-duty-policy.dtd` below defines the OPL/XML syntax of the object-based separation of duty module.

Listing 16: OPL/XML: module-obj-sep-duty-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Object-based Separation of Duty Constraints -->
<!ELEMENT module_obj_sep_duty_policy (objsods)>
<!ELEMENT objsods (objsod)*>
<!ELEMENT objsod EMPTY>

<!ATTLIST objsod
  object_id CDATA #REQUIRED
>
```

The subsequent Sections 6.8 to 6.13 cover workflow-related OPL/XML modules. The formal specification and semantics has been defined in Section 4.6. There are currently five different workflow-related modules:

1. the workflow core module *WFCorePolicy* (see Section 6.8),
2. the workflow separation of duty constraints module *WFSepDutyPolicy* (see Section 6.9),

3. the workflow separation of duty constraints with context constraints module *WFSepDutyCCPolicy* (see Section 6.10),

4. the workflow cardinality constraints module *WFCardinalityPolicy* (see Section 6.11),

5. the workflow bind duty constraints module *WFBindDutyPolicy* (see Section 6.12), and

6. the workflow prerequisite step constraints module *WFPrereqStepPolicy* (see Section 6.13).

## 6.8 Workflow Core Module in OPL/XML

The OPL/XML specifications within this section correspond to the Object Z definitions in Section 4.6. The workflow core module builds the basic connection between the authorization system and the workflow engine. It specifies the set of task-permission assignments, i.e. assignments of workflow task IDs to permission IDs of the authorization system, and the set of task-role assignments, i.e. assignments of workflow task IDs to role IDs of the authorization system. Figure 11 outlines the main elements of the workflow core module.

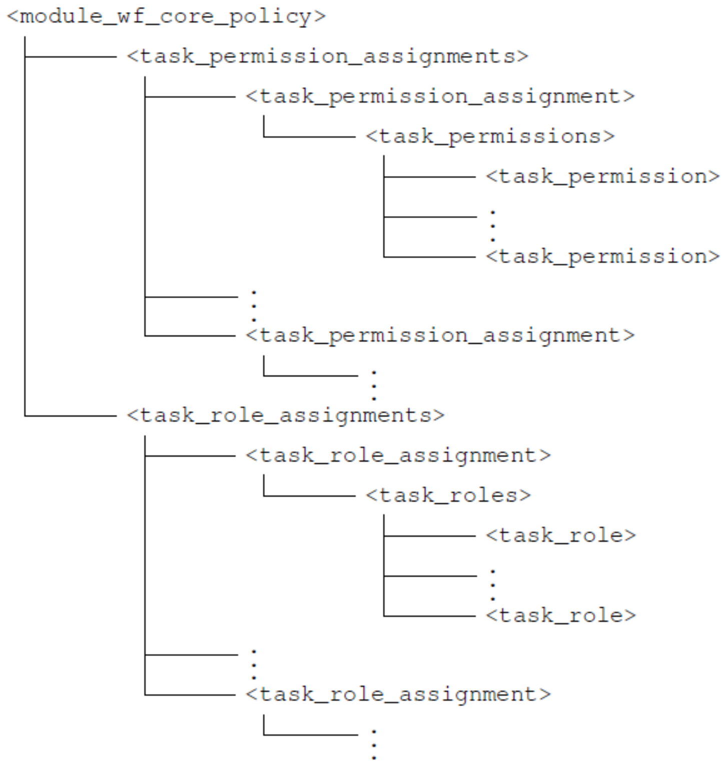


Figure 11: Outline of Workflow Core Module

The following aspects should be emphasized:

- The particular workflow management system does not need to be specified as part of the policy. Instead it is already known to the PDP as part of the environment.
- The element `<task_permission_assignments>` contains a set of `<task_permission_assignment>` elements.
- Each `<task_permission_assignment>` assigns a task to a permission, both specified within the attributes `task_id` and `permission_id` by its identifiers.

- The element <task_role_assignments> contains a set of <task_role_assignment> elements.

- Each <task_role_assignment> assigns a task to a role, both specified within the attributes task_id and role_id by its identifiers.

Listing 17: OPL/XML: Example Workflow Core Fragment

```
<module_wf_core_policy>
  <task_permission_assignments>
    <task_permission_assignment task_id="task:place_customer_order"
                                permission_id="permission:read_customer_record"/>
    <task_permission_assignment task_id="task:place_customer_order"
                                permission_id="permission:write_order_table"/>
    <task_permission_assignment task_id="task:check_credit_worthiness"
                                permission_id="permission:prepare_rating_report"/>
  </task_permission_assignments>
  <task_role_assignments>
    <task_role_assignment task_id="task:place_customer_order"
                          role_id="role:clerk"/>
    <task_role_assignment task_id="task:check_credit_worthiness"
                          role_id="role:manager"/>
  </task_role_assignments>
</module_wf_core_policy>
```

The DTD module-wf-core-policy.dtd of the workflow core module is displayed in the Listing 18 below.

Listing 18: OPL/XML: module-wf-core-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_wf_core_policy (task_permission_assignments,task_role_assignments)
>

<!ELEMENT task_permission_assignments (task_permission_assignment*)>
<!ELEMENT task_permission_assignment EMPTY>
<!ATTLIST task_permission_assignment
  task_id CDATA #REQUIRED
  permission_id CDATA #REQUIRED
>

<!ELEMENT task_role_assignments (task_role_assignment*)>
<!ELEMENT task_role_assignment EMPTY>
<!ATTLIST task_role_assignment
  task_id CDATA #REQUIRED
  role_id CDATA #REQUIRED
>
```

## 6.9 Workflow Separation of Duty Module in OPL/XML

This section introduces the workflow separation of duty constraints module. It allows to define three types of history based dynamic separation of duty constraints as formally specified in Section 4.6:

- HDSoDSL – History-based Dynamic Separation of Duty (Syncless)
- HDSoD – History-based Dynamic Separation of Duty (Simple)
- HDSoDTP – History-based Dynamic Separation of Duty (Task Partitions)

The main elements of this module are outlined in Figure 12.

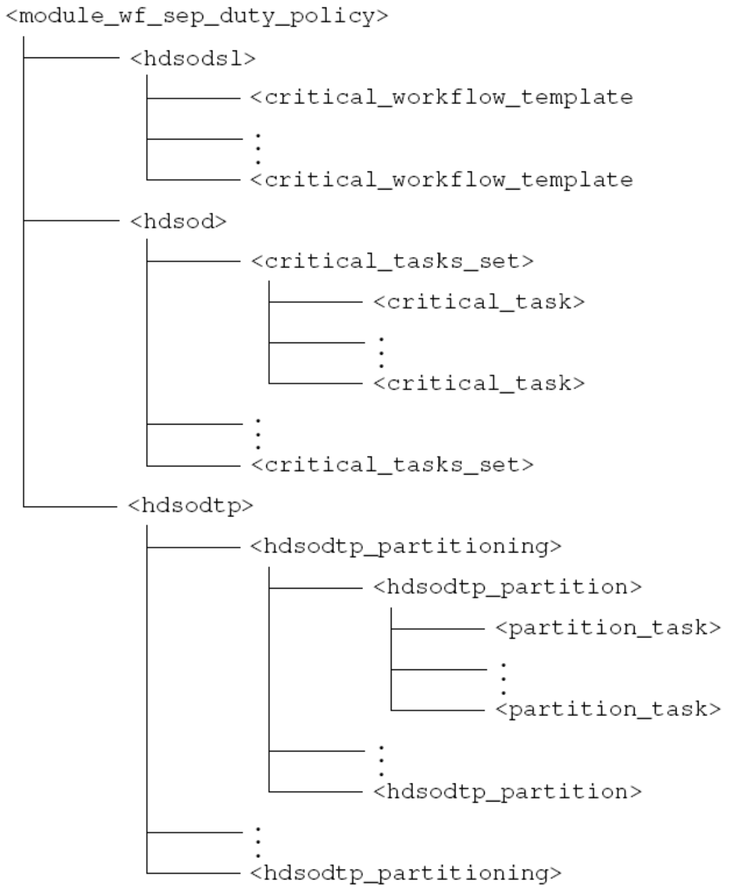


Figure 12: Outline of Workflow Separation of Duty Module

Within `<hdsodsl>`, the `<critical_workflow_template>` element contains the mutual excluding `<workflow_template>` elements. Each `<workflow_ template>` element has a mandatory attribute `template_id` that references external workflow template of the corresponding workflow management system.

For `<hdsod>`, the `<critical_tasks_set>` elements contain a mandatory attribute `cardinality`. The `<critical_tasks_set>` element contains a set of `<critical_task>` elements that are used to reference a task by means of the attribute `task_id` which is an external identifier.

Similar to the Chinese Wall module, `<hdsodtp>` defines mutual exclusive task partitions.

The following listing illustrates the use of the module.

Listing 19: OPL/XML: Example Workflow Separation of Duty Fragment

```
<module_wf_sep_duty_policy>
  <!-- History-based Dynamic Separation of Duty (Syncless) -->
  <hdsodsl>
      <critical_workflow_template template_id="wf_template:loan_origination" />
      <critical_workflow_template template_id="wf_template:windows_perm_change" />
      <critical_workflow_template template_id="wf_template:unix_add_user" />
      <critical_workflow_template template_id="wf_template:unix_perm_change" />
  </hdsodsl>

  <!-- History-based Dynamic Separation of Duty (Simple) -->
  <hdsod>
      <critical_tasks_set cardinality="2">
          <critical_task task_id="task:customer_ident"/>
          <critical_task task_id="task:check_rating"/>
          <critical_task task_id="task:open_account"/>
      </critical_tasks_set>
      <critical_tasks_set cardinality="1">
          <critical_task task_id="task:choose_boundled_product"/>
          <critical_task task_id="task:price_boundled_product"/>
      </critical_tasks_set>
  </hdsod>

  <!-- History-based Dynamic Separation of Duty (Task Partitions) -->
  <hdsodtp>
     <hdsodtp_partitioning>
        <hdsodtp_partition>
            <partition_task task_id="task:input_customer_data"/>
            <partition_task task_id="task:check_rating"/>
            <partition_task task_id="task:bank_signs_form"/>
        </hdsodtp_partition>
        <hdsodtp_partition>
             <partition_task task_id="task:print_opening_form"/>
             <partition_task task_id="task:open_account"/>
        </hdsodtp_partition>
     </hdsodtp_partitioning>
  </hdsodtp>

</module_wf_sep_duty_policy>
```

The DTD `module-wf-sep-duty-policy.dtd` of the workflow separation of duty OPL/XML module is shown in Listing 20 below.

Listing 20: OPL/XML: module-wf-sep-duty-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_wf_sep_duty_policy (hdsodsl?, hdsod?, hdsodtp?)>

<!-- History-based Dynamic Separation of Duty (Syncless) -->
<!ELEMENT hdsodsl (critical_workflow_template*)>
<!ELEMENT critical_workflow_template EMPTY>
<!ATTLIST critical_workflow_template
  template_id CDATA #REQUIRED
>

```

```
<!-- History-based Dynamic Separation of Duty (Simple) -->
<!ELEMENT hdsod (critical_tasks_set)*>
<!ELEMENT critical_tasks_set (critical_task)+>
<!ATTLIST critical_tasks_set
  cardinality CDATA #REQUIRED
  name CDATA #IMPLIED
  description CDATA #IMPLIED
>
<!ELEMENT critical_task EMPTY>
<!ATTLIST critical_task
  task_id CDATA #REQUIRED
>

<!-- History-based Dynamic Separation of Duty (Task Partitions) -->
<!ELEMENT hdsodtp (hdsodtp_partitioning)*>
<!ELEMENT hdsodtp_partitioning (hdsodtp_partition)*>
<!ATTLIST hdsodtp_partitioning
  name CDATA #IMPLIED
  description CDATA #IMPLIED
>
<!ELEMENT hdsodtp_partition (partition_task)+>
<!ATTLIST hdsodtp_partition
  name CDATA #IMPLIED
  description CDATA #IMPLIED
>
<!ELEMENT partition_task EMPTY>
<!ATTLIST partition_task
  task_id CDATA #REQUIRED
>
```

## 6.10 Workflow Separation of Duty with Context Constraints Module in OPL/XML

This section introduces the OPL/XML representation of the module WFSepDutyCCPolicy as formally specified in Section 4.6. The main elements of the module are outlined in the figure below.

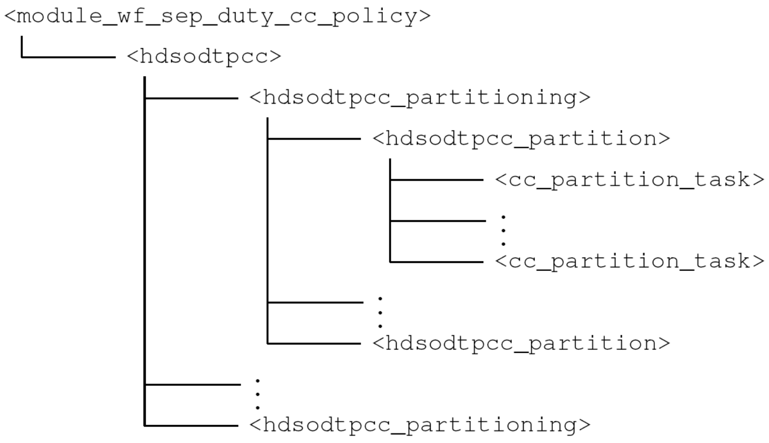


Figure 13: Outline of Workflow Separation of Duty with Context Constraints Module

The following listing illustrated the use of the module.

Listing 21: OPL/XML: Example Workflow Separation of Duty with Context Constraints Fragment

```
<module_wf_sep_duty_cc_policy>
  <!-- History-based Dynamic Separation of Duty with Task Partitions and
       Context Constraints -->
  <hdsodtpcc>
     <hdsodtpcc_partitioning cc_id="on_opening_hours_only">
        <hdsodtpcc_partition>
            <cc_partition_task task_id="task:input_customer_data"/>
            <cc_partition_task task_id="task:check_rating"/>
            <cc_partition_task task_id="task:bank_signs_form"/>
        </hdsodtpcc_partition>
        <hdsodtpcc_partition>
             <cc_partition_task task_id="task:print_opening_form"/>
             <cc_partition_task task_id="task:open_account"/>
        </hdsodtpcc_partition>
     </hdsodtpcc_partitioning>
  </hdsodtpcc>
</module_wf_sep_duty_cc_policy>
```

The DTD `module-wf-sep-duty-cc-policy.dtd` of the workflow separation of duty OPL/XML module is shown in Listing 22 below.

Listing 22: OPL/XML: module-wf-sep-duty-cc-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_wf_sep_duty_cc_policy (hdsodtpcc)>
<!-- History-based Dynamic Separation of Duty (Task Partitions) -->
<!ELEMENT hdsodtpcc (hdsodtpcc_partitioning*)>

<!ELEMENT hdsodtpcc_partitioning (hdsodtpcc_partition)*>
<!ATTLIST hdsodtpcc_partitioning
  cc_id CDATA #REQUIRED
  name CDATA #IMPLIED
  description CDATA #IMPLIED
>
<!ELEMENT hdsodtpcc_partition (cc_partition_task)+>
<!ATTLIST hdsodtpcc_partition
  name CDATA #IMPLIED
  description CDATA #IMPLIED
>
<!ELEMENT cc_partition_task EMPTY>
<!ATTLIST cc_partition_task
  task_id CDATA #REQUIRED
>
```

## 6.11 Workflow Cardinality Module in OPL/XML

This section allows the definition of workflow task cardinality constraints by assigning a cardinality to a workflow task. The OPL/XML specifications within this section correspond to the Object Z definitions in Section 4.6. The basic structure is outlined in Figure 14 below.

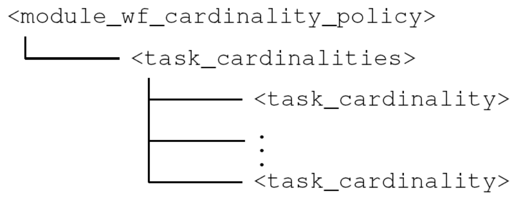


Figure 14: Outline of Workflow Cardinality Module

The module contains a set of `<task_cardinality>` elements. Each of them has two mandatory attributes `task_id` and `cardinality`.

Listing 23: OPL/XML: Example Workflow Cardinality Fragment

```
<module_wf_cardinality_policy>
  <task_cardinalities>
    <task_cardinality task_id="task:check_customer_identity" cardinality="2"/>
    <task_cardinality task_id="task:submit_order" cardinality="1"/>
  </task_cardinalities>
</module_wf_cardinality_policy>
```

The DTD `module-wf-cardinality-policy.dtd` of the workflow cardinality module is shown in Listing 24.

Listing 24: OPL/XML: module-wf-cardinality-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_wf_cardinality_policy (task_cardinalities)>
<!ELEMENT task_cardinalities (task_cardinality*)>
<!ELEMENT task_cardinality EMPTY>
<!ATTLIST task_cardinality
  task_id CDATA #REQUIRED
  cardinality CDATA #REQUIRED
>
```

## 6.12 Workflow Bind of Duty Module in OPL/XML

This section describes the OPL/XML syntax of the workflow bind of duty constraints module, which allows to bind a workflow task to another workflow task. The OPL/XML specifications within this section correspond to the Object Z definitions in Section 4.6. Figure 15 outlines the main OPL/XML elements of the module.

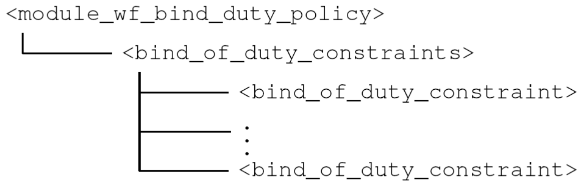


Figure 15: Outline of Workflow Bind of Duty Module

The `<bind_of_duty_constraints>` element contains a set of `<bind_of_duty_constraint>` elements, which have two mandatory attributes `task_id` and `bound_task_id` containing the external workflow task identifiers of the two tasks to be bound.

Listing 25: OPL/XML: Example Workflow Bind of Duty Fragment

```xml
<module_wf_bind_duty_policy>
  <bind_of_duty_constraints>
    <bind_of_duty_constraint task_id="task:input_customer_data"
                             bound_task_id="task:customer_identification"/>
    <bind_of_duty_constraint task_id="task:customer_identification"
                             bound_task_id="task:check_credit_worthiness"/>
    <bind_of_duty_constraint task_id="task:choose_bundled_product"
                             bound_task_id="task:price_bundled_product"/>
  </bind_of_duty_constraints>
</module_wf_bind_duty_policy>
```

The DTD `module-wf-bind-duty-policy.dtd` of the workflow bind of duty constraints module is shown in Listing 26.

Listing 26: OPL/XML: module-wf-bind-duty-policy.dtd

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_wf_bind_duty_policy (bind_of_duty_constraints)>
<!ELEMENT bind_of_duty_constraints (bind_of_duty_constraint*)>
<!ELEMENT bind_of_duty_constraint EMPTY>
<!ATTLIST bind_of_duty_constraint
  task_id CDATA #REQUIRED
  bound_task_id CDATA #REQUIRED
>
```

### 6.13 Workflow Prerequisite Step Module in OPL/XML

The last workflow-related module is used to define the prerequisite tasks of a given task. The OPL/XML specifications within this section correspond to the Object Z definitions in Section 4.6. Its main elements are outlined in Figure 16.

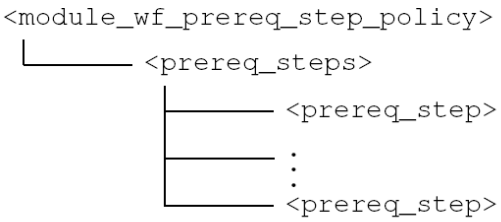


Figure 16: Outline of Workflow Prerequisite Step Module

The element `<prereq_steps>` contains a set of prerequisite constraints modeled by `<prereq_step>` elements. Each `<prereq_step>` element contains two mandatory attributes `prereq_task_id` and `task_id`, where prereq_task_id needs to be fulfilled before task_id may be executed.

Listing 27: OPL/XML: Example Workflow Prerequisite Step Fragment

```
<module_wf_prereq_step_policy>
  <prereq_steps>
    <prereq_step prereq_task_id="task:␣check_credit_worthiness"
                 task_id="task:check_rating"/>
    <prereq_step prereq_task_id="task:check_rating"
                 task_id="task:bank_signs_form"/>
  </prereq_steps>
</module_wf_prereq_step_policy>
```

The DTD `module-wf-prereq-step-policy.dtd` of the prerequisite step constraints module is shown in Listing 28 below.

Listing 28: OPL/XML: module-wf-prereq-step-policy.dtd

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT module_wf_prereq_step_policy (prereq_steps)>
<!ELEMENT prereq_steps (prereq_step*)>
<!ELEMENT prereq_step EMPTY>
<!ATTLIST prereq_step
  prereq_task_id CDATA #REQUIRED
  task_id CDATA #REQUIRED
>
```

# 7 A Banking Scenario

This section specifies a full policy example derived from the case study in Ap1.1. Section 7.1 defines the appropriate policy class as a combination of policy modules. While Section 7.2 states the policy by means of an ad hoc intuitive set syntax, Section 7.3 gives the policy in the defined XML syntax.

## 7.1 Policy Class

*BankingPolicyWFPolicy*

*SepDutyPolicy*
*WFSepDutyPolicy*
*WFSepDutyCCPolicy*
*ExoContextPolicy*

*BankingPolicyWFEngine*

*SepDutyEngine*
*WFSepDutyM*
*WFSepDutyCCM*
*ExoContextEngine*

## 7.2 Policy Example in OPL/Abstract

**Basic Assignments:** The relations *TRA* and *TPA* can be used to represent the assignment of roles, permissions, and workflow steps as described in Table 1 of Ap1.1. Users, roles, permissions, user assignment, and permission assignment are included appropriately.

```
# Users (U)
users
    user:klaus_meier,
    user:karla_meier,
    user:jochen_schmidt,
    user:armin_mueller,
    user:susanne_schaefer;

# Roles (R)
roles
    role:clerk_preprocessor,
    role:clerk_postprocessor,
    role:supervisor,
    role:customer,
    role:manager;

# Permissions (P)
permissions
    permission:query_customer_data   { query(),   CustomerData },
    permission:update_customer_data  { update(),  CustomerData },
    permission:prepare_ratingreport  { prepare(), RatingReport },
    permission:release_ratingreport  { release(), RatingReport },
    permission:post_ratingreport     { post(),    RatingReport },
    permission:query_ratingreport    { query(),   RatingReport },
    permission:update_ratingreport   { update(),  RatingReport },
    permission:query_productbundle   { query_avail_prod(), ProductBundle },
    permission:modify_productbundle  { modify(),  ProductBundle },
    permission:commit_productbundle  { commit(),  ProductBundle },
    permission:print_contract        { print(),   Contract },
    permission:sign_contract         { sign(),    Contract },
    permission:update_contract       { update(),  Contract },
    permission:open_account          { open(),    Account };

# User Assignment (UA)
```

```
user:jochen_schmidt   user-assigned-to  role:clerk_preprocessor;
user:karla_meier      user-assigned-to  role:clerk_postprocessor;
user:klaus_meier      user-assigned-to  role:supervisor;
user:armin_mueller    user-assigned-to  role:manager;
user:susanne_schaefer user-assigned-to  role:customer;

# Permission Assigment to Roles (PA) and Context Contraints (PACC)
permission:query_customer_data   permission-assigned-to  role:clerk_preprocessor;
permission:update_customer_data  permission-assigned-to  role:clerk_preprocessor;
permission:prepare_ratingreport  permission-assigned-to  role:clerk_postprocessor;
permission:release_ratingreport  permission-assigned-to  role:clerk_postprocessor
                  under cc:cc1;
permission:post_ratingreport     permission-assigned-to  role:clerk_postprocessor;
permission:release_ratingreport  permission-assigned-to  role:supervisor
                  under cc:cc2;
permission:query_ratingreport    permission-assigned-to  role:clerk_postprocessor;
permission:update_ratingreport   permission-assigned-to  role:supervisor;
permission:query_productbundle   permission-assigned-to  role:clerk_postprocessor;
permission:modify_productbundle  permission-assigned-to  role:clerk_postprocessor;
permission:commit_productbundle  permission-assigned-to  role:clerk_postprocessor
                  under cc:cc1;
permission:commit_productbundle  permission-assigned-to  role:supervisor
                  under cc:cc2;
permission:print_contract        permission-assigned-to  role:clerk_postprocessor;
permission:sign_contract         permission-assigned-to  role:customer;
permission:sign_contract         permission-assigned-to  role:manager;
permission:update_contract       permission-assigned-to  role:manager;
permission:open_account          permission-assigned-to  role:clerk_postprocessor;

# Context Constraints (CC)
cc:cc1  { creditbureau.mydomain.org , wfi-amount , (=<, 100k) } ;
cc:cc2  { creditbureau.mydomain.org , wfi-amount , (> , 100k) } ;

# Task Role Assignment (TRA)
task:1_input_customer_data  task-assigned-to-role  role:clerk_preprocessor;
task:2_customer_ident       task-assigned-to-role  role:clerk_preprocessor;
task:3a_check_cred_worthin  task-assigned-to-role  role:clerk_postprocessor;
task:3b_check_cred_worthin  task-assigned-to-role  role:supervisor;
task:3b_check_cred_worthin  task-assigned-to-role  role:clerk_postprocessor;
task:3c_check_cred_worthin  task-assigned-to-role  role:clerk_postprocessor;
task:4_check_rating         task-assigned-to-role  role:clerk_postprocessor;
task:5_bank_signs_form      task-assigned-to-role  role:supervisor;
task:6_choose_bundled_prod  task-assigned-to-role  role:clerk_postprocessor;
task:7a_price_bundled_prod  task-assigned-to-role  role:clerk_postprocessor;
task:7b_price_bundled_prod  task-assigned-to-role  role:supervisor;
task:7b_price_bundled_prod  task-assigned-to-role  role:clerk_postprocessor;
task:8_print_opeing_form    task-assigned-to-role  role:clerk_postprocessor;
task:9_customer_signs_form  task-assigned-to-role  role:customer;
task:10_bank_signs_form     task-assigned-to-role  role:manager;
task:11_open_account        task-assigned-to-role  role:clerk_postprocessor;

# Task Permission Assignment (TPA)
task:1_input_customer_data  task-assigned-to-perm  permission:query_customer_data;
task:1_input_customer_data  task-assigned-to-perm  permission:update_customer_data;
task:2_customer_ident       task-assigned-to-perm  permission:query_customer_data;
task:3a_check_cred_worthin  task-assigned-to-perm  permission:prepare_ratingreport;
task:3b_check_cred_worthin  task-assigned-to-perm  permission:release_ratingreport;
task:3c_check_cred_worthin  task-assigned-to-perm  permission:post_ratingreport;
task:4_check_rating         task-assigned-to-perm  permission:query_ratingreport;
task:5_bank_signs_form      task-assigned-to-perm  permission:update_ratingreport;
task:6_choose_bundled_prod  task-assigned-to-perm  permission:query_productbundle;
task:7a_price_bundled_prod  task-assigned-to-perm  permission:modify_productbundle;
```

```
task:7b_price_bundled_prod  task-assigned-to-perm  permission:commit_productbundle;
task:8_print_opeing_form    task-assigned-to-perm  permission:print_contract;
task:9_customer_signs_form  task-assigned-to-perm  permission:sign_contract;
task:10_bank_signs_form     task-assigned-to-perm  permission:sign_contract;
task:10_bank_signs_form     task-assigned-to-perm  permission:update_contract;
task:11_open_account        task-assigned-to-perm  permission:open_account;
```

**Requirement 1:** *No person may be assigned to the two exclusive roles pre/post processor.*

```
# Static Separation of Duty (SSoD)
ssod (1) { role:clerk_preprocessor , role:clerk_postprocessor };
```

**Requirement 2:** *A person may be assigned to the two exclusive roles but must not activate them both within one process. This means that either they are activated in any state or they have not been activated one after another.*

We think it is not efficient to introduce and track a role activation history for each workflow instance. Instead, we make use of the existing task history to ensure the same principle.

```
# Dynamic Separation of Duty (DSoD)
dsod (1) { role:clerk_preprocessor , role:clerk_postprocessor };

# History-based Dynamic Separation of Duty (HDSoDTP)
hdsodtp { { task:1_input_customer_data, task:2_customer_ident },
          { task:3a_check_cred_worthin, task:3b_check_cred_worthin,
            task:3c_check_cred_worthin, task:4_check_rating,
            task:6_choose_bundled_prod, task:7a_price_bundled_prod,
            task:7b_price_bundled_prod, task:8_print_opeing_form,
            task:11_open_account } } ;
```

**Requirement 3:** *If the customer is an industrial customer, the master data must be verified by an independent clerk.*

Since in the example there is no "verification of master data" workflow step, we interpret the "customer identification" step here as such. Hence, if the customer is an industrial customer, then the customer identification needs to be carried out by a person other than the person who performed "input customer data".

```
# Context Constraints (CC)
cc:cc3  { customerinformation.mydomain.org , obj-customer-type , (=, "industrial") } ;

# History-based Dynamic Separation of Duty with Context Constraints (HDSoDTPCC)
hdsodtpcc if cc:cc3 then { {task:1_input_customer_data} ,
                           {task:2_customer_ident} } ;
```

**Requirement 4:** *If the credit bureau rating is negative then the internal rating must be performed by another clerk.*

Similarly to Requirement 3, in case that the credit bureau rating is negative, a history-based separation of duty constraint is placed in such a way that the workflow step "check rating" is performed by a person other than the person who performed the remaining steps for the clerk post processor.

```
# Context Constraints (CC)
cc:cc4  { ratingserver.mydomain.org , internal-rating , (<, 0) } ;

# History-based Dynamic Separation of Duty with Context Constraints (HDSoDTPCC)
hdsodtpcc if cc:cc4 then
```

```
        { { task:4_check_rating } ,
          { task:3a_check_cred_worthin, task:3b_check_cred_worthin,
            task:3c_check_cred_worthin, task:6_choose_bundled_prod,
            task:7a_price_bundled_prod, task:7b_price_bundled_prod,
            task:8_print_opeing_form, task:11_open_account }
        };
```

**Requirement 5:** *If the internal rating is negative, then the case must be confirmed by a supervisor.*

This requirement can be realized in the same way as Requirement 4. Since the necessary workflow step "supervisor confirms" is missing in the template, we also skip writing the appropriate constraint explicitly.

**Requirement 6:** *A clerk may only price a bundled product if he did not perform the operation modify() wrt to the specific offer.*

```
# History-based Dynamic Separation of Duty (HDSoDTP)
hdsodtp { { task:7a_price_bundled_prod },
          { task:7b_price_bundled_prod }
        } ;
```

**Requirement 7:** *If the customer is an industrial customer, then a clerk may perform tasks 1 to 9 or 10 but not both for the same customer.*

```
# History-based Dynamic Separation of Duty with Context Constraints (HDSoDTPCC)
hdsodtpcc if cc:cc3 then
        { { task:1_input_customer_data, task:2_customer_ident,
            task:3a_check_cred_worthin, task:3b_check_cred_worthin,
            task:3c_check_cred_worthin, task:4_check_rating,
            task:5_bank_signs_form, task:6_choose_bundled_prod,
            task:7a_price_bundled_prod, task:7b_price_bundled_prod,
            task:8_print_opeing_form, task:9_customer_signs_form } ,
          { task:10_bank_signs_form }
        };
```

**Requirement 8:** *A principal may be a member of the two exclusive roles pre/post processor and the complete set of authorizations acquired over these roles may cover a critical authorization set, but a principal must not use all authorizations on the same object(s).*

This is a variant of Requirement 9.

**Requirement 9:** *A principal p1 may be assigned to the two exclusive roles post processor and supervisor. He may also activate them but not use them on the same object (Product Bundle).*

Within a process instance this requirement is already fulfilled by the policy for Requirement 6. If this property is supposed to be satisfied throughout the lifetime of a user, we need to introduce a policy such as the following.

```
# Object-based separation of duty (ObjSoD)
objsod { ProductBundle }
```

## 7.3 Policy Example in OPL/XML

Listing 29: OPL/XML: Banking Policy Example

```
   <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE policy_object SYSTEM "policy-object.dtd">
<policy_object>

  <policy_object_attributes>
    <attribute key="name" value="Policy-ABC"/>
    <attribute key="version" value="1.0"/>
    <attribute key="date" value="2008-02-23"/>
    <attribute key="creator" value="rwolf"/>
    <attribute key="description" value="ORKA␣Reference␣Scenario"/>
  </policy_object_attributes>

  <active_modules>
    <active_module name="module_rbac_core_policy" />
    <active_module name="module_exo_context_policy" />
    <active_module name="module_wf_core_policy" />
    <active_module name="module_sep_duty_policy" />
    <active_module name="module_wf_sep_duty_policy" />
    <active_module name="module_wf_sep_duty_cc_policy" />
    <active_module name="module_obj_sep_duty_policy" />
  </active_modules>

  <policy_object_modules>

    <!-- ========== Module RBACCore ========== -->
    <module_rbac_core_policy>

      <users>
        <user user_id="user:klaus_meier"/>
        <user user_id="user:karla_meier"/>
        <user user_id="user:jochen_schmidt"/>
        <user user_id="user:armin_mueller"/>
        <user user_id="user:susanne_schaefer"/>
      </users>

      <roles>
        <role role_id="role:clerk_preprocessor"/>
        <role role_id="role:clerk_postprocessor"/>
        <role role_id="role:supervisor"/>
        <role role_id="role:customer"/>
        <role role_id="role:manager"/>
      </roles>

      <permissions>
        <permission permission_id="permission:query_customer_data">
          <operation operation_id="query()"/>
          <object object_id="CustomerData"/>
        </permission>
        <permission permission_id="permission:update_customer_data">
          <operation operation_id="update()"/>
          <object object_id="CustomerData"/>
        </permission>
        <permission permission_id="permission:prepare_ratingreport">
          <operation operation_id="prepare()"/>
          <object object_id="RatingReport"/>
        </permission>
        <permission permission_id="permission:release_ratingreport">
          <operation operation_id="release()"/>
          <object object_id="RatingReport"/>
        </permission>
```

```xml
      <permission permission_id="permission:post_ratingreport">
        <operation operation_id="post()"/>
        <object object_id="RatingReport"/>
      </permission>
      <permission permission_id="permission:query_ratingreport">
        <operation operation_id="query()"/>
        <object object_id="RatingReport"/>
      </permission>
      <permission permission_id="permission:update_ratingreport">
        <operation operation_id="update()"/>
        <object object_id="RatingReport"/>
      </permission>
      <permission permission_id="permission:query_productbundle">
        <operation operation_id="query_avail_prod()"/>
        <object object_id="ProductBundle"/>
      </permission>
      <permission permission_id="permission:modify_productbundle">
        <operation operation_id="modify()"/>
        <object object_id="ProductBundle"/>
      </permission>
      <permission permission_id="permission:commit_productbundle">
        <operation operation_id="commit()"/>
        <object object_id="ProductBundle"/>
      </permission>
      <permission permission_id="permission:print_contract">
        <operation operation_id="print()"/>
        <object object_id="Contract"/>
      </permission>
      <permission permission_id="permission:sign_contract">
        <operation operation_id="sign()"/>
        <object object_id="Contract"/>
      </permission>
      <permission permission_id="permission:update_contract">
        <operation operation_id="update()"/>
        <object object_id="Contract"/>
      </permission>
      <permission permission_id="permission:open_account">
        <operation operation_id="open()"/>
        <object object_id="Account"/>
      </permission>
    </permissions>

    <user_assignments>
      <user_assignment user_id="user:jochen_schmidt"
                       role_id="role:clerk_preprocessor"/>
      <user_assignment user_id="user:karla_meier"
                       role_id="role:clerk_postprocessor"/>
      <user_assignment user_id="user:klaus_meier"
                       role_id="role:supervisor"/>
      <user_assignment user_id="user:armin_mueller"
                       role_id="role:manager"/>
      <user_assignment user_id="user:susanne_schaefer"
                       role_id="role:customer"/>
    </user_assignments>

    <permission_assignments>
      <permission_assignment permission_id="permission:query_customer_data"
                             role_id="role:clerk_preprocessor"/>
      <permission_assignment permission_id="permission:update_customer_data"
                             role_id="role:clerk_preprocessor"/>
      <permission_assignment permission_id="permission:prepare_ratingreport"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:post_ratingreport"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:query_ratingreport"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:update_ratingreport"
                             role_id="role:supervisor"/>
      <permission_assignment permission_id="permission:query_productbundle"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:modify_productbundle"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:print_contract"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:sign_contract"
                             role_id="role:customer"/>
      <permission_assignment permission_id="permission:sign_contract"
                             role_id="role:manager"/>
      <permission_assignment permission_id="permission:update_contract"
                             role_id="role:manager"/>
      <permission_assignment permission_id="permission:open_account"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:release_ratingreport"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:release_ratingreport"
                             role_id="role:supervisor"/>
      <permission_assignment permission_id="permission:commit_productbundle"
                             role_id="role:clerk_postprocessor"/>
      <permission_assignment permission_id="permission:commit_productbundle"
                             role_id="role:supervisor"/>
    </permission_assignments>

  </module_rbac_core_policy>


  <!-- ========== Module ExoContext ========== -->
  <module_exo_context_policy>

    <context_constraints>

      <context_constraint cc_id="cc:cc1">
        <context_function_id id="equal-or-less-than"/>
        <context_function_parameters>
          <parameter value="creditbureau_provider.get_wfi_amount()" type="int"
                     context="yes" />
          <parameter value="100000" type="int" context="no"/>
        </context_function_parameters>
      </context_constraint>

      <context_constraint cc_id="cc:cc2">
        <context_function_id id="more-than"/>
        <context_function_parameters>
          <parameter value="creditbureau_provider.get_wfi_amount()" type="int"
                     context="yes" />
          <parameter value="100000" type="int" context="no"/>
        </context_function_parameters>
      </context_constraint>

      <!-- For Requirement 3 (Part 1) -->
      <context_constraint cc_id="cc:cc3">
```

```
          <context_function_id id="equals"/>
          <context_function_parameters>
            <parameter value=
              "customerinformation_provider.get_customer_type(parameters.cust-id)"
              type="string" context="yes" />
            <parameter value="industrial" type="string" context="no" />
          </context_function_parameters>
        </context_constraint>

        <!-- For Requirement 4 (Part 1) -->
        <context_constraint cc_id="cc:cc4">
          <context_function_id id="less-than"/>
          <context_function_parameters>
            <parameter value="ratingserver_provider.get_internal_rating()"
                       type="int" context="yes" />
            <parameter value="0" type="int" context="no" />
          </context_function_parameters>
        </context_constraint>

      </context_constraints>

      <context_constraint_assignments>
        <pacc cc_id="cc:cc1" permission_id="permission:release_ratingreport"
                             role_id="role:clerk_postprocessor"/>
        <pacc cc_id="cc:cc2" permission_id="permission:release_ratingreport"
                             role_id="role:supervisor"/>
        <pacc cc_id="cc:cc1" permission_id="permission:commit_productbundle"
                             role_id="role:clerk_postprocessor"/>
        <pacc cc_id="cc:cc2" permission_id="permission:commit_productbundle"
                             role_id="role:supervisor"/>
      </context_constraint_assignments>

    </module_exo_context_policy>


    <!-- ========== Module WFCore ========== -->
    <module_wf_core_policy>

      <task_permission_assignments>

        <task_permission_assignment task_id="task:1_input_customer_data"
                                    permission_id="permission:query_customer_data"
        />
        <task_permission_assignment task_id="task:1_input_customer_data"
                                    permission_id="permission:update_customer_data"
        />
        <task_permission_assignment task_id="task:2_customer_ident"
                                    permission_id="permission:query_customer_data"
        />
        <task_permission_assignment task_id="task:3a_check_cred_worthin"
                                    permission_id="permission:prepare_ratingreport"
        />
        <task_permission_assignment task_id="task:3b_check_cred_worthin"
                                    permission_id="permission:release_ratingreport"
        />
        <task_permission_assignment task_id="task:3c_check_cred_worthin"
                                    permission_id="permission:post_ratingreport"
        />
        <task_permission_assignment task_id="task:4_check_rating"
                                    permission_id="permission:query_ratingreport"
        />
        <task_permission_assignment task_id="task:5_bank_signs_form"
                                    permission_id="permission:update_ratingreport"
        />
        <task_permission_assignment task_id="task:6_choose_bundled_prod"
                                    permission_id="permission:query_productbundle"
        />
        <task_permission_assignment task_id="task:7a_price_bundled_prod"
                                    permission_id="permission:modify_productbundle"
        />
        <task_permission_assignment task_id="task:7b_price_bundled_prod"
                                    permission_id="permission:commit_productbundle"
        />
        <task_permission_assignment task_id="task:8_print_opening_form"
                                    permission_id="permission:print_contract"
        />
        <task_permission_assignment task_id="task:9_customer_signs_form"
                                    permission_id="permission:sign_contract"
        />
        <task_permission_assignment task_id="task:10_bank_signs_form"
                                    permission_id="permission:sign_contract"
        />
        <task_permission_assignment task_id="task:10_bank_signs_form"
                                    permission_id="permission:update_contract"
        />
        <task_permission_assignment task_id="task:11_open_account"
                                    permission_id="permission:open_account"
        />

      </task_permission_assignments>

      <task_role_assignments>

        <task_role_assignment task_id="task:1_input_customer_data"
                              role_id="role:clerk_preprocessor"/>
        <task_role_assignment task_id="task:2_customer_ident"
                              role_id="role:clerk_preprocessor"/>
        <task_role_assignment task_id="task:3a_check_cred_worthin"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:3b_check_cred_worthin"
                              role_id="role:supervisor"/>
        <task_role_assignment task_id="task:3b_check_cred_worthin"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:3c_check_cred_worthin"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:4_check_rating"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:5_bank_signs_form"
                              role_id="role:supervisor"/>
        <task_role_assignment task_id="task:6_choose_bundled_prod"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:7a_price_bundled_prod"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:7b_price_bundled_prod"
                              role_id="role:supervisor"/>
        <task_role_assignment task_id="task:7b_price_bundled_prod"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:8_print_opening_form"
                              role_id="role:clerk_postprocessor"/>
        <task_role_assignment task_id="task:9_customer_signs_form"
```

```
                              role_id="role:customer"/>
        <task_role_assignment task_id="task:10_bank_signs_form"
                              role_id="role:manager"/>
        <task_role_assignment task_id="task:11_open_account"
                              role_id="role:clerk_postprocessor"/>

      </task_role_assignments>

    </module_wf_core_policy>


    <!-- ========== Module ExoContext ========== -->
    <module_sep_duty_policy>

      <!--  For Requirement 1 -->
      <static_separation_of_duty>
        <critical_role_sets>
          <critical_role_set cardinality="1">
            <critical_roles>
              <critical_role role_id="role:clerk_preprocessor"/>
              <critical_role role_id="role:clerk_postprocessor"/>
            </critical_roles>
          </critical_role_set>
        </critical_role_sets>
      </static_separation_of_duty>

      <!-- For Requirement 2 (Part 1) -->
      <dynamic_separation_of_duty>
        <critical_role_sets>
          <critical_role_set cardinality="1">
            <critical_roles>
              <critical_role role_id="role:clerk_preprocessor"/>
              <critical_role role_id="role:clerk_postprocessor"/>
            </critical_roles>
          </critical_role_set>
        </critical_role_sets>
      </dynamic_separation_of_duty>

    </module_sep_duty_policy>


    <!-- ========== Module WFSepDuty ========== -->
    <module_wf_sep_duty_policy>

      <hdsodtp>
        <!-- For Requirement 2 (Part 2) -->
        <hdsodtp_partitioning>
          <hdsodtp_partition>
            <partition_task task_id="task:1_input_customer_data"/>
            <partition_task task_id="task:2_customer_ident"/>
          </hdsodtp_partition>
          <hdsodtp_partition>
            <partition_task task_id="task:3a_check_cred_worthin"/>
            <partition_task task_id="task:3b_check_cred_worthin"/>
            <partition_task task_id="task:3c_check_cred_worthin"/>
            <partition_task task_id="task:4_check_rating"/>
            <partition_task task_id="task:6_choose_bundled_prod"/>
            <partition_task task_id="task:7a_price_bundled_prod"/>
            <partition_task task_id="task:7b_price_bundled_prod"/>
            <partition_task task_id="task:9_print_opening_form"/>
            <partition_task task_id="task:11_open_account"/>
          </hdsodtp_partition>
        </hdsodtp_partitioning>

        <!-- For Requirement 6 -->
        <hdsodtp_partitioning>
          <hdsodtp_partition>
            <partition_task task_id="task:7a_price_bundled_prod"/>
          </hdsodtp_partition>
          <hdsodtp_partition>
            <partition_task task_id="task:7b_price_bundled_prod"/>
          </hdsodtp_partition>
        </hdsodtp_partitioning>
      </hdsodtp>

    </module_wf_sep_duty_policy>


    <!-- ========== Module WFSepDutyCC ========== -->
    <module_wf_sep_duty_cc_policy>

      <hdsodtpcc>

        <!--  For Requirement 3 (Part 2) -->
        <hdsodtpcc_partitioning cc_id="cc:cc3">
          <hdsodtpcc_partition>
            <cc_partition_task task_id="task:1_input_customer_data"/>
          </hdsodtpcc_partition>
          <hdsodtpcc_partition>
            <cc_partition_task task_id="task:2_customer_ident"/>
          </hdsodtpcc_partition>
        </hdsodtpcc_partitioning>

        <!-- For Requirement 4 (Part 2) -->
        <hdsodtpcc_partitioning cc_id="cc:cc4">
          <hdsodtpcc_partition>
            <cc_partition_task task_id="task:4_check_rating"/>
          </hdsodtpcc_partition>
          <hdsodtpcc_partition>
            <cc_partition_task task_id="task:3a_check_cred_worthin"/>
            <cc_partition_task task_id="task:3b_check_cred_worthin"/>
            <cc_partition_task task_id="task:3c_check_cred_worthin"/>
            <cc_partition_task task_id="task:6_choose_bundled_prod"/>
            <cc_partition_task task_id="task:7a_price_bundled_prod"/>
            <cc_partition_task task_id="task:7b_price_bundled_prod"/>
            <cc_partition_task task_id="task:8_print_opening_form"/>
            <cc_partition_task task_id="task:11_open_account"/>
          </hdsodtpcc_partition>
        </hdsodtpcc_partitioning>

        <!-- For Requirement 7 -->
        <hdsodtpcc_partitioning cc_id="cc:cc3">
          <hdsodtpcc_partition>
            <cc_partition_task task_id="task:1_input_customer_data"/>
            <cc_partition_task task_id="task:2_customer_ident"/>
            <cc_partition_task task_id="task:3a_check_cred_worthin"/>
            <cc_partition_task task_id="task:3b_check_cred_worthin"/>
            <cc_partition_task task_id="task:3c_check_cred_worthin"/>
            <cc_partition_task task_id="task:4_check_rating"/>
            <cc_partition_task task_id="task:5_bank_signs_form"/>
```

```
            <cc_partition_task task_id="task:6_choose_bundled_prod"/>
            <cc_partition_task task_id="task:7a_price_bundled_prod"/>
            <cc_partition_task task_id="task:7b_price_bundled_prod"/>
            <cc_partition_task task_id="task:8_print_opening_form"/>
            <cc_partition_task task_id="task:9_customer_signs_form"/>
          </hdsodtpcc_partition>
          <hdsodtpcc_partition>
            <cc_partition_task task_id="task:10_bank_signs_form"/>
          </hdsodtpcc_partition>
        </hdsodtpcc_partitioning>

      </hdsodtpcc>

    </module_wf_sep_duty_cc_policy>


    <!-- ========== Module ObjSepDuty ========== -->
    <module_obj_sep_duty_policy>

      <objsods>
        <!-- For Requirement 9 -->
        <objsod object_id="ProductBundle" />
      </objsods>

    </module_obj_sep_duty_policy>

  </policy_object_modules>

</policy_object>
```

## 8 Conclusions

Within this report we have presented the language specification of OPL, the policy language that was developed in the ORKA research project. We have outlined our modularization approach to support a variety of access control principles. The current library of policy modules supports role hierarchies, separation of duty constraints, context constraints, chinese wall, object-based separation of duty constraints, workflow-based constraints, history-based constraints, bind of duty constraints, and cardinality constraints. We have explained how to combine these policy modules in order to get the required expressiveness without increasing the complexity in language specification and management. The OPL language is based on a formal semantics by means of Object Z and has a XML-based expression syntax to specify policies persistently. Within the ORKA research project the OPL language has been practically tested using a real-world scenario, various performance analyses have been performed. ORKA has developed and implemented a large set of tools for OPL such as various policy enforcement engines, a graphical policy administration tool, and tools for policy validation and for reasoning about policy properties.

# References

[1] The ORKA Project Homepage http://www.orka-projekt.de/index-en.htm (2009-01-15).

[2] G.-J. Ahn. *The RCL 2000 language for specifying role-based authorization constraints.* PhD thesis, George Mason University, Fairfax, Virginia, 1999.

[3] Christopher Alm. An Extensible Framework for Specifying and Reasoning About Complex Role-Based Access Control Models. Technical Report MIP-0901, Fakultät für Informatik und Mathematik, Universität Passau, Germany, 2009. http://www.fim.uni-passau.de/en/research/forschungsberichte/mip-0901.html (2009-01-29).

[4] E. Bertino, E. Ferrari, and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. *ACM TISSEC*, 2(1):65–104, 1999.

[5] Elisa Bertino, Jason Crampton, and Federica Paci. Access Control and Authorization Constraints for WS-BPEL. In *ICWS '06: Proceedings of the IEEE International Conference on Web Services (ICWS'06)*, pages 275–284. IEEE Computer Society, 2006.

[6] R. Bhatti, J. Joshi, E. Bertino, and A. Ghafoor. Access Control in Dynamic XMLBased Web Services with X-RBAC. 2003.

[7] David F.C. Brewer and Michael J. Nash. The Chinese Wall Security Policy. In *Proceedings of IEEE Symposium on Security and Privacy*, pages 206–214, 1989.

[8] D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. *IEEESSP*, pages 184–194, 1987.

[9] Nicodemos Damianou, Naranker Dulay, Emil Lupu, and Morris Sloman. The Ponder Policy Specification Language. *LNCS*, 1995:18–39, 2001.

[10] Roger Duke and Gordon Rose. *Formal Object-Oriented Specification Using Object-Z.* Macmillan Press, 2000.

[11] D. Ferraiolo and R. Kuhn. Role-Based Access Control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.

[12] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control.* Artech House Publishers, 2003.

[13] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Trans. Inf. Syst. Secur.*, 4(3):224–274, 2001.

[14] C.K. Georgiadis, I. Mavridis, G. Pangalos, and R.K. Thomas. Flexible team-based access control using contexts. pages 21–27, May 2001.

[15] V. D. Gligor, S. I. Gavrila, and D. Ferraiolo. On the Formal Definition of Separation-of-Duty Policies and their Composition. In *Proceedings of the 1998 IEEE Symposium on Security and Privacy*, pages 172–185. IEEE, 1998.

[16] Michael Hitchens and Vijay Varadharajan. Tower: A Language for Role Based Access Control. In *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*, pages 88–106. Springer-Verlag, 2001.

[17] Jacques Wainer and Paulo Barthelmess and Akhil Kumar. W-RBAC - A Workflow Security Model Incorporating Controlled Overriding of Constraints. *Int. J. Cooperative Inf. Syst.*, 12(4):455–485, 2003.

[18] J. Joshi, E. Bertino, U. Latif, and A. Ghafoor. A generalized temporal role-based access control model. *IEEE Trans. Knowl. Data Eng.*, 17(1):4–23, 2005.

[19] Tim Moses et al. eXtensible Access Control Markup Language (XACML) Version 2.0. 2005. OASIS Standard.

[20] M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. pages 201–207, 1990.

[21] 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.

[22] Ravi Sandhu. Role hierarchies and constraints for lattice-based access control. In *ESORICS 1996: Proceedings of the Fourth European Symposium on Research in Computer Security*, pages 65–79. Springer-Verlag, 1996.

[23] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, 1996.

[24] Andreas Schaad, Volkmar Lotz, and Karsten Sohr. A Model-checking Approach to Analysing Organisational Controls in a Loan Origination Process. In *SACMAT*, pages 139–149, 2006.

[25] Richard T. Simon and Mary Ellen Zurko. Separation of Duty in Role-Based Environments. In *IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.

[26] Karsten Sohr, Michael Drouineaud, and Gail-Joon Ahn. Formal Specification of Role-Based Security Policies for Clinical Information Systems. In *SAC '05: Proceedings of the 2005 ACM Symposium on Applied Computing*, pages 332–339. ACM, 2005.

[27] American National Standard. *Role Based Access Control.* 2004. ANSI INCITS 359-2004, American National Standards Institute.

## Acknowledgments

We are grateful to Roland Illig with whom we worked together on the definition of context constraints. Thank goes to Michael Drouineaud for the fruitful discussions about the Object Z specifications. We would also like to thank Ute Faltin for her support in the XML specifications.