



Technologie-Zentrum Informatik

Technical Report 51

**A Classification Framework Designed for
Advanced Role-based Access Control Models
and Mechanisms**

**Christopher Alm
Michael Drouineaud
Ute Faltin
Karsten Sohr
Ruben Wolf**

TZI-Bericht Nr. 51
2009



Universität Bremen

TZI-Berichte

Herausgeber:
Technologie-Zentrum Informatik
Universität Bremen
Am Fallturm 1
28359 Bremen
Telefon: +49-421-218-7272
Fax: +49-421-218-7820
E-Mail: info@tzi.de
<http://www.tzi.de>

ISSN 1613-3773

A Classification Framework Designed for Advanced Role-based Access Control Models and Mechanisms ^{*}

Christopher Alm¹, Michael Drouineaud², Ute Faltin³,
Karsten Sohr², and Ruben Wolf³

¹ University of Passau, Germany,
`christopher.alm@uni-passau.de`,

² Center for Computing Technologies, Bremen, Germany,
`{mdruid|sohr}@tzi.de`

³ Fraunhofer Institute for Secure Information Technology, Darmstadt, Germany,
`{faltin|rwolf}@sit.fraunhofer.de`

Abstract. Since its emergence in the early 1990s, role-based access control (RBAC) has gained more and more popularity. Its flexibility has been leading to a multitude of proposed access control models and mechanisms based on the role paradigm. They adapt RBAC according to the specific needs of specific settings, for example, by providing support for delegation of rights in workflow environments [6]. The goal of this paper is to develop a holistic classification framework for such models and mechanisms. By using this framework, firstly, a comparison of different models and mechanisms can be achieved. Secondly, considering them from the perspective of the classification, requirements of a specific setting can be mapped onto a model or mechanism, once the existing models and mechanisms are classified. This is particularly helpful for security officers of organizations who need to evaluate different models and mechanisms. Finally, the framework assists designers of access control models by giving them a structured view on the properties such models can have. We apply the framework to BEA WebLogic Server [10], Adage [71], and X-GTRBAC [17].

1 Introduction

Since its emergence in the early 1990s, role-based access control (RBAC) has gained more and more popularity. It turned out that this access control paradigm is well suited for organizations for reasons such as the following: the role concept reflects the structure of an organization by using roles to represent job functions. As one consequence, access control models and mechanisms based on RBAC can provide an efficient and scalable authorization management. Furthermore, RBAC offers high flexibility in that role-based access control models and mechanisms

^{*} This work was supported by the German Ministry of Education and Research (BMBF) as part of the project “ORKA”, <http://www.orka-projekt.de>

can be accommodated to express fine-grained authorization policies for various environments as well as a wide range of organizational control principles such as separation of duty and delegation of rights.

This flexibility, however, has been leading to a multitude of proposed access control models and mechanisms refining or changing RBAC—for which an ANSI standard model [3] was approved in 2004—in order to adapt to the specific needs of specific settings. Such refinements, for example, add delegation support or add a concept to make access decisions dependent on context information. Crampton and Khambhammettu recently introduced an RBAC extension to support delegation of rights [23]. Neumann and Strembeck developed the xORBAC access control mechanism which provides a flexible way to enforce context sensitive authorization constraints for RBAC [51, 50]. We call such models or mechanisms *advanced role-based access control models* or *mechanisms*, respectively. Note that some advanced role-based access control models use only basic ideas of standard RBAC such as the role concept and come up with a model that is different in many aspects. An example for this case is the Ponder framework [24].

The problem is that there has not been much research neither on the comparison of different models and mechanisms nor on the analysis of their properties in a holistic approach including various perspectives at once such as validation, enforcement, and administration. In particular, many proposals are focused on the pros and cons of the innovative features of their proposed model rather than on the drawbacks of the model from different perspectives. Research in this field is at least important for two groups of people, though. While analyzing or developing an advanced role-based access control model or a mechanism, on the one hand, security researchers and security experts need a structured view on the properties that such a model or a mechanism can have. On the other hand, security administrators and security officers of organizations have to compare different models and mechanisms when they have to decide which one to deploy in their organization. They have to balance the pros and cons of each model and mechanism that comes into question.

The goal of this paper is therefore to develop a classification framework for advanced role-based access control models and mechanisms. This framework is an extensible and structured set of aspects and properties that are used as classification axes. The framework is designed in a holistic way using a specific methodology, and it encompasses a well-considered variety of aspects and properties. It is important to note that even though our framework was created having in mind role-based access control, it can be applied, for the most part, to other security models and mechanisms that include only minor parts of RBAC or that even do not deal with RBAC at all. This is the reason why we added “designed for” to the title of this paper.

By using such a classification framework, firstly, it is possible to compare two different access control models or mechanisms with each other. Secondly, when searching for an appropriate model or mechanism for a specific setting, it can be checked whether the requirements of that setting can be mapped onto an existing model or mechanism by considering the requirements from the perspective of the

classification. Finally, the framework offers a structured view on properties of access control models and mechanisms while developing a new access control architecture.

This paper is organized as follows. After introducing some basic notions in Section 2, it introduces the proposed classification framework in Section 3. Subsequently, we show applications of the framework in Section 4. Section 5 evaluates the framework by discussing our approach and by assessing our methodology. Section 6 gives an outline of related approaches to classification and property analysis of security models and mechanisms. In Section 7 we conclude the paper, summarize our findings, and give further research ideas in this field.

2 Basic Terms

2.1 Model, Policy, Mechanism

An (*authorization*) *policy*⁴ is a document representing the 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. It is based on an *authorization model* defining a (formal or informal) semantic domain for the entities this document may consist of and thus giving a semantics to the policy. The policy can be seen as describing an instance of an authorization model or a state of the authorization model, in case the model has a notion of state. If the policy 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* is an implementation of the components of an authorization architecture that decide access requests and enforce these decisions. Hence, the access control mechanism implements the authorization model. The part of the access control mechanism that comes to the access decisions is called *access decision component*. The decision component is configured by loading a machine readable version of the policy (i.e. written in a policy language) and it can then make decisions based on this policy. The decision component knows how to interpret the policy because it is implemented according to the authorization model giving the semantics to the policy.

⁴ Since we deal only with authorization policies in this paper, we mostly omit the term “authorization” while speaking of policies throughout the rest of this paper.

2.2 Role-Based Access Control

Since the general concepts of RBAC are well-understood and extensively described in the literature [27, 58, 3, 28], we briefly describe only the key aspects. 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.

An important advanced concept of RBAC is *authorization constraints*. 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 [49, 22, 31, 62, 1], constraints on delegation [64], cardinality constraints [59], context constraints [30, 39], and workflow constraints [15].

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 [57].

3 The Classification Framework

3.1 Definition of Framework Structure

The goal of this paper is to classify access control models and mechanisms, which we call here *classification objects*, by means of their properties. Consider the set of all properties that any possible classification object can have. Examples of such properties are the availability of a graphical administration tool, the ability of a model to express separation of duty, or the distribution of a mechanism as a commercial product. A *classification axis* is a subset of this set representing a certain range of semantically related properties, i.e. a certain aspect of a classification object. The elements of an axis are called *classification values*. For example, the available programming language interfaces of an access control mechanism could be a classification axis. Its values would be sets of programming languages such as {Java, C++} or {PHP, Perl}. The *classification framework* presented in this paper is a set of classification axes. The goal is to design the framework in such a way that the union of all axes takes into account as many properties as possible. A *classification* is an assignment of exactly one classification value per each classification axis of the framework to a classification object. In doing so, a classification represents the actual properties of a classification object. It should be noted that the term classification may also refer to the process of assigning classification values to a classification object. We now give some remarks on the characteristics of the classification framework and the classification axes:

- The values of a classification axis have to be mutually exclusive because a classification can only assign one value per axis to an object. If there is an axis having two appropriate values for a certain classification object, a new value combining them has to be introduced.
- Each axis must be exhaustive in order to be able to assign a value of each axis to an object. If it is neither possible to find an appropriate value of an axis for a certain object nor can the axis be extended without making it semantically inconsistent, default values such as “not applicable” or “undefined” can be used. In most cases the default values can be derived from the context and need not to be defined explicitly.
- There are two types of axes which we refer to as *continuous* and *discrete* axes. A discrete axis is an axis which has a set of classification values pre-defined through its definition in the framework. During classification, one has either to choose a value from the axis definition or, in case no value fits in, to create a new value by extending the axis definition. Note that values on discrete axes can be as simple as “supported” and “not supported”. In contrast to this, a continuous axis is intended to introduce for each classification object a new value and does not provide a pre-defined list of values. This is the case, for example, if a classification value is some sort of abstract description such as a description to what extent a model differs from the ANSI RBAC standard.
- The framework does intendedly not assign an intrinsic weighting to the axes because different users of the framework may have completely different priorities. While one user may be focused on the support of inference rules and validation capabilities, another user may be more interested in the expressive power of a model. The importance of axes has to be evaluated by the framework’s user during classification. To use the framework for requirements mapping we provide a way to set up such a weighting in Section 4.
- It is not possible to provide a mathematical similarity measure within the framework for the direct comparison of classification objects. The reason for this is that there are classification axes whose values are conceptual descriptions which cannot be translated into a mathematical representation nor can such axes have an ordering. This is particularly true for all continuous axes. Hence, direct comparisons have to be carried out on the semantic level.

The classification axes are arranged in eight groups in order to provide a structured view during classification. The grouping gives an idea of how different classification axes are correlated semantically. As far as the classification is concerned, however, this grouping is not necessary.

The framework is designed in a holistic way, but it does not claim to be exhaustive. This is because it is not possible to list all properties that any access control model or mechanism can have. New models may add whole new aspects and properties. Therefore the framework can be extended by adding further axes as well as by adding further values to existing axes, according to unforeseen needs arising in the future. Due to this extensibility we speak of a *classification framework* rather than of a *classification scheme*. However, one could call a certain

instance of this framework a classification scheme, i.e. a fixed set of axes with fixed values. When adapting the framework, objects which are already classified need to be classified again with respect to any modified or newly introduced axis.

3.2 Specification of Classification Axes

We propose an initial set of classification axes which is defined in this subsection according to the grouping of axes mentioned in the previous subsection. Each group contains a description of which axes in the group have in common or how the axes are related to each other. Afterwards the axes of the group are listed. Each axis definition contains a description of how classification values in the axes are semantically related to each other followed by a description or list of the values. While italics may emphasize the definition of an axis value, axis names are printed in bold font followed by an axis identifier.

Model Specification This group contains aspects and properties regarding the specification of the classification object.

The **level of abstraction (M1)** forms an axis of our framework: all classification objects describe access control systems, however, at possibly different levels of abstraction. On the one hand an object may be as concrete as the code of a programming language where we speak of an *implemented mechanism* or an *implementation*. On the other hand an abstract *model* may only take into account basic requirements and properties of a system. Thereby the model specifies the behavior of the system at an abstract level. There are also levels in between that answer more *architectural* questions and how a system is going to be *designed*—i.e. how to implement a system according to what is required by the specification. The boundaries between these levels are not clear-cut and we may need to refine them as necessary.

An implementation is always made according to some model. We have to distinguish, however, whether a classification object proposes a new model including an implementation or whether it is just an implementation based on a model which is already in place or which is not really innovative.

The method used for the **specification of the (underlying) authorization model (M2)** forms an axis. A value of this axis, firstly, states the method or technique used to define the authorization model of a classification object (i.e. the semantic domain for the policies of the classification object (confer Section 2)). If the classification object is a mechanism, the value refers to its underlying authorization model. This can be, for example, an UML model or just a prose description. In case that no well-known method has been used, a description of the method should be added. Secondly, the used method is characterized using the following attributes.

- The method is called a *formal method* if the specification of the model is fully expressed by some mathematical formalism or mathematical theory. In

this case we also speak of a *formal specification* of the authorization model. Furthermore, policies of such a model are said to have a *formal semantics*.

- The method is called *semi-formal* if at least a part of the model is expressed by some mathematical formalism or mathematical theory.
- The method is called *informal* if it does not have any mathematical elements in it.
- The method is called *structured* if it makes use of diagrams, flow charts, or similar techniques in addition to prose text descriptions to define the authorization model. An informal modeling language can also be used⁵.

A few examples illustrate these definitions.

- The method of just using plain prose text descriptions to specify a model is informal as well as not structured.
- Formal methods dedicated to software specification and modeling such as Isabelle/HOL, Petri Nets, and Z can be used to state a full authorization model or parts of it [3, 70, 25].
- Using common mathematical English together with formulas, as it is done in mathematical textbooks, can also be regarded as a formal method. This method has widely been used to define authorization models [59, 15, 12, 38]. In some cases, however, if the definition of the model does not provide the full rigor, necessary to be considered a formal method, it might be appropriate to speak of a semi-formal method instead.
- UML plus OCL is an example of a structured method that can be used to specify an authorization model [61, 2].
- The definition of the authorization models where XACML policies [47] and Ponder policies [48] are based on is done through diagrams and prose text descriptions. In particular, these models are informal.

Note that also a combination of (semi-)formal methods, structured methods, and prose descriptions can be used to define an authorization model.

We form an axis out of the **typical application area (M3)** of a classification object. This axis is a continuous axis and therefore we just give a few examples illustrating how possible values may look like. On the one hand we consider examples regarding implementations of access control mechanisms. A mechanism may be designed in a way that it is used typically in *web service environments*. Another mechanism could be used typically together with *workflow management systems (WFMS)*. On the other hand this axis can also be applied to access control models. A model may be used primarily to specify security objectives in the healthcare sector such as authorization requirements of an *electronic patient record system*. Similarly, a model may provide security principles that are especially required in *banking applications* such as separation of duty.

By the **scope (M4)** of a classification object we mean the application domain or usage the object is designed for. This axis is close to the axis about the

⁵ “Informal” means here that the language does not have a formal semantics.

typical application area. However, it takes into account the main design goals of a classification object rather than how the object is actually being used. This axis is also continuous. We give again a few examples for the purpose of illustration while distinguishing models and implemented mechanisms. Firstly, a model may be designed for administration and management of permissions which we then call an *administrative model*. Secondly, a model may have a broader scope than just authorization. It may be designed, for example, to specify *authentication* and *audit requirements*⁶. Thirdly, for mechanisms we can distinguish whether they are designed for making *access decisions* or for the *enforcement* of such decisions.

As mentioned in the introduction, an **ANSI standard for role-based access control (M5)** was established in 2004 [3]. We add an axis dealing with the common ground between a classification object and this standard. A value of this axis is a description of what the object and the standard have in common, where their differences are, and how they are related to each other. Even though it is not possible to give a pre-defined list of values for this axis, there are basically five cases out of which values are generated. Firstly, an object may have (almost) nothing in common with the standard. Secondly, an object may have basic ideas such as the role concept in common with the standard. Thirdly, an object may be derived from the standard in a sense that it changes some ideas and adds some ideas. Fourthly, an object may be based on the standard in that it adds some ideas or concepts but does not change anything. Finally, a mechanism may just implement the standard.

Policy Expressiveness This group encompasses expressiveness aspects including the supported organizational control principles, the suitability of a model to specify policies for certain system layers, and the suitability of a model to specify policies for workflow environments.

The suitability of a model to specify policies for a certain **system layer (P1)** forms an axis. A policy may define high level security objectives for entities such as business processes, real employees, and information assets. On a lower layer, a policy may control authorization in terms of files or data base entries. An even lower layer would take memory locations or data packets in a network into account. A value of this axis describes which system layers a classification object is able to specify a policy for. A model or mechanism may be *hybrid* in that it can be mapped onto different system layers, for example, by interpreting permissions accordingly.

The **range of organizational control principles (P2)** that can be expressed in a policy of a classification object forms an axis of our framework. A value of this axis is a set of organizational control principles. Examples of organizational principles are separation of duty [62], delegation [23], and obligation [69]. It is necessary to have a clear definition of each occurring organizational control

⁶ Logging and auditing are necessary in many cases for the compliance of an organization to external law.

principle in order to provide a common understanding on principles during classification. Giving such definitions, however, would be well beyond the scope of this paper. Schaad’s framework for organizational control principles [60] could be used to define the values of this axis.

An axis illustrating the **types of authorization constraints (P3)** supported by a classification object is difficult to accomplish since the definition and realization of authorization constraints depend heavily on the underlying access control model (in contrast to organizational control principles). For example, using the standard of RBAC an authorization constraint can be defined as a formula of an appropriate logic which is a necessary condition for policy validity. Furthermore, there are numerous attempts to elicit types of authorization constraints and to classify and unify authorization constraints [51, 1, 44]. In order to make this axis applicable to a range of different classification objects an appropriate definition and classification of authorization constraints have to be chosen. It should be noted that in standard RBAC, various organizational control principles can be realized by using authorization constraints. For example, the principle of separation of duty can be expressed through mutual exclusion constraints placed on the set of roles or on the set of permissions, respectively.

We introduce an axis regarding **workflow policy support (P4)**. A value of this axis contains two parts. Firstly, it states which workflow-related organizational control principles can be modeled by a classification object. Examples are prerequisite steps (i.e. a workflow step needs some other workflow step to be completed first in order to be accessible) and history-based separation of duty [62]. Note that this part can also be covered by P2. Secondly, in order to be able to specify such a *workflow-aware* policy an authorization model needs to model at least a part of the workflow environment. Thus, in the second part of a value of this axis we check the following characteristics regarding how workflows are modeled in the (underlying) authorization model of a classification object.

- *Set of tasks*: the basic information needed to state workflow constraints is: which tasks belong to which workflow schema. It is then, for example, possible to specify operational separation of duty [62].
- *History of tasks*: the history of tasks records all tasks that are completed within a workflow instance. Together with the set of tasks, it is now possible to specify history-based separation of duty [31] as well as the aforementioned prerequisite step principle.
- *Routing*: if an authorization model includes workflow routing concepts such as serialization, selection, iteration, and parallelization of steps [67], it is able to control the order of events in a fine-grained manner. Examples of such models are SecureFlow [45], Kandala and Sandhu’s model [41], and Bertino, Ferrari, and Atluri’s model [16].

However, note that the more information about the workflow environment is modeled within the authorization model, the tighter is the integration necessary between the workflow management system and the access control mechanism. A synchronization problem, e.g. regarding workflow schema definitions, might occur between them.

Policy Language Axes in this group encompass properties regarding the policy specification and presentation techniques. This includes policy languages and policy modeling techniques.

A value of the axis **formal syntax (L1)** states whether a formal syntax to specify a policy, as defined in Section 2, is available or not. If it is available, we call this syntax a *policy language*. Note once again that having a formal syntax is a prerequisite for making a policy interpretable and manageable by an access control mechanism. Furthermore, we are now able to combine the value of this axis with the value of the axis M2. Thus a value of this axis also takes into account the following two cases:

- *Language-centric authorization model*: the development of a language-centric authorization model is focused on the definition of a policy language (syntax). To be more precise, the model is defined by assigning a semantics (confer M2) to the expressions of the language. Examples of such models are Ponder [48], XACML [47], EPAL [5], and Rei [40]. While the standard of XACML defines only an informal semantics, as mentioned above, there are attempts to retrofit XACML with a formal semantics [20, 34, 46]. However, it is also possible to define formal syntax and formal semantics at once, as in ASL [56].

It should be noted that in the language-centric case, the policy language and its authorization model are closely related to each other and they can hardly be separated.

- *Model-centric authorization model*: in contrast to the language-centric approach, the development of a model-centric authorization model is focused on the definition of the model itself (i.e. the semantic domain for the policies). This is, for example, the case with RBAC [59], which has a formal specification based on set theory and first-order logic. Such models need to be equipped with a policy language if there is an access control mechanism implemented according to them. This has been done, for example, in Tower [33], X-RBAC [18], and X-GTRBAC [17].

It should be noted that in the model-centric approach a policy language and its authorization model are not as closely related as in the language-centric approach. Thus it is easier to develop a different syntax if necessary.

The **type of policy representation (L2)** refers to the way the policy is presented to a certain entity in the chain from the point of policy specification by a human security administrator to the point of policy interpretation by the policy decision engine (in order to make policy decisions). These types include dedicated *policy specification languages*, *graphical policy representations*, and non-human-readable *technical policy representations*, for example, in terms of memory objects or database entries. Most of them can be regarded as policy languages in the sense of axis L1. Note that there may be further policy representations between policy specification and policy interpretation, for example to make a policy readable for a validation tool. In contrast to this, some classification objects only have one policy representation by using a dedicated policy

specification language. For example, BEA's WebLogic Server can be configured to use only one policy representation in XACML [47, 9]. An example of a graphical policy representation is UML [61]. There is also a graphical definition of policies for the dynamically typed access control model (DTAC) [66]. Only a technical policy representation in terms of memory objects for policy decisions is available in xORBAC [65, 51]. In this implemented access control system, the specification takes places by means of a graphical tool. A value of this axis is the set of all different types of policy representations used to represent the policy for the given classification object.

Finally, if there is a policy specification language the **language properties (L3)** axis encompasses the characteristics of the language. A value is a set of properties such as extensibility, modularity, declarative design, and formal definition of semantics.

Enforcement This group contains the classification aspects concerning the enforcement of access control policies. Enforcement is usually realized by a reference monitor concept [13]. Specifically, in distributed environments such as Service Oriented Architectures (SOA) a reference monitor consists of two components, namely, a policy decision point (PDP) and a policy enforcement point (PEP) [47]. The former computes the access decision according to the defined access control policy whereas the latter is responsible for the enforcement of that access control decision. In the following, the axes of this group are described.

Within the axis **reference-monitor topology (E1)** we distinguish the three topologies: *systemwide enforcement of the reference monitor*, *enforcement of the reference monitor at resource level*, and *application-based reference monitor* [13]. In the first case, a single instance of a reference monitor is running systemwide, most likely, at the kernel of the operating system. In the second case, one or more resources are protected by a reference monitor such as certain service components in SOA. The last case indicates that the application itself is responsible for enforcing the access control policy.

The axis **support for decision and enforcement in decentralized environments (E2)** describes whether access decision and enforcement in decentralized environments is supported or not. This is an important aspect for the scalability of a system.

In distributed environments, both the access control decision and the enforcement could be carried out locally or remotely. For example, a remote authorization server could be used for the calculation of the access control decision. The values of the axis **remote decision and enforcement (E3)** are pairs of the form (*kind of decision, kind of enforcement*), which indicate the kind of decision and enforcement. For example, (*remote decision, local enforcement*) expresses the fact that a classification object supports remote PDPs and local PEPs.

In addition, PEPs could be integrated into a WFMS. Hence, we introduce a further axis called **enforcement in workflow environments (E4)**. We define the following as the values of this axis: *part of WFMS*, *integration with WFMS possible*, and *no integration with WFMS*. This axis is closely related to

P4 because, as already mentioned, the more information about the workflow environment is included in the authorization model the tighter is the integration of the access control mechanism and the WFMS.

Validation Thorough understanding of the uses and abuses of an IT system is the first step toward economical and effective security. Thus, validation is specifically helpful at the beginning of the design process. Moreover, high assurance levels of IT security standards such as Common Criteria require the application of formal validation methods. The group validation consists of five axes describing aspects relevant for validation—namely formal semantics, inference rules, temporal structure, tool support for validation, and formal validation type.

The axis **formal semantics (V1)** tells whether the classification object provides a (mathematically defined) formal semantics for policy specification. Such a semantics may be based on sets, relations, Kripke frames (in the case of a modal logic⁷) and/or other mathematical theories. The existence of such a semantics is a necessary precondition for formal validation since it excludes any kind of ambiguities in policy specification. The axis has the two possible values *existent* and *missing*. It should be noted that the value of this axis can be derived from the value of M2. However, we still keep it here as a separate axis because of its importance for policy validation.

A further axis is called **inference rules (V2)**. It indicates the existence of inference rules which allow to derive conclusions from given specifications respectively authorization constraints syntactically. Such inference rules are a necessary precondition for deduction-based validation (refer to V5). Common examples of formalisms having such inference rules are propositional, first-order, and higher-order logic. As above, this axis has the two possible values *existent* and *missing*.

Another axis refers to the **temporal structure (V3)** of the classification object. It may have values such as *none*, *explicit*, *implicit* and *implicit/explicit extension possible*. If the necessity occurs, this axis may be extended by further values. A formalism like LTL will be classified as having an explicit temporal structure whereas a graph-based formalism including graph transformations (compare DTAC [36, 38, 66]) may be regarded as having an implicit temporal structure. A graph-based formalism without graph transformations may be mapped to the value *implicit extension possible*. Other formalisms can be classified analogously. A (preferably explicit) temporal structure enables the specification of authorization constraints such as history-based separation of duty or various authorization constraints for workflows.

The **tool support for validation (V4)** is naturally an important topic. The values of this axis may be *none*, *names* of single tools, *collections* of several tools, etc. The tools based on a formal semantics may be subdivided in the ones which support deduction-based validation like the theorem prover Isabelle/HOL [55, 52] and those which have been designed for automated validation like the model checker NuSMV [21, 43] (refer to V5). There may also be tools that support both

⁷ Linear temporal logic (LTL) seems to be quite adequate for validation [25].

types of validation. Finally, one may find tools such as the USE tool [63] whose input format has a formal syntax but no formal semantics. Tool support helps to reduce the probability of human mistakes with respect to the validation of security requirements drastically.

The **formal validation type (V5)** is a very important part of the classification in the group validation. Currently, it seems sufficient to have the following values: *none*, *deduction/ proof-based*, and *automated/ configuration-based*. Obviously, future extensions are possible. Deduction-based or proof-based validation [35] requires the existence of inference rules and can be supported by tools such as the theorem prover Isabelle/HOL. Most of the times, the necessary proofs for this type of validation cannot be generated without human intervention. Configuration-based or automated validation [35] can only be applied to problems with a finite scope or, more precisely, a finite amount of possible solutions, which can be checked by some automated algorithm. This type of validation is essentially independent of human intervention. On the other hand, complexity generally is a significant issue for automated validation, i.e. depending on the tool in question there will be a limit to the size of scope beyond which the necessary time for computation will reach an intolerable extent. The value *none* is reserved for classification objects without formal semantics since the existence of such a semantics is a necessary precondition for formal validation (see axis ‘formal semantics’).

Administration Administration of access control systems is a critical task since improper administration may cause severe security breaches. Therefore suitable administration models and methods that help administrators to do the right job have to be found and applied. These models and methods need to be derived from the security requirements placed on administration. In this group we distinguish between axes concerning user interfaces (A1 - A2) and axes concerning administration models (A3 - A5).

The **usability (A1)** of a policy administration interface depends mainly on the ability of human users to understand and comprehend a given policy. Since a policy must be also machine-readable in order to be enforced by the underlying IT system, the policy needs to be represented in a formal syntax. However, a formal syntax may be complex and hard to read for a human user; thus, security administrators may have problems with maintaining the policy. Having different policy representations, one designed for policy administration and one designed for its enforcement, may solve the problem to some extent. It can simplify proper administration and help avoiding errors. Note that for this approach a lossless translation between these representations has to be provided. A value of this axis comprises *machine-readable policies*, *human understandable policies*, *notification of actually relevant access rights*, and *preview of relevant access rights after administrative interventions*.

The **type of an administration user interface (A2)** may be a graphical administration tool or a command line tool which can be based, for example, on some proprietary policy specification language. Note that in case of using

common policy languages such as Ponder or XACML, it may be possible to use common existing administration tools. A value of this axis lists the available *graphical administration tools* as well as available *command line languages and tools*, if any. Whether the usage of *common existing languages and tools* is possible or not should also be taken into account.

A **decentralized administration architecture (A3)** provides flexibility, manageability, and scalability in complex and distributed systems. A value of this axis is a description of whether and how decentralized administration is achieved. In particular a classification should be aware of the following aspects. Firstly, *authentication and identification of administrators* prevents misuse of administration rights. Secondly, *administration domains* are based on the same intention and limit the scope of administration rights. Thereby staff members having different administration rights can be separated through different domains. Furthermore, they may not be able to change their own access rights. Finally, features necessary for scalability are *policy splitting and composition* and as a special case *simultaneous administration*.

A value of the axis **separation of organization and administration (A4)** states whether and how the authorization management is isolated from the actual authorization. For example, it is possible to use the same means as for authorization to confine administration rights. In this case there is no dedicated administration model.

The axis **dynamic policy changes (A5)** takes into account whether there is a mechanism to maintain and change the policy at runtime as well as if there is an event-controlled mechanism that may dynamically trigger access right changes.

Integration This group of axes is related to the architecture of classification objects and particularly to their integration with other services of the environment. The axes are illustrated by Figure 1.

One axis we consider is the **modularity of the core authorization service (I1)**. This core may include well-separated, modularly designed components such as *policy decision points (PDP)*, *policy enforcement points (PEP)*, *a policy repository*, an *administration agent*, and a *validation tool*. A value of this axis is a description of the components that can be identified within the core of the authorization service and how these components are organized. Note that a modularly designed architecture is a prerequisite for different core authorization service components being distributed as well as for their seamless integration with other services. E.g. while a PEP may be part of a WFMS, a PDP may be realized externally through a dedicated authorization server.

A further aspect of integration is the **association of authorization supporting services (I2)** assisting the core authorization service by providing external information such as context information where access decisions may depend on. Examples are a *directory service*, a *time service*, *temperature sensors*, a *history database*, or a *WFMS* providing information about business processes and their tasks. It should be noted that one might argue whether a policy repository is regarded as an authorization supporting service or as a core service. A

value of this axis is a set of such authorization supporting services including a specification of their provided service.

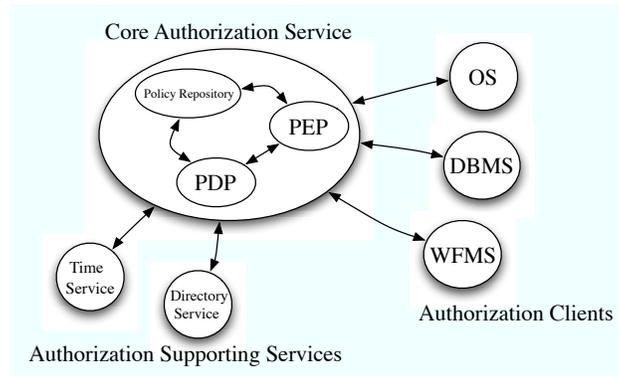


Fig. 1. Integration of Authorization Service.

A value of the axis **communication (I3)** is a description of all interfaces and protocols that are used throughout the whole architecture together with the information about which entities are communicating through them. There are four aspects worth mentioning regarding the values of this axis. Firstly, a classification according to this axis should take into account the *layering of communication relationships*. While at the top layer there may be an application specific protocol, at the bottom layer it has to be decided how raw data should be exchanged. Secondly, a classification should distinguish whether or not communication relationships can be *distributed over a network*. Thirdly, for the type of protocol or interface used at top layer of a communication relationship we distinguish between two cases. We speak of *loose coupling* if there is a string based protocol making few assumptions on communication partners. In contrast, we speak of *tight coupling* if there is a middleware interface such as CORBA or a Java API (using, for example, remote method invocation) or a library level interface such as a dedicated Perl API. Finally, during classification one should be particularly aware of *security features of protocols* such as SSL.

Miscellaneous This group contains axes regarding general aspects of classification objects. It is also the default group for axes that do not fit in any other group.

By the **maturity (G1)** of an implementation we mean the extent to which an implementation has reached an advanced stage of development. Maturity may imply a certain degree of stability and quality. We distinguish three different cases to form a classification axis regarding maturity. Note that this axis is only

applicable to implementations. Firstly, the fact that an implementation is declared as a *prototype* implies a low maturity that is, for example, not sufficient for the use in a productive environment. This can be the case for a proof-of-concept implementation coming out of a scientific project. Secondly, the maturity of *commercial products* varies depending on whether it is a *well-established* product or whether it is *new on the market*. The same is true for *non profit open source products*. Note that in these cases it can sometimes take a long time to achieve an advanced stage of development, i.e. a high level of maturity. Finally, the last value we add to this axis has more to do with the quality of an implementation than with its maturity. If an implementation has a *level of assurance* certified through a trusted third party, there is a piece of evidence for having a certain degree of quality. This implies in most cases a certain degree of maturity. For example, a software may have a *level of assurance* according to the Common Criteria.

The **market share (G2)** is an axis with a value from 0 to 1 reflecting the spread and the popularity of an implementation. However, in many cases an actual classification would give an estimated, vague market share value rather than a precise number. Although they are not distributed as products on a market, this axis can also be applied to access control models in the sense of popularity and spread. Together with its maturity, the market share may give a hint on the quality of an implementation.

4 Application of the Framework

In this section we apply our framework to three classification objects—namely X-GTRBAC [17], BEA WebLogic Server [10], and Adage [71]. Afterwards we show how to use the framework for direct comparison of different objects and provide a method to apply the framework for the evaluation of the classified objects according to a set of requirements.

During the analysis of classification objects we found that the three mentioned classification objects provide us with a rich set of applicable axes covering a broad range of different values. This is the reason why we chose these three classification objects.

We use continuous text for the classification because it is readable and does not take much space. Furthermore, we strive for keeping the classification short and concise even though we classify the objects with respect to (almost) all axes, i.e. the axes that are applicable within the scope of this paper. To carry out a fully elaborated and unabridged classification, we recommend to use a database so that comparisons of different objects are easy to accomplish and the classification values from different classification objects can be related to each other according to axes definitions.

We provide axis identifiers, such as *MI*, in line with each assigned classification value in order to refer to the appropriate axis the value belongs to.

4.1 Classification of X-GTRBAC

As a first example, we consider X-GTRBAC developed by Bhatti et al. [17] because it covers various classification aspects. In addition, it is a relatively new access control framework/architecture based upon XML. X-GTRBAC is a prototypical implementation (*M1*) of GTRBAC [39] which extends the ANSI RBAC standard (*M5*) with elaborated temporal constraints and hierarchies.

X-GTRBAC can be integrated in web service environments (*M3*). Thus, enforcement can be carried out at the middleware layer, which means that we have a reference monitor at resource level (*E1*). Due to the web service integration, X-GTRBAC supports the enforcement of policies in decentralized environments (*E2*) and remote decision and enforcement (*E3*).

X-GTRBAC provides a graphical user interface for administration (*A2*). A decentralized administration architecture has also been introduced and implemented [19] (*A3*). Since XML is used as the policy specification format, X-GTRBAC supports machine-readable policies (*A1*). Moreover, X-GTRBAC provides an XML policy repository (*I1*, *I2*).

Concerning policy specification, mainly temporal access control rules and simple static and dynamic separation of duty can be expressed (*P2*). As a consequence, various types of temporal authorization constraints (e.g., on role activation, permission and user assignment, role hierarchies) and mutually exclusive roles are supported (*P3*). In addition, X-GTRBAC supports context constraints with respect to location [17]. Hence, context information with respect to time and location must be provided (*I2*). Consequently, X-GTRBAC is well-suited for organizations where context constraints are needed such as in hospitals or large companies (*M3*). Although temporal authorization constraints are useful in workflow environments, specific support for workflow access control policies does not exist (*P4*, *E4*). Due to the fact that XML is employed for policy specification, policies are represented by a policy specification language (*L2*).

X-GTRBAC and GTRBAC provide a formal syntax and semantics (*L3*, *V1*) and an explicit temporal structure for periodic time expressions based on calendars (*V3*) [39]. Inference rules do not exist (*V2*). Algorithms for the automated validation of GTRBAC policies exist and have been realized in a system for supporting GTRBAC constraints on top of a DBMS [39]. Thus X-GTRBAC provides tool support for automated validation (*V4*, *V5*) [17].

4.2 Classification of BEA Weblogic Server

BEA WebLogic Server is a *mature, commercial (G1)* J2EE application server (*M1*) that builds a foundation for applications in a *web service environment (M3, M4)* [10]. It also includes an authorization mechanism to protect its resources. According to Gartner's market share application integration and middleware worldwide 2005 report, which was published in June 2006, BEA WebLogic Server ranked number one with a market share of *0.338 (G2)* [8, 7].

The authorization engine of WebLogic Server enforces its decisions *at the resource level (E1)*. It is possible to distribute service delivery and particularly access decision enforcement amongst multiple servers that all belong to one unified

security domain [10] (*E2*). In particular, policy decisions and policy enforcement can be delegated by means of so-called security providers to a remote server that is integrated in the so-called WebLogic security framework (*E3*). For the time being, there is no specific workflow support in WebLogic Server (*E4*).

The default authorization engine uses XACML as its only policy representation (*L2*). In particular a subset of the XACML RBAC profile [4] is supported [9, p.7-5]. This profile is based on the ANSI RBAC standard (*M5*) but has some drawbacks regarding separation of duty and role hierarchies. BEA WebLogic Server inherits the expressiveness and flexibility of XACML and is able to enforce separation of duty constraints (via role conditions), cardinality constraints, context constraints, and time constraints [9, p.5-3 et seq.] (*P3*). Using such constraints, organizational control principles such as separation of duty, prerequisites, and obligation [47, p.9] can be expressed (*P2*). Delegation (*P2*), however, is not yet a part of the XACML standard but is supposed to arrive with the next major version upgrade XACML 3.0 [53, p.7]. Even though XACML can be used to express workflow related concepts, these capabilities are not applicable to BEA WebLogic Server (*P4*, *E4*). BEA policies specify security requirements at the resource level (*P1*), which can be, for example, Common Object Model resources, Enterprise JavaBean resources, Java Database Connectivity resources, web application resources, and web service resources [11, p.3-15 et seq.].

However, as with XACML in general, there is no mathematical model where authorization policies are based on. Even though there are approaches to retrofit XACML with a formal semantics [32, 46, 68] (*V1*) and appropriate validation tools such as Margrave [29] and RW [68] (*V4*), validation and a formal foundation have never been a concern during the design of XACML. Therefore formal policy validation methods are applicable in this product only to some extent. In particular there is no approach providing inference rules (*V2*) or a temporal structure (*V3*).

There is a graphical administration console implemented as a web application including the configuration of the WebLogic security framework and particularly the configuration of the authorization engine with an XACML policy (*A2*). The administration architecture is decentralized in that there are well separated administrative domains which can be locked while making configuration changes in order to prevent inconsistencies resulting from simultaneous administration [10] (*A3*). The administration itself is not confined by the authorization policy. It is clearly separated from the actual rights management (*A4*).

BEA WebLogic Server is a highly modular system. Its security services are distributed amongst several logical entities, so-called security providers, which can in turn be distributed physically amongst several servers (*I1*, *I2*). All security services are accessible internally within WebLogic Server through a unified Java interface (*I3*), using the aforementioned WebLogic security framework. A client can access an application running on WebLogic Server through web (*I3*). All connections—internal between servers and external from a client to a server—can be secured using SSL (*I3*).

4.3 Classification of Adage

The Authorization toolkit for Distributed Applications and Groups (Adage) [71] was a project at The Open Group Research Institute that came up with a new model and a set of tools (*M1*) for authorization in distributed environments (*I1*). Adage is special because it is based on several overarching design principles: user-centered design (*A1*), policy-neutral design (*P2*), and role-based design (*M5*).

Adage provides a graphical user interface as well as an interface based on Adage's own policy language called authorization language (AL) (*A2, L2*). The Adage authorization clients (*A3*) communicate with the authorization server through an API (*I3*). The authorization server acts as a reference monitor (*E1*), allowing distributed policy enforcement (*E2*) with support for both remote decisions and remote enforcement (*E3*). Adage provides a simple API for developers to communicate with Adage components for the purpose of administration, access decisions, and enforcement. The Adage architecture is highly modular (*I1*), and the various components of its core architecture communicate in a secure way (*I3*). The Adage architecture is highly flexible and can work with current and probably also with upcoming authentication and attribute services, secure communication infrastructures, administrator languages, and databases (*I2*). Adage uses two logically distinct databases to maintain persistent information for Adage policies. One is used to store policy objects defined through the AL interpreter, while the other stores a compiled version that is optimized for the evaluation by the authorization decision engine (*I1, I2*). The enforcement engine of Adage is based on a combination of roles and rules for specifying and interpreting policies (*M5*). The Adage language and engine retain the ability to express the simple tuple-based authorization relationships between subjects, actions, and objects in an access control matrix model (*P1*). Most of the advanced Adage features are modifications and enhancements to this basic model. Statements in the Adage language are interpreted as definitions and constraints (*P2*) and are evaluated by the decision engine. AL provides a formal syntax but no formal semantics [71]. Consequently, Adage is classified with the values *missing* (*V1, V2*) and *none* (*V4, V5*). There is no temporal structure available (*V3*).

Furthermore, AL can express various types of separation of duty constraints including operational separation of duty and history-based separation of duty [62].

Adage does not provide any workflow specific features (*E4, P4*).

4.4 Direct Comparison of Classification Objects

As we stated in Section 3.1, the framework cannot have a mathematical similarity measure so that direct comparisons of different classification objects (as shown in Figure 2a) have to be carried out on the semantic level. For example, to find out the different types of policy representations one has to search the occurrences of the axis identifier *L2* and to compare what is written there in each case: while BEA WebLogic Server and X-GTRBAC use XML based policy representations, Adage introduces its own policy language called AL.

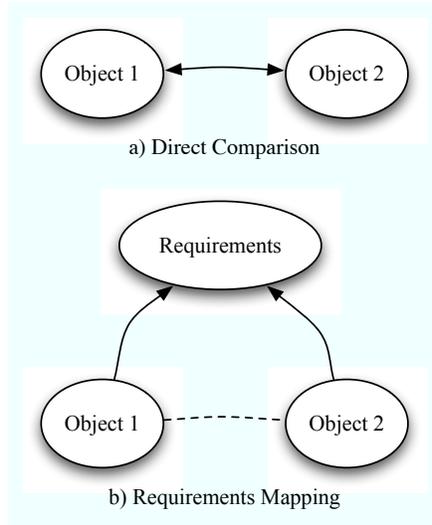


Fig. 2. Comparison and Evaluation of Classification Objects.

4.5 Using the Framework for Requirements Mapping

Say, a user is searching for a model or mechanism for a specific application. As mentioned in Section 1, by using the framework it is possible to evaluate which model or mechanism satisfies the user's requirements best. For this purpose, the user needs to view his/her requirements from the perspective of the framework, i.e. regarding the requirements as classification values. In case that a requirement cannot be expressed within the framework, the framework needs to be extended accordingly, as is described in Section 3. Note that having available the classifications for all potential models and mechanisms is a prerequisite for this application of the framework.

Regarding the requirements as classification values, they can be compared with the existing classifications of classified objects by means of a simple ranking system provided below in this subsection. By doing so, the user can find out which classification object ranks best, i.e. meets his/her requirements best. This process of requirements mapping is depicted in Figure 2b. The dashed line indicates that by providing the requirements evaluation for two objects, they can also be compared amongst each other by using the requirements as a reference.

For the ranking system we choose the simplest and most intuitive valuation function, which is based on linear weights. The reason for this choice is that the ranking system has to be used directly by the user so that an easy understandable structure is of highest importance. In this ranking system, the user needs to assign to each axis x_i a weight $\omega_{x_i} \in \mathbb{R}_0^+$ representing the importance the user would like to attach to a classification aspect. To evaluate a classification object,

a rating $m_{x_i} \in \{0, \dots, 5\}$ is assigned to each axis x_i where the user can assess to which extent the considered object meets the requirements with respect to the aspects represented by the axes. The values for the rating range from completely failing (represented by 0) up to perfectly meeting the requirement (represented by 5). The range from 0 to 5 is arbitrary and has no implication on the usage of the framework. However, it is reasonable to keep the range small, because it is difficult to find a different semantics of the values, say, 51 and 52 in a range from 0 to 100. The overall rating—i.e. the valuation function—of the classification object can then be computed by

$$\sum_{i=1}^n m_{x_i} \omega_{x_i}$$

where n is the number of axes. The higher the value of this sum the better is the ranking, i.e. the better meets the considered object the requirements of the user, presumed that he/she has chosen the weights and ratings properly. It is important to note that the ratings and weights are subjective and are based on assessment, which is due to the fact that it is again not possible to provide a mathematical representation of classification values (confer Section 3.1). Therefore results of such an evaluation have to be used with care.

We give a brief example illustrating such an application of the framework. We would like to find out which of the three classified objects from above—X-GTRBAC, BEA WebLogic Server, and Adage—meets an imaginary set of requirements best with respect to a reduced set of axes. We summarize the results within Table 1.

Table 1. Evaluation Example

	V1 ($w_{v_1} = 1$)	V3 ($w_{v_3} = 1$)	P3 ($w_{p_3} = 2$)	M5 ($w_{m_5} = 1$)	G1 ($w_{g_1} = 4$)	Val Fun
X-GTRBAC	5	5	4	5	2	31
BEA WebLogic	3	0	2	2	5	29
Adage	0	0	2	3	2	15

Firstly, we are searching for a mechanism with a formal semantics (*V1*). Secondly, a temporal structure should be available (*V3*). Thirdly, in terms of the authorization constraints (*P3*) X-GTRBAC is best for us because of its temporal and separation of duty features. Fourthly, we are searching for a model which is close to the ANSI RBAC standard (*M5*). Finally, we would like to have a mature, commercial product (*G1*). According to our choice of weights and ratings, X-GTRBAC ranks best in this comparison. We admit, it does not make much sense to consider only 5 axes, but this is just to show how it works.

5 Evaluation of the Approach

The framework presented in this paper was developed after we had to face the challenge of comparing and analyzing 19 different models and mechanisms for

access control, as part of the research project ORKA [54]. Due to the many problems in evaluating and comparing the properties of those models and mechanisms we decided to come up with a classification framework and the goals already mentioned. An argument towards the exhaustiveness of the framework is the methodology we used for its development. We took every property we encountered while analyzing the models and mechanisms to build the fundamental set of properties for the framework. Out of this set we identified six groups where we found that all properties naturally fit in. Afterwards we created the axes by trying to find similarities between properties of different models and mechanisms. This sometimes included a process of abstraction. However, we still needed to leave the framework open so that it can be accommodated according to unforeseen properties introduced by new models or mechanisms.

As shown in Section 4, we meet our goal that different classification objects can be compared with each other. It is also shown, how the classification can be used for a requirements mapping. Even though the ranking, which is done in this case, is subjective and based on assessment so that its results have to be taken with care, it is a good aid while searching for a suitable access control model or mechanism. It can particularly help to elicit a set of candidates.

One drawback of our framework is that there is no similarity measure provided through the framework that can be used for direct comparison of different objects. Instead, the classification values need to be evaluated semantically in order to make the values comparable. The reason for this is given in Section 3.1.

6 Related Work

In the past years, there have been various contributions to the area of RBAC which emerged as an alternative of classical discretionary and mandatory access control approaches. Ongoing research activities in RBAC led to the ANSI standard for RBAC [3]. While former RBAC models mainly covered database management and network operating systems, today's contributions consider aspects such as management of web services, workflow integration, ease of administration, and usability. Up to now, there are only a few contributions comparing and analyzing existing role-based approaches. Most papers propose a new model and they contain some kind of comparison within the related work section. In contrast to these contributions, our classification framework considers RBAC models and mechanisms in a holistic manner. An elaborated depiction of existing role-based models and mechanisms describing their special characteristics and applications is given by Ferraiolo et al. [28].

Other contributions provide comprehensive classifications for access control models in general. Kane and Browne [42] present a classification scheme for implemented access control mechanisms. The paper provides six classification axes and analyses the systems Akenti, dRBAC, CRISIS, Kraft-Schäfer, BitTorrent and WPA. Evaluation of theoretical models and mechanisms—which is also within the focus of our paper—is not covered. Additionally, our main focus is on role-based approaches and not on access control models in general. Bertino

et al. [14] propose a general framework for representing a large variety of access control models. The framework is based on a logical formalism and is general enough to model discretionary, mandatory, and role-based access control models. In contrast, the focus of our paper is not only on functional properties such as the expressiveness of an access control model but it also includes characteristics which may be important for practical usage, such as enforcement, validation, administration, integration and usability aspects.

7 Conclusions and Future Work

In our work we have found several important criteria for classifying advanced role-based access control models and mechanisms. These criteria allow to analyze, compare, and evaluate such models and mechanisms with respect to various requirements for validation, enforcement, and specification of policies. This helps organizations to find adequate access control mechanisms for their purposes. On the other hand, the described criteria can now be used as an orientation for the development of access control models.

For future work we identified five tasks. Firstly, we strive for the creation of an extensible database containing classifications of important and common access control models and mechanisms, which may be used by interested organizations. Secondly, having precise definitions for all possible or all actually used classification values is a prerequisite for a useful application of the framework. In particular, for the axis on types of authorization constraints $P\beta$ we could not provide such definitions within the scope of this paper. Thus we aim at providing an exhaustive database of precise value definitions for all values of discrete axes as future work. Thirdly, we see further classificatory work in case of the axis for language properties $L\beta$. In this case we strive for a more fine-grained classification in order to provide more structure to the somewhat flat axis. This could be realized through a separate framework focused on policy specification languages. Other axes may need to be refined as well. Fourthly, we strive for a visualization of classifications, for example, by means of charts or diagrams in order to make them more accessible. Finally, we will use this framework as a basis while developing a new authorization architecture in the ORKA [54] research project.

References

1. G.-J. Ahn. *The RCL 2000 language for specifying role-based authorization constraints*. PhD thesis, George Mason University, Fairfax, Virginia, 1999.
2. G.-J. Ahn and M. E. Shin. Role-Based Authorization Constraints Specification Using Object Constraint Language. *Wetice*, page 157, 2001.
3. American National Standard Institute (ANSI) for Information Technology. Role Based Access Control, ANSI INCITS 359-2004, Feb. 2004.
4. A. Anderson. *XACML Profile for Role Based Access Control (RBAC)*, 2004.

5. P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (epal 1.2), 2003. <http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/> (2007-10-01).
6. V. Atluri and J. Warner. Supporting conditional delegation in secure workflow management systems. In *SACMAT*, pages 49–58, 2005.
7. Bea Systems. Press Release: BEA Systems Seizes Top Spot in Application Servers Based on Total Software Revenue in 2005. http://uk.bea.com/news/2006/060620_gartner_application_servers.jsp (2007-10-01).
8. Bea Systems. Press Release: BEA WebLogic Server Demonstrates Once Again that Not All Application Servers Are Created Equal. http://uk.bea.com/news/2006/060919_webLogic_server.jsp (2007-10-01).
9. Bea Systems. *Bea WebLogic Server, Securing WebLogic Resources Using Roles and Policies*, 2006. <http://edocs.bea.com/wls/docs92/pdf/secwires.pdf> (2007-10-01).
10. Bea Systems. *Introduction to BEA WebLogic Server and BEA WebLogic Express*, 2006. <http://edocs.bea.com/wls/docs92/pdf/intro.pdf> (2007-10-01).
11. Bea Systems. *Understanding WebLogic Security*, 2006. <http://edocs.bea.com/wls/docs92/pdf/secintro.pdf> (2007-10-01).
12. D. E. Bell and L. LaPadula. *Secure Computer Systems: Mathematical Foundations*. 1973. ESD-TR-73-278, Bedford.
13. M. Benantar. *Access Control Systems: Security, Identity Management and Trust Models*. Springer-Verlag, New York, 2006.
14. E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A logical framework for reasoning about access control models. *ACM TISSEC*, 6(1):71–127, 2003.
15. 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.
16. 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.
17. R. Bhatti, A. Ghafoor, E. Bertino, and J. Joshi. X-GTRBAC: an XML-based policy specification framework and architecture for enterprise-wide access control. *ACM TISSEC*, 8(2):187–227, 2005.
18. R. Bhatti, J. Joshi, E. Bertino, and A. Ghafoor. Access Control in Dynamic XMLBased Web Services with X-RBAC. 2003.
19. R. Bhatti, B. Shafiq, E. Bertino, A. Ghafoor, and J. Joshi. X-GTRBAC admin: A decentralized administration model for enterprise-wide access control. *ACM TISSEC*, 8(4):388–423, 2005.
20. J. Bryans. Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35. ACM Press, 2005.
21. A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV2: an open source tool for symbolic model checking. Technical report, Jan. 01 2002.
22. D. D. Clark and D. R. Wilson. A comparison of commercial and military computer security policies. *IEEESSP*, pages 184–194, 1987.
23. J. Crampton and H. Khambhammettu. Delegation in Role-Based Access Control. In *Proc. of the ESORICS*, pages 174–191, 2006.
24. N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The Ponder Policy Specification Language. In *Proc. Policy 2001*, volume 1995 of *LNCS*, pages 17–28. Springer, 2001.
25. M. Drouineaud, M. Bortin, P. Torrini, and K. Sohr. A First Step Towards Formal Verification of Security Policy Properties for RBAC. *QSIC*, pages 60–67, 2004.

26. D. Ferraiolo and R. Kuhn. Role-Based Access Control. In *15th NIST-NCSC National Computer Security Conference*, pages 554–563, 1992.
27. D. Ferraiolo, R. Sandhu, S. Gavrila, D. Kuhn, and R. Chandramouli. Proposed NIST standard for role-based access control. *ACM TISSEC*, 4(3), Aug. 2001.
28. D. F. Ferraiolo, D. R. Kuhn, and R. Chandramouli. *Role-Based Access Control*. Computer Security Series. Artech House, Boston, 2003.
29. K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05*, pages 196–205, 2005.
30. C. Georgiadis, I. Mavridis, G. Pangalos, and R. Thomas. Flexible team-based access control using contexts. In *Proc. of the SACMAT*, pages 21–27, May 2001.
31. 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.
32. M. M. Greenberg, C. Marks, L. A. Meyerovich, and M. C. Tschantz. The Soundness and Completeness of Margrave with Respect to a Subset of XACML. Technical Report CS-05-05, 2005.
33. M. Hitchens and V. 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.
34. P. Humenn. *The Formal Semantics of XACML (draft)*. PhD thesis, 2003. <http://lists.oasis-open.org/archives/xacml/200310/msg00094.html> (2007-10-01).
35. M. Huth and M. Ryan. *Logic in Computer Science, Modelling and Reasoning about Systems*. Cambridge University Press, 2nd edition, 2004.
36. T. Jaeger and J. Tidswell. Practical safety in flexible access control models. *ACM TISSEC*, 4(2):158–190, 2001.
37. N. Jardine and R. Sibson. *Mathematical Taxonomy*. John Wiley & Sons Ltd, 1971.
38. Jonathon E. Tidswell and John M. Potter. A Dynamically Typed Access Control Model. In *Proc. of the Australasian Conf. on Inf. Sec. and Priv.* Springer, 1998.
39. 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.
40. L. Kagal. A Policy Language for the Me-Centric Project, 2002.
41. S. Kandala and R. S. Sandhu. Secure Role-Based Workflow Models. In *IFIP Workshop on Database Security*, pages 45–58, 2002.
42. K. Kane and J. C. Browne. On Classifying Access Control Implementations for Distributed Systems. In *Proc. of the SACMAT*, pages 29–38. ACM Press, 2006.
43. K.L. McMillan. The SMV system, symbolic model checking - an approach. Technical Report CMU-CS-92-131, Carnegie Mellon University, 1992.
44. M. Kohler, C. Liesegang, and A. Schaad. Classification Model for Access Control Constraints. In *Proc of the 3rd Int. WS on Inform. Assur.*, to appear in Apr. 2007.
45. W. kuang Huang and V. Atluri. SecureFlow: A Secure Web-Enabled Workflow Management System. In *ACM Workshop on Role-Based Access Control*, pages 83–94, 1999.
46. M. Mankai and L. Logrippo. Access Control Policies: Modeling and Validation. In *NOTERE*, pages 85–91, 2005.
47. T. Moses. *eXtensible Access Control Markup Language (XACML) Version 2.0*, 2005. OASIS Standard.
48. N. Damianou and N. Dulay and E. C. Lupu and M. Sloman. Ponder: a language for specifying security and management policies for distributed systems. *Imperial College Research Report DoC 2000/1*, 2000.

49. M. J. Nash and K. R. Poland. Some conundrums concerning separation of duty. In *Proc. of the IEEE Symp. on Sec. and Priv.*, pages 201–207, 1990.
50. G. Neumann and M. Strembeck. Design and Implementation of a Flexible RBAC-Service in an Object-Oriented Scripting Language. In *Proc. of the ACM CSS*. ACM Press, 2001.
51. G. Neumann and M. Strembeck. An Integrated Approach to Engineer and Enforce Context Constraints in RBAC Environments. *ACM TISSEC*, 7(3):392–427, 2004.
52. T. Nipkow, L. Paulson, and M. Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*. Springer Verlag, 2002.
53. Oasis. *XACML v3.0 Administrative Policy Version 1.0 Working Draft 16, 22 February 2007*, 2007. <http://www.oasis-open.org/committees/download.php/22559/xacml-3.0-administration-v1-wd-16.zip> (2007-10-01).
54. ORKA. Project homepage <http://www.orka-projekt.de> (2007-10-01).
55. L. Paulson and T. Nipkow. Isabelle. <http://www.cl.cam.ac.uk/research/hvg/Isabelle> (2007-10-01).
56. S. Jajodia and P. Samarati and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. pages 31–42.
57. R. Sandhu. Role hierarchies and constraints for lattice-based access control. In *ESORICS*, pages 65–79, 1996.
58. R. Sandhu. Role activation hierarchies. In *Proceedings of the third ACM workshop on Role-based access control*. ACM Press, 1998.
59. R. Sandhu, E. Coyne, H. Feinstein, and C. Youman. Role-Based Access Control Models. *IEEE Computer*, 29(2):38–47, Feb. 1996.
60. A. Schaad. *A Framework for Organisational Control Principles*. PhD thesis, University of York, UK, 2003.
61. M. E. Shin and G.-J. Ahn. UML-Based Representation of Role-Based Access Control. In *Proc. of the 9th IEEE WETICE, 2000.*, pages 195–200, 2000.
62. R. T. Simon and M. E. Zurko. Separation of Duty in Role-Based Environments. In *IEEE Computer Security Foundations Workshop*, pages 183–194, 1997.
63. K. Sohr, G.-J. Ahn, M. Gogolla, and L. Migge. Specification and validation of authorisation constraints with UML and OCL. In *Proc. of the ESORICS*, 2005.
64. K. Sohr, M. Drouineaud, and G.-J. Ahn. Formal Specification of Role-based Security Policies for Clinical Information Systems, Santa Fe, New Mexico. In *Proc. of the 20th ACM Symposium on Applied Computing*, 2005.
65. M. Strembeck. <http://wi.wu-wien.ac.at/home/mark/xoRBAC/index.html> (2007-10-01).
66. J. E. Tidswell. A graphical definition of authorization schema in the DTAC model. In *Proc. of the SACMAT*, pages 109–120, New York, May 3–4 2001. ACM Press.
67. W. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems*. B & T, 2004.
68. N. Zhang, M. Ryan, and D. P. Guelev. Synthesising verified access control systems in XACML. In *FMSE '04*, pages 56–65, 2004.
69. X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal Model and Policy Specification of Usage Control. *ACM TISSEC*, 8(4):351–387, Nov 2005.
70. Z.-L. Zhang, F. Hong, and J.-G. Liao. Modeling Chinese Wall Policy Using Colored Petri Nets. *CIT*, page 162, 2006.
71. M. Zurko, R. Simon, and T. Sanfilippo. A user-centered, modular authorization service built on an RBAC foundation. In *Proc. of the IEEE Symp. on Sec. and Priv.*, pages 57–71, Oakland, CA, May 1999. IEEE Computer Society Press.