Università degli Studi di Pisa

Dipartimento di Informatica
Dottorato di Ricerca in Informatica

Ph.D. Thesis: SSD-01/IN

Access and Usage Control in Grid

Aliaksandr Lazouski

Supervisor
Fabio Martinelli

May 16, 2011

Abstract

Grid is a computational environment where heterogeneous resources are virtualized and outsourced to multiple users across the Internet. The increasing popularity of the resources visualization is explained by the emerging suitability of such technology for automated execution of heavy parts of business and research processes. Efficient and flexible framework for the access and usage control over Grid resources is a prominent challenge.

The primary objective of this thesis is to design the novel access and usage control model providing the fine-grained and continuous control over computational Grid resources. The approach takes into account peculiarities of Grid: service-oriented architecture, long-lived interactions, heterogeneity and distribution of resources, openness and high dynamics.

We tackle the access and usage control problem in Grid by Usage CONtrol (UCON) model, which presents the continuity of control and mutability of authorization information used to make access decisions. Authorization information is formed by attributes of the resource requestor, the resource provider and the environment where the system operates. Our access and usage control model is considered on three levels of abstraction: policy, enforcement and implementation.

The policy level introduces security policies designed to specify the desired granularity of control: coarse-grained policies that manages access and usage of Grid services, and fine-grained policies that monitor the usage of underlying resources allocated for a particular Grid service instance. We introduce U-XACML and exploit POLPA policy languages to specify and formalize security policies. Next, the policy level presents attribute management models. Trust negotiations are applied to collect a set of attributes needed to produce access decisions. In case of mutable attributes, a risk-aware access and usage control model is given to approximate the continuous control and timely acquisition of fresh attribute values.

The enforcement level presents the architecture of the state-full reference monitor designed to enforce security policies on coarse- and fine-grained levels of control.

The implementation level presents a proof-of-concept realization of our access and usage control model in Globus Toolkit, the most widely used middleware to setup computational Grids.

Acknowledgments

This work would be impossible to accomplish without support and contribution of many people. First of all, I thank my supervisor Dr. Fabio Martinelli for giving me this opportunity to work in the field of computer security, for his support and many valuable advises. Our numerous discussions, his continuous support and endless scientific enthusiasm that inspired me many times, were crucial for the completion of this thesis.

Further I would like to thank the whole security group at the CNR, IIT for a very nice scientific atmosphere. Special thanks to Dr. Paolo Mori who helps me a lot for my work.

I express my appreciation to Prof. Pierpaolo Degano and Prof. Francesco Pegoraro for a very well organized G. Galilei graduate school and for their kind readiness to be helpful every time when the support was needed.

Special thanks go to Prof. Pierpaolo Degano and Prof. GianLuigi Ferrari for their feedback as internal reviewers of this thesis, and to Prof. Sandro Etale, Eindhoven Technical University and University of Twente, and Prof. Javier Lopez, University of Malaga, for their feedback as external reviewers.

I would also like to thank Dr. Hristo Koshutanski, University of Malaga, for the joint work and for encouraging me many times.

Many thanks to EU-FP7-ICT NESSoS project for partly supporting my work.

Contents

Chapter 1

Introduction

Grid is designed to execute programs on shared computational resources. This makes unsecured Grid a potentially fertile ground for attacks on machines donating these resources. This thesis explores advances in access and usage control to protect computational Grids. It investigates security mechanisms to continuously monitor a usage of Grid resources, to provide a sufficient granularity of control taking into account peculiarities of Grid environment and security implications of Grid administrators and resource providers.

The first section of this chapter overviews Grid computing basics and Grid security implications. The second section outlines main concepts of access and usage control. Section 1.3 focuses on open problems and challenging issues of access and usage control in Grid. Section 1.4 provides a bird's-eye view of main contributions of this thesis and lists my publications. This Chapter ends with describing the structure of this thesis (section 1.5).

1.1 Grid Computing

Web was originally designed to connect endhosts users. All data and applications were stored and processed in these endhosts, and a network simply provided a transit to share the data. But this scenario is no longer a case. Web becomes a computer. Data is processed and applications are executed in the 'virtual' computer.

Grid is a technology which allows a seamless sharing, integration and usage of underutilized computing resources of endhosts and makes these resources available through Web. The most common resources donated to Grid are CPU cycles, memory, storage and networking. Outsourced resources to Grid can be exploited by users to solve heavy computational tasks taking place in advanced scientific and engineering applications [39]. The primary way to exploit Grid resources is to split these tasks in separate parts and each part can be executed in parallel on different machines involved in Grid. This allows to minimize the execution time of the application and to occupy efficiently underutilized resources of Grid participants. Grid

already has been successfully exploited for execution of real-world computationally intensive applications with a very large scale simulations, e.g. earthquake simulation [107], climate study [88].

Another important Grid computing feature is the ability to facilitate collaborations among a wider audience. Grid users can dynamically organize a virtual organization (VO). The VO is a set of individuals and institutions, companies, universities, research centers, industries, etc., engaged in sharing resources to tackle the problems that require the combined efforts [37] (see Figure 1.1). Grid users can be members of several real and virtual organizations.

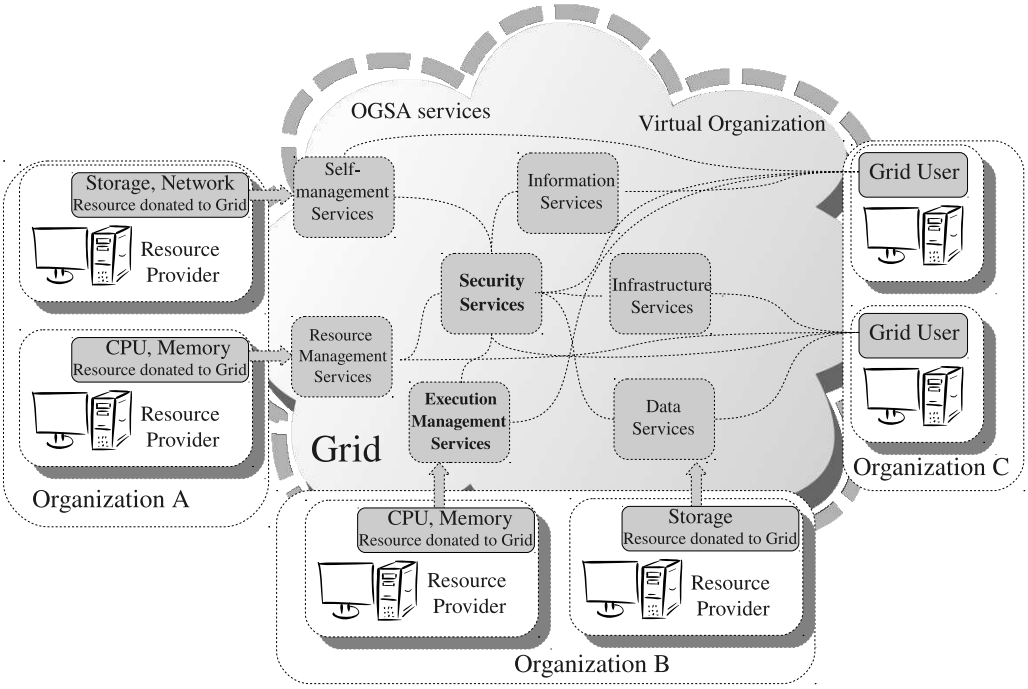

Figure 1.1: Grid Computing

Grid Design

Grid represents hardware and software components used to interconnect different Grid computers to achieve their collaborative goals. Users and resource providers should install Grid middleware on their machines to enroll in Grid and to use and donate Grid resources. As Grid grows, a degree of planning and management of Grid resources is required.

Several industrial and research organizations like OGF, IETF, W3C and OASIS are involved in planning and specifying Grid computing. OGF formulated capabilities and requirements for Grid by presenting Open Grid Service Architecture (OGSA) [41], a set of technical specifications that align Grid technologies with Web service technologies. A basic premise of OGSA is that resources donated to Grid,

i.e. processor cycles, storage, memory, network, and so on, can be combined and exposed to Grid users as services. Grid services are accessed through a set of interfaces (sometimes referred as portTypes) to exploit their functionality.

From architectural point of view, Grid is a generic infrastructural fabric managing a number of Grid services (see Figure 1.1). OGSA specifies core Grid services which must be presented to have a fully functional Grid environment. They are [41]:

- *Infrastructure services* support naming and registration in Grid. Also their manage technical specifications to deploy and built Grid services. In fact, OGSA advises to exploit Web-Services foundations for Grid Services, such as WSDL to describe service interfaces, and SOAP as the primary message exchange format for Grid services;

- ***Execution management services*** give to Grid users a possibility to submit and execute computational jobs in Grid on user's behalf. These services also address resources provisioning and lifetime management for scheduled and running jobs;

- *Data services* concern with management, access and usage of data (storage) resources along with the transfer of data between resources;

- *Resource management services* keep the track of all resources available in Grid and allocate resources for execution management services;

- ***Security services*** facilitate the enforcement of security policies within Grid. They also provide authentication, identification, credentials and trust management, authorization, audit and secure logging of security-relevant events in Grid;

- *Self-management services* accelerate operating in Grid by reducing cost and complexity of service level agreements, adopting dynamically to changes in Grid, and tuning the Grid infrastructure to achieve the best efficiency;

- *Information services* hold information about services offered to Grid users, and are responsible for services registration and discovery.

Grid Software

There are several initiatives which implement the functionality of core Grid services specified by OGSA, such as *Globus Toolkit* (GT) [40, 38, 2], Gridbus [17], Legion [29], WebOS [113], Avaki, DataSynapse, Entropia [37].

Indeed, GT is the most notable and used middleware to set up Grid. It fully corresponds to OGSA standards and provides a set of core Grid services as well as tools to develop and deploy new Grid services with additional functionality. The main GT components are Grid Security Infrastructure (GSI) (implements OGSA security

services), Grid Resource Allocation Manager (GRAM) (implements OGSA execution management services), Monitoring and Discovery service (MDS) (implements OGSA resource management and information services), Grid File Transfer Protocol (GridFTP) (implements OGSA data services), and Grid services developer's kit.

Grid Models

Grid models vary based on several factors, particularly *topology* and *problems Grid is designed to solve*.

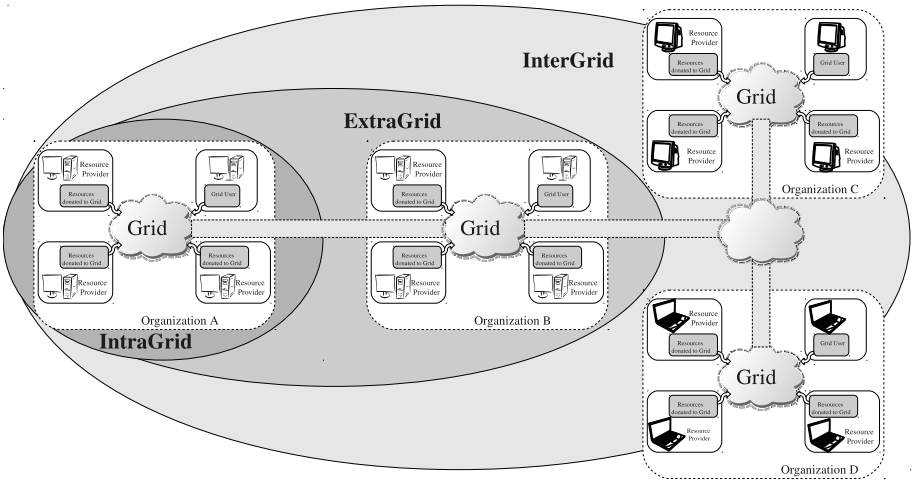

Figure 1.2: Grid Topology[37]

Grid can be built in all sizes ranging from just a few machines in a single organization to billions of desktop and mobile computers distributed over the world and collaborating together in Grid. Based on *topology*, Grid models can be divided on (see Figure 1.2) [37]:

- *IntraGrid* is an appropriate solution for organizations which want to utilize their resources efficiently by scaling internal jobs. It is composed by several machines within the organization providing a basic set of core Grid services. The primary characteristics of intraGrid are a single security provider, the number of Grid participants is permanent, and all participants have almost the same software and hardware components;

- *ExtraGrid* relaxes single organization constrains, and bridges together two or more intraGrids. The primary characteristics of extraGrid are multiple organizations forming Grid, available Grid resources change dynamically, and several security providers exist (e.g. one per organization). The number of users is fixed, and collaborations inside extraGrid assume pre-existing trust relationships between participating organizations;

- *InterGrid* combines dynamically all interested parties (i.e. organizations and independent endhosts) to collaborate. The primary characteristics of interGrid are heterogeneity of resources donated to Grid, openness since each participant can join and leave Grid freely, and existence of several security providers without pre-existing trust relationships between them.

Besides the topology, there are different business *problems Grid is designed to solve*. Some Grids are designed to take advantages of extra processing resources, whereas some Grids are designed to support collaborations between various organizations. Depending on which core Grid services are more developed and prevail, Grid models are divided on [37]:

- *Data Grid* focuses on providing secure access to distributed, heterogeneous pools of data (storage). The data Grid federates several databases and functions like a single virtual database;

- *Computational Grid* expands abilities and maximize the utilization of existing distributed computational resources (mainly CPU cycles, memory and network) through aggregation and sharing. A well-known example of a computational Grid is SETI@home used to analyze noisy radio transmissions received from outer space. This type of Grid is primarily comprised of low powered computers with a minimal storage capacity. Also the peculiarity of computational Grids is *long-lived* services.

Using Computational Grid

Secure computational Grids are main concern of this thesis, and this subsection gives a typical usage scenario of a computational Grid. In this scenario, a Grid user wants to execute a computational job on the "best" Grid computer, to monitor the running job, and to store the results afterwards somewhere in Grid.

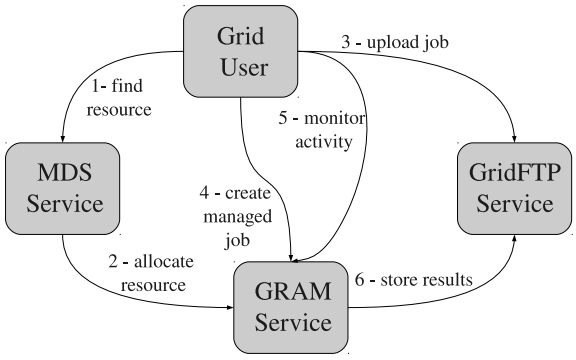

Figure 1.3: Computational Grid Usage Scenario

The Grid user should perform some steps to successfully fulfil this scenario. To begin, a new Grid user should enroll into Grid, obtain Grid-user-ID, and generate

Proxy Certificate (temporal Grid-user-ID used for delegation of authority). Once this is accomplished, the Grid user may request to find computational resources where the computational job can be executed (step 1 in Figure 1.3). The choice is made by the MDS service according to some metrics specified by the Grid user, e.g. "the quickest execution time", "the cheapest rate", etc.

The Grid user (or the MDS service on the user's behalf) queries the GRAM service (selected in the previous step as the job executor) to determine availability of resources. If this is confirmed, the Grid user requests the resources allocation (or reservation) for the job execution (step 2).

Then, the Grid user should upload the job code and input data to Grid. This is done by means of the GridFTP service (step 3). Invocation of the `createManagedJob` interface of the GRAM service allows to start the job execution (step 4). It is worth to notice, that jobs submitted for execution in Grid are usually *long-lived* activities which might last hours or even days. The Grid user might be interested in monitoring the execution status, e.g. to obtain preliminary results or terminate jobs which behave erroneously (step 5). Actually, such checks do not require that the Grid user is permanently attached to the GRAM service instance which executes the job. In fact, the Grid user is able to disconnect form the job and then at a later time and possibly from a different location reattach to it and monitor the execution process.

When the execution is over and it completes successfully, the Grid user is notified and the results of the execution can be stored in Grid or downloaded to the Grid user machine (step 6). This ends the computational Grid usage scenario.

In fact, this usage scenario of the computational Grid shows that the remote job execution is not a single-step action, but a composition of several core Grid and more important *long-lived* services. More comprehensive usage scenarios might imply an execution of a job that requires resources at multiple sites, advanced scheduling and reservation of resources, sophisticated job control options, etc.

1.1.1 Grid Security

There are many benefits of exploiting underutilized resources setting up the Grid environment. However, advantages of Grid computing are valuable if security implications of Grid users, administrator and resource providers are treated adequately.

Grid user refers to the process attempting to access Grid resources. *Grid administrator* is a Grid-aware entity responsible for the overall functioning of Grid. *Grid resource provider* donates actual computational resources to Grid and host Grid services.

Grid User Security Implications

A Grid user security might be compromised by stealing user's confidential data submitted along with a job in Grid, as well as the job itself and results of the

execution. *Secure interactions* and *trust relationships* between Grid users and Grid resources (services) might assure that Grid is trusted to keep user's data secret.

Another security implication arises from the fact that Grid users usually delegate some authority to Grid services to operate on the user's behalf (e.g. in the step 2 in Figure 1.3, the MDS service is allowed to allocate resources on the user's behalf). The least-privilege *delegation of access rights* model is required to promote the efficient collaborations within Grid from one side and prevent abuse of authority from another.

Grid users might also concern with a reliable *enforcement of user's access policies* over resources created in Grid. For instance, the running job created by the Grid user may be accessed and monitored by other parties. In this case, the Grid user decides who is granted to access the running job while the enforcement of the access decision is performed at the Grid resource provider site.

Grid Administrator Security Implications

The most fundamental security issue addressed by a Grid administrator is providing an infrastructure for *identity and credentials management*. A Grid-wide identity should be assigned for each Grid participant (e.g. Grid users and Grid resource providers) and this identity should be verifiable by other entities involved into collaborations.

The Grid administrator manages also the *access and usage control* of all Grid resources (services). This includes setting permissions for Grid users to access Grid services as well as tracking the resource usage and implementing a corresponding accounting system. Grid-central access policies impose a better coordination of Grid resources. For example, the security policy might require that the GridFTP service and the GRAM service, participating in the job execution scenario, should locate in the same security domain, i.e. hosted by the same resource provider. Another security policy might set a quota of jobs a particular Grid user is allowed to submit for execution in Grid.

Grid-central access policies might base on credentials issued by Grid-level Certification Authorities and by local security authorities to which the Grid entities belong. The Grid administrator should facilitate *interoperability between different security mechanisms* and metrics of Grid participants.

A decision on granting access to a Grid service is performed by Grid-central security services while the enforcement of this decision is a duty of the resource provider who serves as the last point of control. The Grid administrator needs to establish *trust relationships with resource providers* to guarantee that access decisions are enforced correctly.

Grid Resource Provider Security Implications

Grid resource providers should be able to *authenticate requests from Grid users* and *enforce security policies of different stockholders* regarding the requested resource. Native security policies of the resource provider might base on contractual agreements with Grid (e.g. a quota and type of donated resources and how Grid is allowed to use them) and as well as on bilateral *trust relationships between the resource provider and the Grid user* requested the resource.

For computational Grids, the resource provider should address what can be expected of jobs submitted by Grid users. The best approach is to implement a protective sandbox around the submitted job so that it cannot cause any disruption to the donating machine. In fact, the resource provider should be able to *terminate jobs behaving abnormally*. The job is determined as abnormal if it consumes more resources that expected or allowed.

Grid Security Approach

Grid security Infrastructure (GSI) is the initial approach which addresses the security implications of Grid users, administrators and resource providers. GSI encompasses different security services and provides an infrastructure that supports *secure interactions*, *identity and credentials management*, *delegation of access rights*, and *access control* in Grid environment. GSI is implemented in GT and its approaches are based on existing security standards like X.509 certificates, TLS, SAML, etc.

GSI *secure interactions* model implements WS-Security and WS-Secure Conversation specifications and provides confidentiality and integrity protection mechanisms for SOAP-messages (SOAP is the message protocol used for communications between Grid entities).

GSI supports *identity and credentials management* and *delegation of access rights* through the use of X.509 Certificates and public key encryption. X.509 End Entity Certificate (X.509 EEC) provides each Grid entity with a unique distinguished name issued by the Grid-central Certification Authority. GSI supports delegation of access rights through the use of X.905 Proxy Certificates. The X.509 Proxy Certificate is short-term and substitutes the long-term X.509 EEC to delegate access rights temporally to another Grid entity.

The OGSA security model enhances GSI and introduces new approaches to Grid security. This model proposes to outsource security functionality to specialized *Grid security services* which should support well-defined interfaces and protocols and to fit security needs of all Grid participants. OGSA specifies following security services (see Figure 1.4):

- *Credential Validation Service* evaluates security credentials used for identity management, delegation of access rights, and access control;
- *Attribute Service* manages Grid-central credentials of Grid participants;

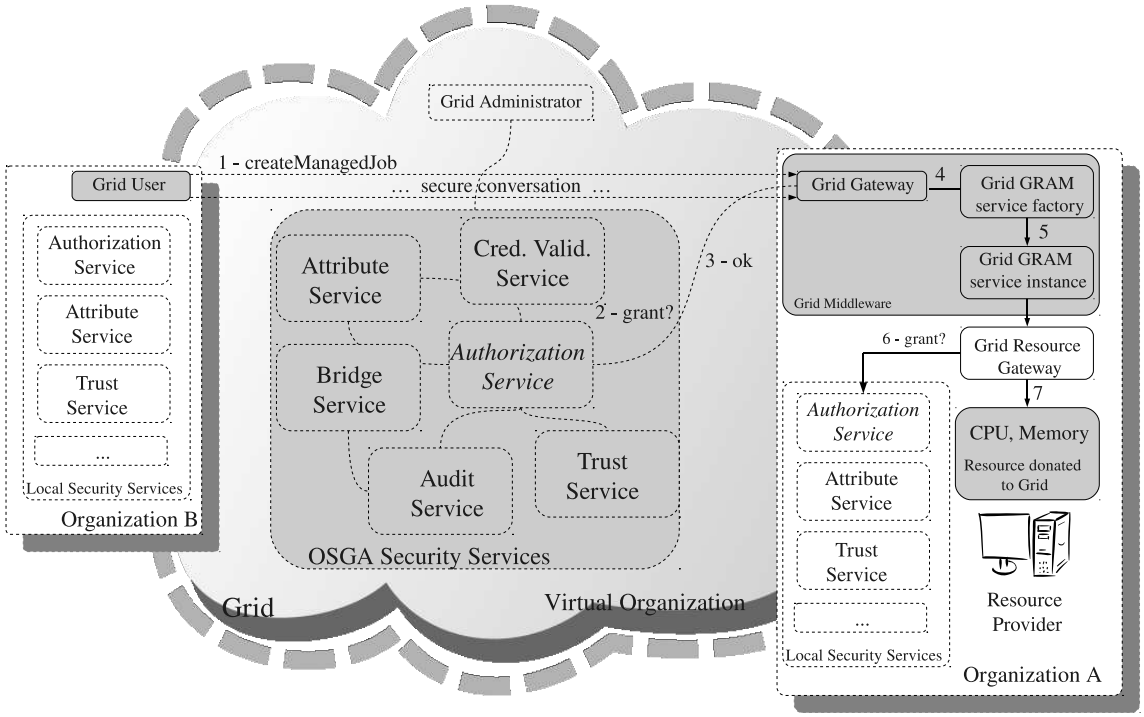

Figure 1.4: Grid Security Overview

- *Trust Service* serves as a Grid-central Certification Authority and issues security credentials (e.g. X.509 EEC) to Grid participants. Also, this service establishes *trust relationships* with native authorities of Grid users and resource providers;

- ***Authorization Service*** evaluates Grid-central access and usage policies and answers whether the Grid user is allowed to access the specified Grid resource (service). It is expected that the resource provider which hosts the accessing Grid resource (service) will enforce the access decision of the Grid authorization service;

- *Bridge/Translation Service* facilitates interoperability between different security mechanisms of Grid participants. For instance, this service transforms security credentials issued in the Grid user security domain to the Grid-central credentials. Then, the converted credential can be used by the authorization service to evaluate the Grid-central access policy;

- *Audit Service* tracks security-relevant events in Grid environment (e.g. enrolling to Grid, submitting a job, etc.).

Grid Access Control Approach

This subsection explains how a Grid user is allowed to access Grid resources. Imagine the simplified usage scenario, where the Grid user requests the GRAM service to execute a computational job. Grid receives the request and processes it by forwarding to Grid Gateway (step 1 in Figure 1.4). Grid Gateway is a part of Grid middleware running of the resource provider machine hosting the GRAM service. Grid Gateway is the trusted process by the resource provider and is allowed to handle all communications with entire Grid.

After authentication of the received request, Grid Gateway contacts Grid security services to establish whether the Grid user is allowed to use Grid resources (services) hosted on the resource provider machine. In fact, Grid Gateway asks Grid Authorization Service to produce the access decision (step 2). Grid Authorization Service manages all Grid-central and other stakeholders security policies, and interacts with other Grid security services (e.g. Credential Validation Service, Trust Service, etc.) to produce a credible access decision. When the access decision is ready, it is sent back to Grid Gateway (step 3). This level of control is called ***coarse-grained***.

If the access decision is positive, Grid Gateway proceeds the Grid user initial request by invoking the Grid service factory and creating the actual service instance which will execute the submitted job (step 4-5). This service instance is the process attempting to run on the resource provider machine, thus, it should be also authorized by the local security manager called Grid Resource Gateway. Grid Resource Gateway is the last point of control which enforces local security policies and usage agreements with Grid on donated resources (step 6). The local policies might be a Grid user-centric, but the common practice is that the local policies are more restrictive then the Grid-central security policies. If local policies are satisfied, Grid Resource Gateway allows to access the resources by the service instance (step 7). This level of control is called ***fine-grained***.

The very initial approach of access control proposed by GSI and implemented in GT is called *GridMap Authorizarion*. Grid Gateway and Grid Resource Gateway as well as Grid-central and local security policies coincide accordingly in this authorization scenario. There is only one point of control and security policy to enforce. The Grid administrator adds entries in the security policy called *grid-map-file* which maps Grid users IDs to specific local users IDs on the resource provider machine. When a Grid user requests a Grid service, a resource gateway checks if there is in a grid-map-file a local user ID counterpart for the requesting Grid user. If it is true, the resource gateway creates the Grid service instance which runs on behalf of the local user and is controlled by legacy access control mechanisms.

More sophisticated access control models for computational Grids are surveyed in Chapter 2. These models assume two-levels of control performed by both, local resource managers (i.e. Grid Resource Gateway) using legacy security mechanisms and local policies - *fine-grained control*, and by Grid-central security services (i.e.

Grid Gateway and Grid Authorization Service) - *coarse-grained control*.

Although, these approaches introduce viable solutions for intra- and extraGrid models, they are inadequate to handle security implications of computational interGrids. This will be shown in section 1.3, while the next section gives some basic definitions of access and usage control concepts.

1.2 Access and Usage Control: Main Definitions

Access control aims to assure that only trusted principals are granted to access a resource [7]. *Usage control* is in charge to guarantee that principles remain trusted also when the access is in progress, i.e. when these principles use the resource.

A *security policy* defines what is a trusted principal. The security policy is analogous to a set of laws and typically it is defined in terms of high-level rules and requirements.

Security mechanisms are access and usage control machinery that enforces security policies. These mechanisms work by intercepting each request to a resource, determining whether the request is trusted in accordance with security policies, and enforcing the access decision by executing or aborting the request. During the access, the security mechanisms work by continuously deciding whether the principal issued the request remains trusted, and if not (i.e. the security policy is violated) the security mechanisms should be able to terminate the access and release the resource. A collection of security mechanisms, sufficient to enforce security policies, is referred in the sequel as a *reference monitor*.

The idea to separate security policies and mechanisms underlies in design of access and usage control. In fact, this thesis adheres to the PEI design model proposed by R. Sandhu et al. [66] that suggests to explore access and usage control on three levels of abstraction: *policy*, *enforcement* and *implementation*.

Policy Level

The policy level introduces means *to express security policies* and mechanisms *to produce access decision*.

The first step toward expressing a security policy is to give a basic system abstraction specifying *objects* to be protected, *subjects* accessing resources, and *security-relevant events* which should be intercepted and authorized by a reference monitor.

```
tryaccess(user,GRAMservice,createManagedJob)
```

is an example of the security-relevant event, called *access request*, where the subject `user` requests the object `GRAMservice` to start execution of the subject's job in Grid by invoking `createManagedJob` portType.

Besides this basic system abstraction, auxiliary information could be required to produce the access decision. For example, in attribute-based access control [89, 104]

each access request is accompanied by a set of *attributes*. Attributes are assertions done by trusted peers about subjects and objects participating in access control. Attributes and other information about the system state used to produce access decision is named as *authorization information*.

The second step is to introduce a *policy specification language* served to express authorization information and to program security policies. The language states mainly syntax to determine which collection of symbols are correct expressions accepted by a reference monitor. The language is usually a high-level and user-friendly, application-independent, general and flexible, suitable to express a wide range of security policies and authorization information in different computational environments. XACML is the most widely-used XML-based policy specification language to express attribute-based access control [25, 119].

The next step is to provide a *formal model* of expressions written in a policy specification language. It includes a formal semantics of expressions plus a set of axioms and inference rules used to derive an *access decision* and some additional consequences. A component of a reference monitor which computes access decision is named as a *Policy Decision Point* (PDP). OGSA Authorization Service implements PDP functionality on the coarse-level of control.

PDP consumes a security-relevant event to authorize (e.g. the access request), authorization information and a security policy and returns an access decision (either `permitaccess()` or `denyaccess()`/`revokeaccess()`) and some additional information. In *access control* the decision is evaluated *only once*, while in *usage control - continuously*.

Attributes-based access control usually has a logic-based formalization [104, 7, 89, 57]. Such approaches assume that an access request, authorization information, and security policy are modeled as logical formulas. The access is permitted if the logical implication of the security policy from the access request and authorization information could be drawn. Otherwise, the access is usually immediately denied.

History-based access control (where authorization information encodes a history of all security-relevant events occurred in the system so far) often assumes automata-based formalization [18, 103, 80, 76]. An access request is presented as an input to the automaton, authorization information - as the automaton state, and a security policy serves as the automaton transition function. The access is granted if the automaton accepts the access request. Besides making the access decision, PDP also updates authorization information accordingly to the taken access decision.

Notice, that the same policy specification language might have different underlying formalizations. For example, the XACML policy language was formalized using a logic-based approach [57] as well as using process calculi [25].

The policy specification language and policy formal model formulate together *access and usage control model*. Models vary based on scenarios they are capable to express. The most popular attribute-based models are DAC [47], MAC, RBAC [100], UCON [90, 124, 92]). Our particular interest in this thesis is dedicated to **Usage CONtrol (UCON)** model introduced by R. Sandhu and J. Park. The UCON

model considers *long-lived access requests* (e.g. the GRAM service instance executing subject's job) and assumes that authorization information might change during the access execution. In fact, the *continuous control* and *mutability of attributes* are main novelties of the UCON model.

Also, the policy level might introduce some *enhancements*. The policy level enhancements address mathematical models of all security mechanisms serving to facilitate the enforcement of security policies. For example, these enhancements might describe how authorization information is collected and kept up-to-date? What supplementary information could be generated with access decision?

Enforcement Level

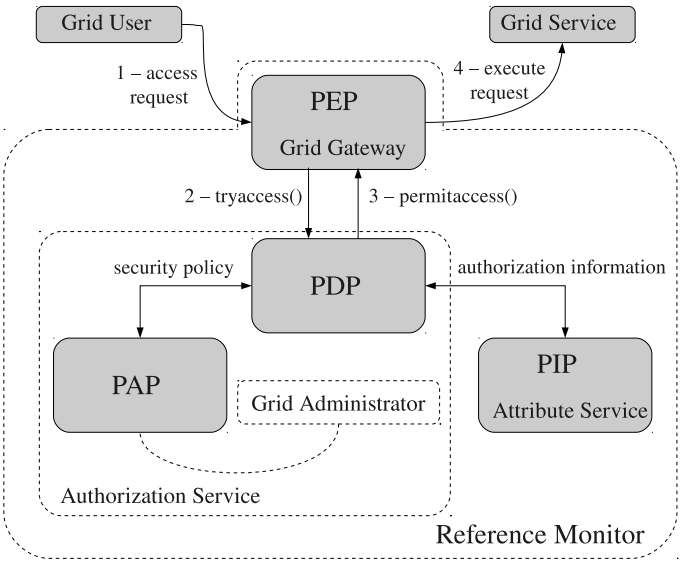

Figure 1.5: Reference Monitor Architecture

The policy level gives mathematical models of security mechanisms. The enforcement level encompasses all security mechanisms and draws an *overall architecture of a reference monitor*. The enforcement level handles a *specification of interactions between security mechanisms* and answers how these mechanisms collaborate to enforce access and usage control policies.

On the coarse-grained level of control in Grid, the architecture of the reference monitor is formed by components which corresponds to OGSA security services (see Figure 1.5).

Usual components (i.e. security mechanisms) to enforce a security policy are [114]:

- *Policy Enforcement Point (PEP)*, a component which operates as a guard giving or denying a real access to a resource;

- *PDP* (was discussed on the policy level);
- *Policy Information Point (PIP)*, a component which manages an authorization information, i.e. provides facilities for its storing, updating, retrieving, delivery to PDP;
- *Policy Administrative Point (PAD)*, a component which provides and manages security policies.

Interactions between security mechanisms presented in Figure 1.5 are the same as discussed in Grid access control approach subsection (just with a slight difference in syntax).

Implementation Level

The implementation level presents a low-level implementation of security mechanisms. It provides the cheapest and fastest solution and cares on the efficiency and performance of the reference monitor.

For instance, the access decision making is often computationally complex and heavy procedure. The implementation level could provide the optimization and minimize security overhead by caching some requests and instead of security policies reevaluation when a new request arrives - returns the cached access decision.

1.3 Problem Statement

The main purpose of this thesis is to explore security advances in protecting computational interGrids.

Identity-based vs Attribute-based Access Control

By default, access control in Grid is realized for a service-level invocation and implies an identity-based access control. Authorization information consists of Grid user identifiers (e.g. public keys), and a security policy (GridMapFile in Globus toolkit [2]) maps these identifiers to local accounts of resource providers hosting Grid services. Access control based on GridmapFile is very coarse and faces a scalability problem due to *openness* of Grid. InterGrid, by its nature, is a large-scale open system which includes enormous number of entities. These entities may belong to different administrative domains with potentially unknown (trust) relationships among them. It is difficult to estimate and identify its participants which usually are unknown and can freely join/leave Grid. Moreover, assigning permissions based solely on peers' identity becomes a risky business since an unknown peer should be treated as untrusted and potentially malicious. An access decision should be based on attributes of an entity requesting access to a particular resource, rather than its identity.

Recently, advantages of *attribute-based access control* in Grid has been shown [56, 111, 110, 78, 3]. Authorization information in attribute-based access control consists of requestor's attributes (e.g. Grid membership, requestor's reputation), resource's attributes (e.g. a type of applications what a service is able to execute) and environmental attributes (e.g. time, CPU load). When an access request appears, PDP checks validity of presented attributes and grants or denies access based on a security policy which defines mapping between attribute values and access rights.

Establishing Trust Between Grid Participants

Most of access control frameworks mentioned above assume that the requester has some preliminary knowledge about attributes required to access the resource. These attributes later are pushed or pulled to PDP which replies with boolean grant or deny. This approach is not expressive and flexible enough to work in an open and dynamic environment like Grid.

Trust negotiation [117, 118] is a promising technique that was proposed for access control in open environments. Trust negotiation allows to authorize peers which do not have complete knowledge about each other, belong to different administrative domains, and may have never interacted before. Trust negotiation is a policy-based technique that provides entities with the right to protect their own credentials (signed attributes) and to negotiate with other entities access to those credentials. Trust negotiation allows two entities to establish requirements to access a resource by mutually requesting each other sensitive credentials until a sufficient trust is established.

The next step in evolution of access control in Grid is the incorporation of a trust negotiation capability.

Continuous Usage Control of Long-lived Services

Usually, an access decision in Grid is evaluated only once before starting a service, and no further control is preformed while the service is executing, i.e. the access is in progress. Widely known authorization frameworks that have been integrated in Globus Toolkit, such as CAS, VOMS, PERMIS and Akenti [32] support this scenario and assume that attributes, used to produce the access decision, remain invariable in time. Therefore, the access decision can be checked only once before granting the access and the security policy will also hold when the access is in progress. But, *high dynamics* of Grid imposes a high mutability of attributes as a result of Grid participants activity. The peer's location and reputation are examples of attributes which might evolve in time. Access rights that have been granted once on the basis of attribute values at a given time, could authorize an access that lasts even when attributes changed and do not satisfy a security policy anymore.

This problem becomes more urgent especially for *long-lived services* which are

the peculiarity of computational Grids and can last hours or even days. Indeed, one of the fundamental Grid functionality is the ability of clients to execute their applications on remote resources allocated by Grid. For instance, imagine that *access to a computational service is granted if requestor's reputation is above a given threshold*. During the application execution, the reputation can be lowered as the result of other activities of the requestor. Meanwhile, one-time authorization is inadequate to affect granted permissions and can not terminate the service execution when the security policy is violated. Hence, a shift from access to usage control should be done to protect Grid services. The control should be continuously kept and a policy should be reevaluated continuously during a service execution.

The best candidate model to respond this issue is UCON [92]. UCON is capable to express comprehensive access and usage control scenarios and it handles both - continuity of control and mutability of attributes.

UCON introduces a security policy for a continuous control of resources and states how attributes can be affected and modified as a result of the policy enforcement. More precisely, continuous control means that an access decision is made not only before allowing access a service, but also when the access is in progress, i.e. the service is used by the requestor. If during ongoing access a security policy is not satisfied anymore, a reference monitor terminates the service execution and releases resources.

Assume the following scenario to highlight advantages of usage control:

security policy allows the usage of a Grid service instance s1 created by an user u1 only for 20 minutes and if the user u2 in the meanwhile never tries to access the service s2 and his reputation remains above the given threshold.

An effective usage control system should interrupt the execution of the service instance s1 and release corresponding resources as soon as the security policy for running this service does not hold anymore. The current state-of-the-art in Grid authorization is inadequate neither to express in a precise way a usage control policy, nor to enforce it. As a matter of fact, implementation of usage control in Grid is also missed.

Security Mechanisms for Continuous Control

A continuous policy enforcement poses new problems which have never been addressed in access control. Particulary, they concern *when* to perform access decision checks, and *how* to terminate ongoing accesses if a usage policy has been violated?

Access decision checks should be triggered when attributes change. Also, a 'pure' continuous enforcement stipulates that attributes should remain unchangeable until an access decision is taken and enforced. This synchronization between the access decision making and updating of attributes is needed to assure that the access decision is never falsified by possible concurrent attribute updates and the policy is enforced correctly.

If all attributes are local and reside under control of a reference monitor, the system is aware when these attributes are going to be modified and Grid participants can follow some sort of a locking protocol to achieve a continuous enforcement [104]. For instance, the system suspends an ongoing access before updating attributes, and awaits until all updates are executed. The updating actions have privilege of the exclusive access to attributes and the policy reevaluation is done only when no updates are being present in the system. When all updates are performed, the system unlocks attributes, reevaluates the access decision using new attribute values and resumes the suspended access.

Due to heterogeneity and hight dynamics of Grid, attributes are rarely under a full control of an administrative domain which enforces a security policy. Attributes reside in different places and are managed and updated by several principals. There may be a case when new attribute values are emitted constantly and, consequently, can not be locked by the system (requestor's reputation and location are examples of such attributes). Thus, the question when and how attributes are changed becomes very difficult to answer. As a result, a continuous policy enforcement is cumbersome or even unfeasible in Grid and needs some approximations.

Such approximation can be achieved through a timely-scheduled inquiring of all attributes repositories and, if attributes change a policy reevaluation is initiated [126]. Frequent checks pose a security overhead, whereas rare checks lead to possible policy violations. In any case, the system is not aware about attribute changes happened in-between of adjacent checks. For example, if a security policy grants access rights to users resided in a certain location, there is no any evidence that mobile users remained in the same location and never left it in-between of adjacent policy checks. As a matter of fact, an access decision is made taking into account some uncertainties associated with attributes.

By the way, there are several peculiarities of Grid which contribute and enhance uncertainties associated with attributes. Heterogeneous, distributed, and open nature of Grid assumes lack of a central administration and therefore of a central trust. Attributes arrive from trusted and untrusted sources, and the impact of untrusted information should be calculated and tolerated. Also, a system faultiness (e.g., revocations of attributes may get lost, update actions delayed) and a malicious activity (e.g., a man-in-the-middle, eavesdropping and impersonating of data) contribute to the trustworthiness of attributes. Further, freshness of attributes is affected by delays occurred during delivery due to a distributed nature of Grid and a network latency.

As better uncertainties associated with attributes are estimated, the better approximation of a continuous enforcement could be implemented in Grid. Authorization framework should knows as good as possible *how* and *when* attributes change to tackle this issue.

Granularity of Control

Access and usage control in Grid should be realized from several perspectives to satisfy security requirements of its participants: a *coarse-grained level of control* that manages accesses to service instances and service workflows, and a *fine-grained level of control* that monitors the usage of underlying resources allocated for the service instance.

Most of existing authorization solutions for Grid have granularity to manage access to a service invocation (i.e. coarse-grained control). Grid computational services might execute remote and potentially malicious jobs on a platform of a resource provider. If it is hard to inspect these jobs in advance, what poses serious security threats to breach a system functionality. Currently, a job submitted for execution in Grid is assigned to a local account of a resource provider host and has all permissions granted to this account. Such approach compromises the principle of least privilege [100]. Grid computational services require additional monitoring facilities and continuity of control on a level of a resource provider for better protection of computational resources.

Leveraging Existing Standards

There issue concerns the implementation of access and usage control in Grid. In fact, current authorization solutions [32] are inadequate to revoke an ongoing access and terminate a Grid service instance when a security policy protecting this service is violated. Moreover, authorization in Grid is often done through custom implementations, but standardized solutions for writing policies, performing and enforcing access decisions, etc. might eliminate time and cost paid during the design and deployment of authorization framework. Existing standards for authorization (e.g. XACML and SAML) should be exploited and extended to capture new functionalities.

1.4 Summary of Contributions

The ultimate objective of this thesis is to address problems pointed in the previous subsection and to propose a novel trustworthy, reliable, and flexible authorization framework providing a fine-grained and efficient access and usage control over computational Grid services. In order to tackle this issue, we adopt the usage control (UCON) model on the *policy level*, propose an architecture of a state-full server-side reference monitor on the *enforcement level*, and present on the *implementation level* a realization of our authorization framework in Globus Toolkit. In the following, contributions done on each level are described in more details.

Policy Level

The policy level allows to express comprehensive access and usage control scenarios. Our model is based on the Usage CONtrol (UCON) model proposed by R. Sandhu and J. Park. UCON is an attribute-based access and usage control model suitable for highly dynamic, heterogeneous and open computing environments like Grid. UCON covers scenarios where a continuity of control is needed and mutability of attributes exists. Access decision is based on evaluation of authorization and condition predicates, and fulfillment of obligation actions. UCON also assumes updating of attributes as a result of a policy enforcement. This thesis presents several examples of UCON policies to protect Grid services on both coarse- and fine-grained levels. POLPA [82, 16] language was chosen as an underlying formalism to express security policies. The policy level contributes our authorization framework also by providing U-XACML policy specification language [33]. The U-XACML language enhances the XACML language with usage control features.

The coarse-grained level of control is enhanced with trust negotiation. Trust negotiation preserves security threat of unauthorized disclosure of sensitive credentials and security policies. It preserves peer's privacy and reveals minimum of peer's information needed to succeed coarse-grained authorization.

Fine-grained control protects a Grid node from threats of a malicious behavior of jobs submitted for execution. A fine-grained reference monitor observes traces of actions performed by jobs and terminates those executions which violate security policies. Our model integrates two levels of control, coarse- and fine-grained, through sharing attributes between these two levels.

A continuous policy enforcement requires a scheduler telling when the access reevaluation should be taken. This scheduler is given in scope of a risk-aware access and usage control model. This model computes uncertainties associated with each attribute, while a security policy states a threshold of the acceptable uncertainties. We name this threshold as a risk, and more precisely it specifies a probability of a failure of policy statements, which use the uncertain attribute value, multiplied on the impact of this failure. The main purposes of the risk is to reason how trustworthy are attributes, and how better to approximate a continuous policy enforcement.

Enforcement Level

The enforcement level revises functionalities of authorization components and authorization protocol to handle a continuous policy enforcement. It provides an architecture of a server-side reference monitor which enforces security policies. Monitoring of Grid services is assumed on two levels of control: a coarse-grained - that manages accesses to a service instance, and a fine-grained - that monitors a usage of underlying resources allocated for the service instance. We constitute state-full interactions between authorization components which form the reference monitor.

Implementation Level

The proof-of-concept implementation of coarse- and fine-grained levels of control is done in Globus Toolkit, the most used middleware to setup computational Grids. The coarse-grained level is assumed for any Globus service, while the fine-grained level is realized for the GRAM service only. Our implemented prototype shown good performance results and could be easily plugged into the existing Globus authorization infrastructure. The main novelties of our implementation are a continuous policy enforcement, efficient management of attributes and possibility to terminate ongoing accesses to services whose security policies have been violated.

To resume, the coherent outcome of this thesis is the full design of access and usage control model in Grid from specification till implementation.

1.4.1 List of Publications

The following is a list of publications of conference and journal papers, where I am a co-author, produced during my PhD studies. The list consists of two parts. The first part enumerates publications used to write this thesis:

[30] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. Controlling the usage of grid services. *International Journal of Computational Science*, 3(4):373–387, 2009

[31] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. On usage control for grid services. In *proceedings of International Joint Conference on Computational Sciences and Optimization: CSO'09*, pages 47–51, Washington, DC, USA, 2009. IEEE Computer Society

[32] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. *Handbook of Information and Communication Security*, chapter Access and Usage Control in Grid Systems, pages 293–308. Springer, 2010

[33] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. A proposal on enhancing XACML with continuous usage control features. In *proceedings of CoreGRID ERCIM Working Group Workshop on Grids, P2P and Services Computing*, pages 133–146. Springer US, 2010

[38] D. Fais, M. Colombo, and A. Lazouski. An implementation of role-base trust management extended with weights on mobile devices. *Electron. Notes Theor. Comput. Sci.*, 244:53–65, August 2009

[60] H. Koshutanski, A. Lazouski, F. Martinelli, and P. Mori. Enhancing grid security by fine-grained behavioral control and negotiation-based authorization. *International Journal of Information Security*, 8(4):291–314, 2009

[66] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin. Influence of attribute freshness on decision making in usage control. To appear in proceedings of 6th International Workshop on Security and Trust Management: STM'10, 2010

[67] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin. Risk-aware usage decision making in highly dynamic systems. *International Conference on Internet Monitoring and Protection: ICIMP'10*, pages 29–34, 2010

[68] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin. Risk-based usage control for service oriented architecture. In *proccedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing: PDP'10*, pages 641–648. IEEE Computer Society, 2010

[73] A. Lazouski, F. Martinelli, and P. Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, 2010

The second part lists other papers not included in the thesis:

[34] G. Costa, N. Dragoni, A. Lazouski, F. Martinelli, F. Massacci, and I. Matteucci. Extending security-by-contract with quantitative trust on mobile devices. In *proceedings of International Conference on Complex, Intelligent and Software Intensive Systems: CISIS'10*, pages 872–877, Washington, DC, USA, 2010. IEEE Computer Society

[35] G. Costa, A. Lazouski, F. Martinelli, I. Matteucci, V. Issarny, R. Saadi, N. Dragoni, and F. Massacci. Security-by-contract-with-trust for mobile devices. *Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications*, 1(4):75–91, 2010

[65] L. Krautsevich, A. Lazouski, F. Martinelli, P. Mori, and A. Yautsiukhin. Usage control, risk and trust. In *proceedings of 7th international conference on Trust, privacy and security in digital business: TrustBus'10*, pages 1–12, Berlin, Heidelberg, 2010. Springer-Verlag

1.5 Structure of the Thesis

Each chapter of the thesis corresponds to the particular contribution highlighted in the previous section. In order to make the thesis self-contained, the introductory to the usage control (UCON) proposed by R. Sandhu and J. Park as well as the survey of existing access and usage control approaches in Grid are outlined in Chapter 2. Moreover, each chapter contains a brief and specialized state-of-the-art related entirely to the discussed issues.

Chapter 2 provides fundamental background information on access and usage control in Grid [32, 70]. It outlines the Usage CONtrol (UCON) model proposed by

R. Sandhu and J. Park. Furthermore, Chapter 2 surveys the existing implementations of access and usage control in Grid: CAS, PERMIS, VOMS, Cardea, PRIMA, etc.

Chapter 3 outlines the policy level of our access and usage control model for Grid services. It introduces the conceptual policy model, U-XACML policy specification language, the policy formal model based on POLPA, and gives several examples of security policies on coarse- and fine-grained levels of control.

Chapter 4 presents the enhancements on the policy level and introduces models of attributes management and continuous policy enforcement. Trust negotiations schema is proposed to acquire the initial set of needed attributes. A risk-aware access and usage control model is proposed to facilitates a continuous and reliable policy enforcement in distributed environment.

Chapter 5 proposes the enforcement level of our framework. It gives the model of the state-full reference monitor, its component-based architecture, functionality of main components and the message-flow between them.

Chapter 6 presents the implementation level of our framework. It proposes the integration of trust negotiations in Globus Toolkit. The coarse-grained level of control is realized to monitor any Grid service, while the fine-grained level is implemented for Grid computational services only. The performance evaluation is given to estimate the security overhead.

Chapter 7 briefly recalls the main achievements and results of the thesis, draws final concluding remarks, and points to future investigations in the area.

Chapter 2

Background and State of the Art

This chapter outlines the Usage CONtrol (UCON) model proposed by R. Sandhu and J. Park [91, 101, 90, 92, 70]. UCON is the attribute-based access and usage control model suitable for highly dynamic, heterogeneous and open computing environments (section 2.1).

Furthermore, this Chapter surveys the existing access and usage control approaches in Grid [32]: CAS, PERMIS, Akenti, Shibboleth, VOMS, Cardea, PRIMA, Sandhu's approach (section 2.2). Whereas these approaches go far beyond the first generation of access control in Grid based on the GridMapFile (see section 1.1.1), they are still inadequate to express and enforce usage control in a precise way.

2.1 Usage CONtrol (UCON)

This section presents a novel promising access and usage control model proposed by R. Sandhu and J. Park and called UCON. UCON enhances attribute-based access control models (e.g. DAC, MAC, RBAC [100]) in two novel aspects [92]: 1) **mutability of attributes**, and 2) **continuity of control**. *Mutability of attributes* means that attribute (used to produce access decision) are not static and can change in time. Moreover, changes of attributes mights be caused as the result of the entire security policy enforcement. *Continuity of control* means that access decision should be evaluated before granting access to a resource and also when the requestor executes access rights on the resource. If attributes change when access is in progress and the security policy is not satisfied anymore, the reference monitor revokes the granted access rights and terminates the resource's usage. This section describes UCON following the PEI model [66]. Subsection 2.1.4 presents the policy level of UCON, while subsection 2.1.3 focuses on the enforcement level. Subsection 2.1.4 ends this section by listing existing implementations of UCON.

2.1.1 Policy Level Specification

Policy level states what authorization information is used to make access decisions, how access decisions are made, and what scenarios can be expressed by means of the UCON model.

A request to access and use a resource is granted based on attributes of the requestor, of the resource, and the environment. There are three decision factors in UCON: *authorizations*, *obligations*, and *conditions*. Usage control policies specifies 24 core scenarios and vary depending on when an access decision is made, what decision factor is used to produce the access decision, and when attributes are updated as a result of the policy enforcement.

UCON Model Components

The primary step in the design of any access control model is to specify resources (*objects*) to be protected, user (*subjects*) which issue requests to access and execute some *access rights* on resources.

Subjects. A requestor or *subject* requests a resource, and executes granted access rights on requested resources [90]. The subject is represented by subject' attributes, $ATT(S)$. Attributes are assertions on the subject properties, characteristics and capabilities (e.g. subject's id, affiliation, reputation, location) issued by some trusted authorities.

Objects. Objects are resources, and subjects can access or use objects. Objects can be of various types, e.g. high-level services, low-level computational resources like CPU cycles, files, network sockets, etc. Objects in UCON are also characterized by attributes, $ATT(O)$. Examples of attributes are resource's type, security label assigned to the resource, etc.

Attributes. Besides subject's attributes $ATT(S)$ and objects's attributes $ATT(O)$, the UCON model specifies environmental attributes $ATT(E)$, system-central assertions about the computational environment where the subject and the object operate. A system time is the most common example of the environmental attribute.

The main novelty of the UCON model is *attributes might change in time*. The idea of attributes mutability is slightly and implicitly introduced in existing access control models. For example, a status of roles in the RBAC model [22, 121] is mutable (it can be passive or active) and depends on a current authorization context. Also, the DRM system introduced in [67] defines mutable attributes (e.g. a number of possible accesses to a digital content) which decrease iteratively when a new request arrives.

Attribute mutability is a backbone of the UCON model since *attribute changes affect taken access decisions and might cause the reevaluation of security policies*. Attributes can be mutable by several reasons: 1) by the nature (e.g. the system time), 2) by other activities of subjects and objects (e.g. the subject's location, the reputation of the object's provider), 3) attributes can be modified as the result of

access (e.g. an attribute - number of previous accesses - increases after each access). Attribute changes, occurred as the result of access, are encoded into a security policy and are named as *attribute update actions*. The UCON model implies updates before the access is granted (encoded as *preUpdate*), during the access (*onUpdate*), and after the access (*postUpdate*) [92].

There are many possible ways to classify attributes is UCON. Besides *mutable attributes*, UCON also deals with *immutable attributes*. These attributes can be modified by administrative actions only (e.g., subject's identity is an immutable attribute). Based on *purposes of mutability*, attributes are classified in [93] on:

- *Exclusive/Inclusive* attributes which are used to resolve conflicts of interests, e.g. dynamic separation of duty;
- *Consumable* attributes, .e.g pre-paid credits, digital tickets. These attributes usually are destroyed as the result of a security policy enforcement;
- *Immediate revocation* attributes, e.g. the system time. Access to a resource is terminated exactly when time expires;
- *Obligation* attributes, e.g. a status of an agreement between a subject and an object. These attributes are changes as the result of obligation actions fulfillment (obligations will be presented in the next subsection);

Taking into account time validity, attributes may be *temporary* or local (valid only for a single access) and *persistent* or global (valid for many access). The more detailed classification of UCON attributes is given in [93].

Attributes in UCON can be of different types and can be issued by different authorities. Originally, the UCON model implies that all attributes are created and managed within one security domain. These attributes are neither transferable nor understandable in other security domains. Definitely, this approach needs some further improvements to handle scenarios which can happen in open systems, e.g. collaborations in interGrid assume that a subject (a Grid user) and an object (a Grid service) reside in different security domains. The approach in [102] examines this issue and extends the original UCON model to address the dynamic aspects of multi-domain dynamic interactions. The approach assumes that a subject is mobile and traverses through multiple security domains accessing different objects using same attributes. Attributes issued in one security domain should be properly translated and used in another security domain. In [102] there are four possible attribute combinations:

- *Pre-defined Local Attributes* are attributes defined in a local security domain and which are used only in this local domain;
- *Pre-defined Multi-domain Attributes* are issued in some security domains but understandable in all collaborating domains. This requires a priori agreements between domains over semantics of such attributes;

- *Dynamic Local Attributes* are created and destroyed dynamically (i.e. as a result of executing access rights) by the system. These attributes are used in the local security domain;
- *Dynamic Multi-domain Attributes*: are created and destroyed dynamically by several domains. It could be a case, that an attribute is created in one domain, but destroyed in another as a result of performing some activity by its holder. To use such attributes an ad-hoc interpreter from one domain to another is required.

Access Rights. UCON access rights are the same as access rights and permissions in traditional access control. Access rights denote permissions which can be exploited by subjects on objects [92] (e.g. to read or write a file, to invoke a particular interface of a Grid GRAM computational service). The main difference is that UCON access rights imply *long-lived accesses* which can last hours or even days.

Access Decision Making

Access decision in UCON is a function mapping current attributes either to grant or deny. It is based on evaluation of *authorizations* and *conditions*, and fulfillment of *obligations*. Authorizations are predicate which put constrains on subject's and/or object's attributes. Conditions are environment restrictions that are required to be valid before or during the usage. Obligations are actions that are required to be performed by a subject [92] before, during or after the usage. Authorizations, conditions, and obligations are called *decision factors* and are discussed in this subsection.

Authorizations. Neither the UCON conceptual model nor the formal policy model specify what formalism should be used to express attributes and reason on authorization predicates validity. In fact, authorization predicates should be always computable in a reasonable time and they validity should be proved by ordinal mathematical reasoning.

Definition 1. (**Authorization Predicate**) *Authorization predicate is a computable boolean function which maps subject and object attributes to true or false:*

$$P_A : ATT(S) \cup ATT(O) \rightarrow \{true, false\}$$

As example, assume the authorization predicate which grants access to ".doc" files only to users which are elder than 21 years and have name "John":

$$P_A : subject.name = John \wedge subject.age > 21 \wedge object.type = doc$$

In contrast with traditional access control models, UCON authorization predicates are evaluated before granting access (*pre*-authorizations) and when access

rights are executed (*on*-authorizations). *On*-authorizations should be valid all the time when access is in progress.

Obligations. To the best of our knowledge, the term "obligations" was firstly introduced by Minsky and Lockman [85]. They focused on the integrity protection of a resource, rather than on the confidentiality usually imposed by authorizations. The execution of access rights may violate integrity of the resource. Thus, some obligation actions must be performed in a foreseeable future to recover the resource integrity.

Obligations encoded as policy statements appear in access control to refer to some actions associated with access rights [23]. Zhao et al. [130] discuss PERMIS RBAC authorization infrastructure, where obligations are tasks and requirements fulfilled together with the enforcement of access decisions. Further, in DRM solutions, a payment procedure for usage of a digital resource can be identified as an obligation action. Recently, several policy languages were enhanced to support the specification of obligation requirements [51, 48].

Obligations in UCON have some specific features and present an active area of research [51, 23, 43]. Obligations examine the accomplishment of mandatory tasks that are relevant to the resource usage and must be fulfilled to grant the access. As example, assume that to access a white paper of a company, a user is required to sign a privacy policy, to watch an advertisement while reading the paper on-line and, finally, to delete the paper from his PC within 10 days if he would download the paper. Thus, obligations in UCON can be fulfilled before, during, and after executing access rights [55].

UCON obligations are defined in [55] as a tuple of *OBL = (OBS, OBO, OBA, WHEN, DURATION)*.

OBS and *OBO* refer to an obligation subject (*OBS*) and an obligation object (*OBO*) respectively. The obligation subject is an entity that actually has to perform an obligation action (the reference monitor itself could be *OBS*). The obligation subject and object may not be the same subject and object of the access request, i.e. the requestor subject (S) and the resource object (O). Relationships between these components (i.e. between *OBS* and S and between *OBO* and O) are not specified by the original UCON model, but depend on a specific scenario. Actually, obligations specified by a security policy are based on S's and O's attributes and access rights, while the fulfillment of obligations can be performed by additional entities (e.g., *OBS* and *OBO*) [90]. It's worth to mention, that in [55] attribute update actions are considered as obligations performed by the reference monitor. In the sequel, we adhere to the original UCON model and distinguish attribute update actions from obligations.

WHEN addresses when obligations should be fulfilled: before the access (*pre*-) or during the access (*on*- and sometimes are identified as provisions) or after when the access is ended (*post*-obligations). Post-obligations is a concept widely used in *distributed usage control* [48]. Distributed usage control protects access to digital information copied and disseminated in the network.

DURATION means that obligations should be fulfilled within a specific period of time. For example, a user must watch an advertisement for the first 20 seconds starting to use an object.

Conditions. Conditions are environmental constrains that are taken into account producing an access decision [92].

Definition 2. (**Condition Predicate**) *Condition predicate is a computable boolean function which maps environmental attributes to true or false:*

$$P_C : ATT(E) \rightarrow \{true, false\}$$

"An object is available on working hours only" is an example of conditions:

$$P_C : environment.currentTime \in [Mon - Fri, 8a.m. - 6p.m.]$$

The evaluation of condition predicates can be done before granting the access to an object (*pre*-conditions) and/or when the subject uses the object (*on*-conditions). Conditions evaluation cannot result in update of subject, object or environmental attributes. Environmental attributes can be changed intrinsically as a result of environment modifications.

Usage Policy and Scenarios

The UCON reference monitor is modeled as a finite-state transition system and its current state is characterized by current values of attributes. That is, the state is a function on a set of subjects and their attributes, the set of objects and their attributes, and the set of environmental attributes [124]. There is a special system attribute to specify a status of a single access process. This attribute maps the access process to *initial*, *requesting*, *denied*, *accessing*, *revoked* or *end* states [127]. The reference monitor transits from one state to another as the result of attributes update actions, obligations fulfillment, predicate checks and usage control actions execution.

By default, the reference monitor remains in the *initial* state. A subject s requests an access right r to use an object o by executing the $tryaccess(s, o, r)$ action. The reference monitor captures this request and moves to the *requesting* state. Further, the reference monitor evaluates *pre*-authorization predicates and either denies the access immediately by issuing $denyaccess(s, o, r)$ action (if *pre*-authorization predicates are not satisfied) or proceeds by executing the $preupdate(s, o, r)$ action to update attributes. Later on, the reference monitor grants the access to the resource by performing the $permitaccess(s, o, r)$ action. The subject s starts to execute granted access rights r, and the reference monitor transits to an ongoing *accessing* state and starts *continuous control*. The reference monitor continuously monitors mutable attributes and reevaluate authorization and condition predicates each time attributes change. Also, updates of subject's/object's attributes may be required by

the security policy and this is performed by the $onupdate(s, o, r)$ action. If at least one among authorizations, conditions or obligations is violated when the access is in progress, the reference monitor interrupts the access by the $revokeaccess(s, o, r)$ action. If the subject wishes to terminate the ongoing access from his side, the $endaccess(s, o, r)$ action is initiated. In both cases, the $postupdate(s, o, r)$ action is performed on subject's and/or object's attributes, but the result of these updates varies whether the usage process was terminated by the reference monitor and it transits to the *revoked* state, or by the subject and it transits to the *end* state. For instance, such attribute like subject's reputation can be updated differently. It could be increased if the usage was ended normally, or decreased if the usage was terminated by the reference monitor as the result of the policy violation. The usage session and policy enforcement is over when the $postupdate(s, o, r)$ action is executed and the reference monitor moves to its *initial* state.

Definition 3. (**UCON Model**) *UCON model is a 5-tuple* $M = (S; A_T; P_A; P_C; P_B)$ *[124]:*

- S *is a set of sequences of system states, where a single state is characterized by current values of* $ATT(S)$, $ATT(O)$, $ATT(E)$;
- A_T *is a finite set of usage control actions* $A_T = \{tryaccess(s, o, r)$, $endaccess(s, o, r)$, $permitaccess(s, o, r)$, $denyaccess(s, o, r)$, $revokeaccess(s, o, r)$, $preupdate(attr)$, $onupdate(attr)$, $postupdate(attr)\}$;
- P_A *is a finite set of authorizations;*
- P_C *is a finite set of conditions;*
- P_B *is a finite set of obligation actions.*

The UCON security policy is constructed from ingredients of the UCON model and serves to specify acceptable sequences of system states, while the UCON reference monitor preserves that it behavior always adheres to the policy.

Each UCON security policy is paired with the following meta information expressing UCON novelties - attributes mutability and continuity of control [9]:

- *Access continuity and decision timings* specify when access decision is made. If the decision to grant some access rights is made before the usage, the resulting usage control scenarios are encoded with the prefix *pre*. Access decisions evaluated continuously during the execution of granted access rights are encoded with the prefix *on*;
- *Policy statement type* a security policy can be stated using authorization predicates only, obligation actions only, condition predicates only, or any of their combinations. These policy statements are encoded with letters A, B, and C respectively;

- *Attributes update timings* subject's and object's attributes may require updates before the access (these updates are done by *preupdate*(*attr*) action and denoted with 1), when the access is in progress (*onupdate*(*attr*) action and encoded with 2), or when access rights execution is revoked by the protection system or terminated by the subject (*postupdate*(*attr*) action and encoded with 3). If no attribute updates are assumed by the policy, the corresponding usage scenarios are encoded with 0.

The UCON policy meta-information is shown in figure 2.1. It represents the UCON scenarios on a time line highlighting a continuity of control. In the hatched frame there is a meta-information usually used to express traditional access control scenarios.

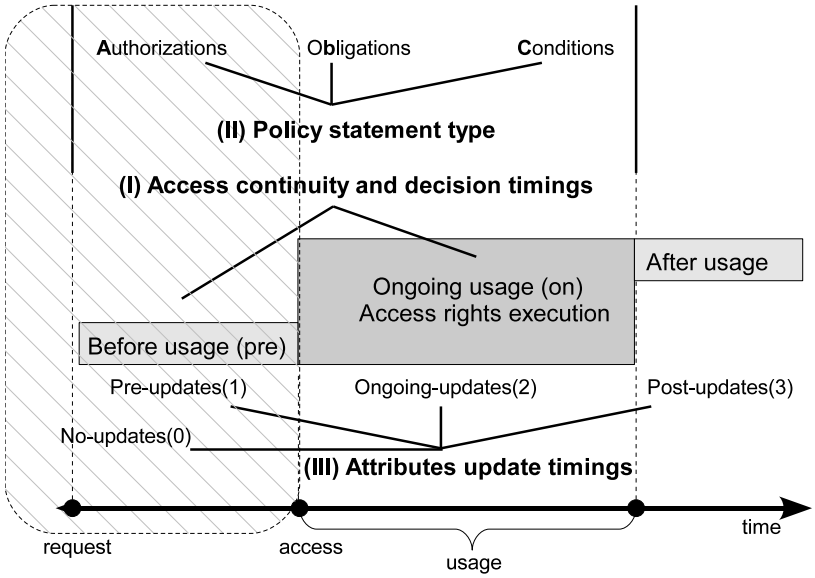

Figure 2.1: UCON usage scenarios

The UCON policy model identifies 24 core scenarios - **pre,on** × **A,B,C** × **0,1,2,3** - depending on what elements of UCON model are used to build a policy: pre and ongoing authorizations, obligations and conditions with attribute updates that might be performed before, during, or after the access.

For example, a **preA3** UCON policy, called as *pre-authorization with post-updates*, represents a usage scenario where access decision is evaluated only once before usage starts. The access decision is done by checking authorization predicates only, and one or more attributes are updated after the resource usage ends. The example of the *preA*3 policy given in [92] grants access rights to a subject who possesses the attribute that proves his membership in a certain organization. The subject has to present an approval with the initial request before using the resource. Finally, the subject attribute 'balance' is charged when the usage is ended according to the usage time.

In [90] all 24 usage scenarios are described systematically and comprehensively. Several examples of traditional access control particularly DAC, MAC, RBAC are modeled there by means of UCON. Actually, traditional access control can be expressed entirely with **preA0** (pre-authorizations without attribute updates) and **preA1** (pre-authorizations with attribute pre-updates) UCON policies. However, complicated usage scenarios may require a combination of several core policy models.

2.1.2 Policy Level Formalization

The foremost innovation of UCON is **continuity of control**. UCON moves from a single evaluation of the access decision (before the access starts) to its continuous reevaluation during the access. This demands specific formal models to express UCON security policy and enforcement mechanisms.

The formal model should be expressive enough to portray the UCON model components and usage scenarios. Also, it should deal with concurrency. For example, one might need to express a concurrent fulfilment of obligation actions, attribute updates and access decision reevaluation in *on*-going usage scenarios, updates of attributes shared between concurrent usage sessions, etc.

The UCON policy is formalized either as logical formulas in some kinds of temporal logic, or as a process in process algebras. In this section, each group of models is sketched. The rest of this subsection points to compare these models.

Logic-Based Formalization of UCON Policy

UCON in Temporal Logic of Action (TLA). To the best of our knowledge, the UCON model was firstly formalized and analyzed using an extension of TLA (temporal logic of actions in notation of Lamport [69]) by Zhang et al. [124, 128, 129, 127].

TLA was introduced as a method for describing and reasoning about concurrent systems. Variables, values, actions, predicates, functions are basic concepts of TLA [69]. The semantics of TLA is defined in terms of states. A state is an assignment of values to variables. In the UCON model, a system state is a mapping from a set of attribute names to the collection of attribute values. The usage control policy defines a permitted sequence of system states. Enforcing the UCON policy, the UCON reference monitor should transit only to states which satisfy policy statements.

The policy statement is represented in TLA as a logical formula built from predicates and actions with logic connectors and temporal operators. A predicate maps a state to a boolean value, and an action represents a relation between old and new states. The logical formula in UCON is defined by the following grammar in BNF [128]:

$$\text{ø} ::= a|p|(\neg\text{ø})|(\text{ø} \wedge \text{ø})|(\text{ø} \longrightarrow \text{ø})|\Box\text{ø}|\Diamond\text{ø}|\bigcirc \text{ø}|\text{ø}U\text{ø}|\blacksquare\text{ø}|\blacklozenge\text{ø}|\ominus \text{ø}|\text{ø}S\text{ø}|$$

where a is an action (e.g. *tryaccess(s,o,r)*, *permitaccess(s,o,r)*, *denyaccess(s,o,r)*, *endaccess(s,o,r)*, *revokeaccess(s,o,r)*), p is a predicate (e.g. authorizations or conditions), there are temporal operators $\Box$ "always", $\Diamond$ "eventually", $\bigcirc$ "next", U "until", $\blacksquare$ "has-always-been", $\blacklozenge$ "once", $\ominus$ "previous", S "since" whose formal semantics as well as TLA axioms can be found in [69].

The UCON **preA0** usage scenario implies no attribute updates. The example given in [124] considers the DAC protection system, where individual identities and an access control list (ACL) are subject and object attributes, respectively. ACL is a functional mapping of an object to multiple subject's *ids* and rights r. This model can be expressed in TLA as follows [124]:

$$permitaccess(s,o,r) \rightarrow \blacklozenge(tryaccess(s,o,r) \wedge ((id(s),r) \in ACL(o)))$$

According to this policy, the UCON reference monitor performs the *permitaccess(s,o,r)* action which grants a permission to s and starts the execution of r. The policy states that the *permitaccess(s,o,r)* action implies that authorization predicates have to be true "before" this action is executed. The predicate $((id(s),r) \in ACL(o))$ is true if the access control list (modeled as object's attribute) contains a mapping of subject's id (modeled as subject's attribute) and requested rights. The predicate should have been true in the state when the request was received from the user (the *tryaccess(s,o,r)* action).

In [127, 124] there is a detailed formalization of all UCON core models. By the way, each core model can be expressed by instantiating formulae derived from the fixed set of rules. Regarding to the completeness and soundness of the UCON policy formal mode, Zhang et al. enlisted the fixed set of scheme rules (2 control rules (CR) for pre (CR1) and ongoing (CR2) access decisions and 6 update rules (UR) for pre- (UR1), ongoing (UR2, UR3, UR4) and post- (UR5, UR6) attribute updates) [124]:

- control rules are:

$$CR1 : permitaccess(s,o,r) \rightarrow \blacklozenge(tryaccess(s,o,r) \wedge (\bigwedge_{n_i} pa_{n_i}) \wedge (\bigwedge_{n_k} pc_{n_k})) \wedge (\bigwedge_{n_j} \blacklozenge ob_{n_j})$$
$$CR2 : \Box(\neg((\bigwedge_{n_i} pa_{n_i}) \wedge (\bigwedge_{n_j} (pb_{n_j 1} \wedge ... \wedge pb_{n_j m} \rightarrow ob_{n_j})) \wedge (\bigwedge_{n_k} pc_{n_k})) \wedge (state(s,o,r) = accessing) \rightarrow revokeaccess(s,o,r))$$

  where $1 \leq n_i \leq i, 1 \leq n_j \leq j, 1 \leq n_k \leq k$, and $pa_1, ..., pa_i$ is a set of authorization predicates; $ob_1, ..., ob_j$ is a set of obligation actions; $pb_{n_j 1}, ..., pb_{n_j m}$ are predicates to determine when the ongoing obligations ob_{n_j} should be fulfilled; $pc_1, ..., pc_k$ is a set of condition predicates;

- update rules are:

$$UR1 : permitaccess(s,o,r) \rightarrow \blacklozenge preupdate(attr)$$
$$UR2 : permitaccess(s,o,r) \rightarrow \Diamond(onupdate(attr) \wedge$$

$(endaccess(s, o, r) \vee revokeaccess(s, o, r)))$
$UR3 : \Box((state(s, o, r) = accessing) \rightarrow onupdate(attr))$
$UR4 : \Box((state(s, o, r) = accessing) \wedge p_{u1} \wedge ...p_{uj} \rightarrow onupdate(attr))$
$UR5 : endaccess(s, o, r) \rightarrow \Diamond postupdate(attr)$
$UR6 : revokeaccess(s, o, r) \rightarrow \Diamond postupdate(attr)$

where $p_{u1}, ..., p_{uj}$ are predicated that trigger an attribute update during the access.

The example of the DAC protection system presented before can be expressed by the CR1 control rule only.

Actually, the UCON formal model proposed by Zhang et al. has several assumptions. Firstly, policies are specified for positive access permissions only. Anything not explicitly permitted by a security policy is prohibited. Further, the formal model focuses on a specification of a single usage process, interactions between several concurrent usage processes is not pointed. Then, the formal model requires some background in TLA to write a policy for ongoing usage scenarios. Finally, the logic-based specification is useful to check some properties of the system, if the system and properties are given in the same formalism. Indeed, the UCON reference monitor should enforce core UCON scenarios rather than to perform some kinds of reasoning. Thus, the advantages of such formalization from the implementation prospective are not clear.

UCON with Components Creation. To overcome the assumption of a single usage session made in the initial formalization, Zhang et al. [124] introduced a formal model with sequential usage sessions and components creation. Particularly, the formal model specifies how to create and destroy objects, subjects, and their attributes, and considers serialized usage processes. By serialization, Zhang et al. assume that no interference between any two usage processes occurs, so that an overall effect of the UCON policy enforcement is as though the individual usage processes are executed sequentially one after another. The formal model was proposed to analyze security properties and expressiveness of the UCON model. The formal model does not focus on ongoing usage scenarios and consider only the overall effect induced by a sequence of non-interfering usage processes in creating/destoying new subjects/objects and updating attributes.

UCON policies proposed in [124] define mainly pre-authorization and pre-obligation usage scenarios with post-updates (e.g. the **preA3** and the **preB3** usage scenarios). Instead of the usage control actions *tryaccess(s,o,r)*, *endaccess(s,o,r)*, *permitaccess(s,o,r)*, *denyaccess(s,o,r)*, *revokeaccess(s,o,r)*, the formal model introduces the $permit(s, o, r)$ predicate and a set of new primitive actions: *createObject()* (generates a new object or subject within the system), *destroyObject()* (removes a object or subject from the system) and *updateAtt()* (updates a value of an attribute). Primitive actions cause the state transition of the protection system, and normally the *createObj()* action is followed by the *updateAtt()* action to initialize attributes of the object/subject just created by the system.

The usage control policy consists of two parts [124]: a *condition* and *body*:

$$policy\_name(s, o) :$$
$$(condition) : p_1 \wedge p_2 \wedge ... \wedge p_n \rightarrow permit(s, o, r)$$
$$(body) : act_1; act_2; ...; act_k$$

The *condition* part contains access rules. Access rules determine the access decision and are represented as a conjunction of authorization, condition predicates and obligation actions. If the conjunction is true the access rule grants a certain access right to a subject. The *condition* part is assumed to be computable by means of ordinary mathematical reasoning.

The second part of the policy, *body*, is a sequence of primitive actions. If the conditional part of the policy is satisfied, the UCON reference monitor enforces the second part executing these primitive actions.

The example of the usage policy proposed in [124] considers the following scenario. Suppose that an object (an electronic document, *doc*) can be issued by a subject whose attribute '*role*' equals to '*scientist*'. For *anonymous* users this document can be read online no more than 10 times. This statement is managed by the object attribute '*readTimes*'. Each time the anonymous user is allowed to read the document, the object's attribute is decreased by one. The usage control policies are as follows [124]:

- *Creating policy*:

$$createDoc\_policy(s, doc) :$$
$$(s.role = scientist) \rightarrow permit(s, doc, create)$$
$$createObj(doc); updateAtt : doc.readTimes = 10$$

- *Reading policy*:

$$readDoc\_policy(s, doc) :$$
$$(s.role = anonymous) \wedge (doc.readTimes > 0) \rightarrow permit(s, doc, read)$$
$$updateAtt : doc.readTimes = doc.readTimes - 1$$

The formal model addresses some very important issues and ideas to analyze safety of the UCON model [129]. As limitation, it does not specify all 24 basic UCON usage scenarios while dealing with *pre*-usage scenarios with *post*-updates mainly.

UCON in Interval Temporal Logic (ITL). An alternative approach to formalize the UCON model is documented in [53]. The approach deals mostly with continuous *on*-going usage scenarios. It redefines the semantics of the usage process and request. The proposed formal model assumes several usage requests within one usage process, while the original UCON model implies a single usage process for

each usage request. For example, consider the usage of a personal mailbox though a web-interface. A user starts the usage process by performing the login action (e.g. *tryaccess(s,o,r)*), and further is allowed to perform additional usage requests (e.g. mail browsing, mail sending) within the same usage process. The user terminates the usage process by logging out (e.g. *endaccess(s,o,r)*). Thus, Janicke et al. [53] concern about subject's behavior, i.e. a sequence of semantically related usage requests placed into one usage process. Notice, that generally the order of usage requests is formed dynamically and arbitrary by the subject. Moreover, the usage process is accompanied with the attribute updates concurrent to subject's behavior.

The formalization is based on ITL and presents the usage control policy as a set of logical formulas. The semantics and syntax of ITL can be found in [53]. A key notion of ITL is the interval. The interval is a (in)finite sequence of system states. A usage process is formed by a sequence of intervals constructed using a set of operators. For instance, (i) int^* - "star" operator - denotes an interval decomposable into a set of a (in)finite number of intervals int, (ii) $int_1; int_2$ - "chop" operator - implies that a last state of int_1 coincides with an initial state of int_2. For instance, the model assumes that the final state of the $tryaccess(s, o, r)$ action is pared with the initial state of $denyaccess(s, o, r)$ and $preupdate(attr)$ actions. The specification of the usage process in terms of intervals is defined as follows [53] (the operator $\oplus$ denotes the logical exclusive-or):

$$
\begin{aligned}
usage(s, o, r) ::= {} & tryaccess(s, o, r); \\
& (denyaccess(s, o, r) \oplus \\
& \quad (preupdate(attr) \wedge permitaccess(s, o, r); \\
& \quad\quad (((usage(s, o', r') \oplus idle(s)) \oplus do(s, o, r))^* \wedge \\
& \quad\quad\quad onupdate(attr)); \\
& \quad\quad (revokeaccess(s, o, r) \oplus endaccess(s, o, r)); \\
& \quad\quad postupdate(attr)))
\end{aligned}
$$

ITL extends the original UCON model in several ways. Firstly, an explicit statement is given when the access should be denied. This makes possible to express in the same policy positive and negative authorizations. Secondly, if the access is permitted, a subject can initiate other usage requests within the main one. Generally, the choice of the action to execute is non-deterministic and made by the subject. A security policy concerns the order of subject's actions, and an action is permitted depending on the subject behavior, i.e. the history of previous usage requests. The subject behavior is indicated by $((usage(s, o', r') \oplus idle(s)) \oplus do(s, o, r))$. *do(s,o,r)* refers to the execution of the requested action, *idle(s)* denotes that the subject does not perform any actions, and $usage(s, o', r')$ indicates a new usage request within the main usage process. In the example of using of the personal mailbox though a web-interface. The main usage process $usage(s, o, r) = mail\_box$ is started by performing the login action, and further a particular sequence of usage requests can be initiated within the main process. $usage(s, o', r') = compose\_mail$ and than $usage(s, o'', r'') = send\_mail$ is an example of such sequence.

The UCON formal model based on ITL assumes also a separate attribute update policy for each usage process [53]. *preupdate*(*attr*) denotes all update actions that must be performed before the usage, and similarly *onupdate*(*attr*) and *postupdate*(*attr*) during and after the usage. Thus, the model states that a subject behavior goes concurrently with attribute updates.

Distributed Usage Control. The term "distributed usage control" was introduced by Hilty et al. [48]. It addresses specific issues on protecting of digital information released by a provider to a remote consumer. The distributed usage control ensures that digital information is used by the remote user according to the security policy defined by the provider. It is powerful to enforce various security requirements, e.g. "do not redistribute an object", "delete an object within 30 days", etc. Actually, the distributed usage control is relevant to DRM models. Moreover, the obligation specification language (OSL), distributed usage control policy language, can be partly translated into DRM rights expression languages such as XrML, ODRL [48].

The distributed usage control model intersects with the original UCON model but uses different definitions and terms [48]. Actually, it also deals with the continuous enforcement of usage decisions, but is less suitable to specify attribute updates. Moreover, the distributed usage control approach is able to formalize post-obligations. Post-obligations are mandatory tasks expected to be fulfilled in the future after the usage process terminates. Usually, post-obligations must be fulfilled in a specific time point, e.g. "an object must be deleted within 7 days". Neither TLA-based nor ITL-based UCON formal models deal with post-obligations.

The formal model of the distributed usage control is logic-based. The semantics of the formal model is defined over traces with discrete time steps. At each time step, a set of events can occur. Each event corresponds to the execution of an action. An example of an event is $(play, \{(object, m)\})$, i.e. the object m is played. If the time step is 3 minutes and the subject has been playing (using) the object for 2 minutes, resulting events in the trace are *((play, {(object, m)}),start)*, *((play, {(object, m)}),ongoing)*, *((play, {(object, m)}),ongoing)*. The protection system allows a set of traces according to the security policy. The security policy consists of declarations of events and a set of obligational (logical) formulas. Obligational formulas contain usage restrictions which prohibit certain usages under given circumstances, and mandatory actions that must be executed either unconditionally or after the specified usage has been performed. A security policy is specified in the OSL language. Syntax and semantics of OSL were formalized in Z, a formal language based on typed set theory and first-order logic with equality [48]. OSL operators are modified and generalized operators of a linear temporal logic. Since a concept of attributes and update actions are not defined explicitly in the distributed usage control, OSL operators are used to synthesize the expressiveness obtained by exploiting mutable attributes. As example, $repmax(5, E_{all}(play, \{(object, m)\}))$ using operator E_{all} requires that an object m must be played no more than 5 time steps.

The distributed usage control is an active area of research. Beyond the formalization, the distributed usage control studies offer models for the policy refinement [97], enforcement [96], negotiations [98], architectural solutions [21, 49].

UCON in Process Algebra

The process algebra is a perfect candidate for describing open distributed systems which deal with time and concurrency. Process algebra models operate with processes by algebraic means. A process describes the behavior of a system, i.e. all events or actions that a system can perform, the order in which they can be performed and possibly some other aspects such as timings, probabilities, etc. The process algebra lays down axioms on processes composition to describe a complex behavior of a system.

UCON in POlicy Language based on Process Algebra (POLPA). The POLPA language specification can be found in [16, 81, 82, 83, 58] and generally it presents a variant of Milner's Calculus of Communicating Systems (CCS). In POLPA, the UCON security policy is modeled as a process that defines the order in which usage control actions, authorization and condition predicates, obligation actions and attribute updates must be performed and when. Thus, the security policy is represented as a sequence of actions and POLPA offers several operators on sequences composition using the following grammar:

$$P ::= \bot|\top|\alpha(\overrightarrow{x}).P|p(\overrightarrow{x}).P|\overrightarrow{x} := \overrightarrow{e}.P|P_1 or P_2|P_1 par_{\alpha_1,...,\alpha_n} P_2|\{P\}|Z|$$

where P is a policy, $\alpha(\overrightarrow{x})$ is an action, $p(\overrightarrow{x})$ is a predicate, $\overrightarrow{x}$ is a variable, and Z is a constant process definition.

The formal semantics of POLPA is given in [82]. Informal semantics is the following:

- $\bot$ is *deny-All* operator;
- $\top$ is *allow-All* operator;
- $\alpha(\overrightarrow{x}).P$ is a sequential operator, and represents the possibility of performing an action $\alpha(\overrightarrow{x})$ and then behave as P;
- $p(\overrightarrow{x}).P$ behaves as P if a predicate $p(\overrightarrow{x})$ is true;
- $\overrightarrow{x} := \overrightarrow{e}.P$ assigns to variables $\overrightarrow{x}$ the values of the expressions $\overrightarrow{e}$ and then behaves as P;
- $P_1 or P_2$ is an alternative operator, and represents a non-deterministic choice between P_1 and P_2;
- $P_1 par_{\alpha_1,...,\alpha_n} P_2$ is a synchronous parallel operator. It expresses that both P_1 and P_2 must be simultaneously satisfied. This is used when two policies deal with actions in $\alpha_1, ..., \alpha_n$;

- $\{P\}$ is an atomic evaluation, and represents the fact that P is evaluated in an atomic manner, indeed once started must be completed. P here is assumed to have at most one action, predicates, and assignments. It allows the testing or update of variables prior or after an action;
- Z is the constant process, and there is a specification for the process $Z \doteq P$ and Z behaves as P;

Several derived operators may be defined:

- $P_1; P_2$ is a sequential operator, allows to behave P_1 and P_2 in a sequence;
- $i(P)$ behaves as the iteration of P zero or more times;
- $r(P)$ behaves as P running in parallel zero or more times;
- $P_1 par P_2$ is a parallel operator, and represents the interleaved execution of P_1 and P_2;

As a basic example, lets consider a security policy shown in Table 2.1 and which specifies the **preA0** core UCON scenario. This policy describes a process that starts with the $tryaccess(s, o, r)$ action executed by the subject, and if the p_A predicate is satisfied, the $tryaccess(s, o, r)$ action is followed by the $permitaccess(s, o, r)$ action which grants the access to the object. If the authorization predicate p_A is not satisfied, the execution of the $permitaccess(s, o, r)$ action is denied by the policy, and consequently, the system does not allow to access the object and terminates the policy enforcement. Otherwise, the $endaccess(s, o, r)$ action end the policy enforcement.

`tryaccess(s,o,r).`	line 1
`Pa.`	line 2
`permitaccess(s,o,r).`	line 3
`endaccess(s,o,r)`	line 4

Table 2.1: preA0 policy example

The POLPA language is expressive to model basic usage scenarios. They are distingue by positioning attribute update actions, obligations actions and predicates checks respectively to other predicates and actions. Before-usage (pre-authorization) policy statements are placed in between $tryaccess(s, o, r)$ and $permitaccess(s, o, r)$ action. Actions and predicates stated in between $permitaccess(s, o, r)$ and $endaccess(s, o, r)$ (or $revokeaccess(s, o, r)$) correspond to the ongoing usage. Actions occurred after $endaccess(s, o, r)$ (or $revokeaccess(s, o, r)$) are post-updates of attributes and these actions end the policy enforcement.

The formalization of UCON in POLPA has several benefits comparing to logic-based approaches. The POLPA policy states rigorously how the policy enforcement

should go and avoids ambiguity in the order of usage control actions, obligation and update actions executions, and predicates evaluation. Further, the POLPA language is very suitable to express the ongoing usage scenarios thanks to parallel composition operators. For instance, one might easily express in POLPA the execution of obligation actions, undate actions, and predicate checks running in parallel.

Timed Constrained Programming (TCP). Jagadeesan et al. presented a policy algebra in the timed concurrent constraint programming paradigm [52]. The formal model presented there is not compliant with the original UCON model but expressive enough to formalize usage control scenarios with pre- and ongoing authorization checks. Authorization predicates are considered as constrains on variables and tokens. The formal model addresses the history-based access control, and can express the explicit denial of access . It imposes temporal constrains on the evolution of the system depending on the past behavior and supports equational reasoning on security policies. The complete model details and specification are given in [52]. Details are omitted here, since the model can be potentially used for usage control, but yet is too far to capture UCON core models.

Comparing Models Expressiveness

The expressive power of the UCON model was greatly enhanced by the access decision continuity and attribute mutability comparing to traditional access control models. Table 2.2 examines UCON formal models presented in subsection 2.1.2 for compliance with these novelties. The symbol "+" denotes that the given formalism is qualified to express a concept, while "-" denotes the opposite. A formal model is *compliant with the original UCON*, if it is expressive enough to formalize the 24 UCON core scenarios. *Continuous usage decision* and *attributes updates* correspond to the novelties of usage control and both are a part of the original UCON model. *Specific post-obligations management* refers to formal models which formalize post-obligations. These are actions which must be fulfilled after the usage is terminated. Comprehensive usage scenarios may require the formal specification of *concurrent usage sessions*. *Explicit subject behavior* means that a formal model is capable to constrain subject's behavior explicitly by defining a sequence of actions allowed to execute on an object, i.e. to express a history-based access control. Certainly, subject's behavior can be modeled using attributes, but this approach essentially complicates the policy. *Explicit denial* shows whether a formalism can express not only permitted but also prohibited accesses explicitly.

2.1.3 Enforcement Level

Enforcement level answers how to facilitate a security policy enforcement. It aims to present an architecture of the UCON reference monitor, its components interfaces and authorization protocol.

Table 2.2: UCON formalization models

	Compliant with original UCON	Continuous usage decision	Attribute updates	Specific post-obligations management	Concurrent usage sessions	Explicit subject behavior	Explicit denial policy
TLA	+	+	+	-	-	-	-
$UCON_A$ with creation	-	-	+	-	+	-	-
ITL	+	+	+	-	+	+	+
OSL	-	+	-	+	-	+	+
POLPA	+	+	+	-	+	+	-
TCP	-	+	-	-	-	+	+

As proposed in [101], the UCON reference monitor consists of a *usage decision facility* (UDF/PDP) and a *usage enforcement facility* (UEF/PEP). In contrast with the reference monitor used in traditional access control, communications between the UDF/PDP and the UEF/PEP are not a state-less 'request-response'. In UCON, the UDF/PDP is always active and interactions between the UEF/PEP and the UDF/PDP are state-full. Thus, if the security policy is violated during the resource usage, the UDF/PDP generates a revocation event, and this event is enforced by the UEF/PEP terminating the usage process.

The UDF/PDP contains three basic components [90]: an *authorizations module*, a *conditions module* and an *obligations module*. The authorizations module and the conditions module control that authorization and condition predicates specified in the security policy are satisfied respectively. The obligations module is detailed in [55] and decides which obligation actions have to be fulfilled before or during the usage. The access decision is a conjunction of decisions evaluated by each module.

The UEF/PEP consists of three components too [90]: a *customization module*, a *monitoring module*, and an *update module*. The monitoring module is designed to monitor the fulfilment of obligation actions performed by an obligation subject, whereas the result of such monitoring is analyzed by the update module. The update module is responsible also for updates of subject's and object's attributes. The access decision received from the UDF/PDP can be either 'yes' or 'no', or contains a metadata information authorizing just a subset of requested access rights. This metadata information is forwarded to the UEF/PEP and customizes access rights in the *customization module*.

To classify UCON reference monitors, a number of factors can be taken into account. Some of them are enlisted below:

- *Location*: *client-* and *server-side reference monitor* (CRM and SRM). Here,

the server is the entity that provides the resource while the client requests and uses this resource. The SRM resides within the server system environment and mediates all accesses to the object. The CRM is settled on the client environment and controls accesses to copied digital objects (particularly digital information, e.g. text, media) on behalf of the server [101]. The SMR and the CRM may coexist to provide a better usage control. A location of the reference monitor is the most essential issue since it characterizes various usage scenarios and enforcement mechanisms;

- *Attribute acquisition*: push and/or pull models. Since the UCON access decisions are based on attributes, a crucial issue of the model enforcement is an obtaining of trusted and fresh attribute values. As a matter of fact, certain attributes can be either pushed by a subject with the initial request to the reference monitor, or the reference monitor pulls attributes from the system repository itself.

2.1.4 Implementation Level

Implementation level addresses real implementation of the UCON model which achieve the UCON security goals and is unfeasible to brake in a reasonable time.

Currently, the UCON model is not widely implemented in various computer environments. Although, there are several approaches in P2P, Grid computing (see subsection 2.2.8), operating systems, data base management systems, and mobile devices [120, 68, 8, 125, 82, 109, 126, 84, 79, 49, 115, 21, 77, 44].

2.2 State-of-the-art: Access and Usage Control in Grid

This section outlines the existing approaches of access and usage control in Grid.

The first generation of access control in Grid, the GridMapFile, pairs local accounts at the Grid resource provider machine to distinguished names (DN) of Grid users. In this case, the security policy is defined by the privileges paired with the local account and is enforced by the operating system running on the Grid resource provider host. The GridMap Authorization is a viable solution for intraGrid topology.

GridMap Authorization requires local accounts for possibly large number of Grid users. To relax these drawbacks, GSI also provides some other simple security mechanisms to make access control in Grid more flexible. These alternative mechanisms work out as a specific PDP and are classified to [37]:

- *None*: all requests are granted;

- *Self*: access is allowed if the requestor's identity equals to the resource's owner identity;
- *GridMapFile*: access control list mapping;
- *Host authorization*: access is allowed if the requestor presents a host credential that matches a specified hostname;
- *SAML Callout authorization*: allows to plug in an external third-party PDP. This PDP serves as the OGSA Authorization Service (see Figure 1.4) and support SAML assertions and protocol to specify and enforce access control policies.

Since the Globus Toolkit allows the adoption of external PDPs by means of *SAML Callout authorization*, many solutions have been emerged to improve access and usage control in Grid, and this section describes the main ones. These models vary on what information is used to produce an access decision, on a policy granularity and expressiveness. Whereas these models go far beyond the first generation of access control in the Globus Toolkit based on GridMapFile, they are still inadequate to address problems identified in the introduction chapter of this thesis.

2.2.1 CAS

The Community Authorization Service (CAS) [42, 94] is a Virtual Organization (VO) wide authorization service that has been developed by the Globus team. The main aim of CAS is to simplify the management of user authorization in Grid, i.e., to relieve the Grid resource providers from the burdens of updating their environments to enforce the VO security policies. CAS is a Grid service that manages a database of VO security policies, i.e., the policies that determine what each Grid user is allowed to do as a VO member on the Grid resources. In particular, the VO security policies stored by the CAS service consist of:

- the VO security policies about the Grid resources: this policies determine which access rights are granted to which Grid users;
- the CAS service's own access control policies, such as who can delegate access rights or maintain groups within the VO;
- the list of the VO members.

The local policies of the Grid resource providers, instead, are stored locally on the Grid nodes. Hence, the CAS service can be considered as a trusted intermediary between the VO users and the Grid resources. To transmit the VO policies to the Grid resources where they should be enforced, the CAS service issues to Grid users credentials embedding CAS policy assertions that specify the users' rights on

the Grid resources. The CAS assertions can be expressed in an arbitrary policy language. The Grid user contacts the CAS service to request a proper assertion to request a service on a given resource. The credentials returned by the CAS server will be presented by the Grid user to the service it wants to exploit. This requires that resource providers participating in a VO with CAS will deploy CAS-enabled service, i.e., services that are able to understand and enforce the policies expressed in the CAS assertions.

The CAS system works as follows:

1 The user authenticates himself to the CAS server, using his own proxy credential. The CAS server establishes the user's identity and the rights in this VO using its local database.

2 The CAS server issues a signed policy assertion containing the user identity and rights in the VO. On the user side, the CAS client generates a new proxy certificate for the user that embeds the CAS policy assertion, as non critical X.509 extension. This proxy is called *restricted proxy* because it grants only a restricted set of right to the user, i.e., the rights that are described in the CAS assertion it embeds.

3 The user exploits the proxy certificate with the embedded CAS assertion to authenticate on the Grid resource. The CAS-enable service authenticates the user using the normal authentication system. Then it parses the CAS policy assertion and takes several steps to enforce both VO and local policies:

   – Verifies the validity of the CAS credential (signature, time period, etc.);
   – Enforces the site's policies regarding the VO, using the VO identity instead of the user one;
   – Enforces the VO's policies regarding the user, as expressed in the signed policy assertion in the CAS credential;
   – Optionally, enforces any additional policies concerning the user (e.g., the user could be in the blacklist of the site).

Hence, the set of rights that are granted to the user is the intersection of the rights granted by the resource provider to the VO and the rights granted by the VO to the user, taking into account also specific restrictions applied by the resource provider to the user.

Once the access is authorized, the Grid user is then mapped on the local account paired with the CAS service. Hence, in the Grid resource GridMapFile there is only one entry that pairs the CAS distinguished name with the local account used to execute the jobs on behalf of the Grid users. This simplify the work of the local node administrator because he has to add one local account only for each CAS service, instead of one local account for each Grid user.

2.2.2 PERMIS

PERMIS is a policy-based authorization system proposed by Chadwick et al. [27, 110, 28] which implements RBAC in Grid. Instead of assigning certain permissions to a specific user directly, roles are created for various responsibilities and access permissions are assigned to specific roles possessed by the user. The assignment of permissions is fine-grained in comparison with ACLs, and users get the permissions to perform particular operations through their assigned role. PERMIS is based on a distributed architecture, that includes the following entities: i) Sources of Authority, that are responsible for composing the rules for decision making and credential validation services; ii) Attribute Authorities, that issue the attributes that determine the role of users; iii) Users, that are the principals that perform operations on the resources; and iv) Applications, that provide to the users the interfaces to access the protected resource. Obviously, users and resources can belong to distinct domains. A policy file written in XML contains the full definition of roles in regard to protected resources and permissions related to a specific role. The PERMIS toolkit provides a friendly graphical user interface for managing its policies. The policies may be digitally signed by their authors and stored in attribute certificates, in order to prevent them from being tampered. PERMIS is based on the Privilege Management Infrastructure (PMI) that uses X.509 attribute certificates (ACs) to store the user's roles. Every AC is signed by the trusted Attribute Authority (AA) that issued it, whilst the root of trust for the PMI target resource is called the Source of Authority (SOA). All the ACs can be stored in one or more LDAP directories, thus making them widely available.

PERMIS also provides the Delegation Issuing Service (DIS), which allows users to delegate (a subset of) their privileges to other users in their domain by giving them a role in this domain, according to the site's delegation policy. Since PERMIS is tightly integrated with the Globus Toolkit, authorization information for an access decision consists of user's DN, resource and action request. For authorization decision making, PERMIS provides a modular PDP and a credential validation service (CVS, or PIP according to the Globus model). PERMIS implements the hierarchical RBAC model, which means that user roles (attributes) with superior roles inheriting the permissions of the subordinate ones. The PERMIS policy comprises two parts, a role assignment policy (RAP) that states who is trusted to assign certain attributes to users, and a target access policy (TAP) that defines which attributes are required to access to what resources and under what conditions. CVS evaluates all received credentials against the RAP, rejects untrusted ones, and forwards all validated attributes to the PEP. The PEP in turn passes these to the PERMIS PDP, along with the user's access request, and some environmental parameters. The PDP obtains an access control decision based on the TAP, and sends its granted or denied response back to the PEP. Hence, in order to gain access to a protected target resource a user has to present his credentials and the PERMIS decision engine (CVS and PDP) validate them according to the policy in order to make an access decision. Current

version of PERMIS Authorization Service supports SAML Authorization Callout and provides Java APIs for accessing CVS and PDP. Technical specifications and implementation issues can be found in [4].

2.2.3 Akenti

The paramount idea of Akenti proposed by Thompson et al. [112, 111, 1] is to provide a usable authorization system for environment consisting of highly distributed resources shared among several stakeholders. By exploiting fine-grained authorization for job execution and management in Grid, Akenti provides a restricted access to resources using access control policy which does not require a central administrative authority to be expressed and to be enforced.

In this model, control is not centralized. There are several stakeholders (parties with authority to grant access to the resource), each of which brings its own set of concerns in resource managing. Access control policy for a resource is represented as a set of (possibly) distributed X.509 certificates digitally signed by different stakeholders from unrelated domains. These certificates are independently created by authorized stakeholders and can be stored remotely or on a known secure host (probably the resource gateway machine). They are usually self-signed and express what attributes a user must have in order to get specific rights to a resource, who is trusted to make such use-condition statements and who can certify user's attributes. Akenti policy is written in XML and there exists three possible types of signed certificates: Policy certificates, Use-condition certificates and Attribute certificates. Use-condition certificates contain the constraints that control access to a resource and specify who can confirm to the required user's attributes and thus who may sign Attribute certificates. Attribute certificates assign attributes to users that are needed to satisfy the usage constraints. Complete policies specification and language used to express them can be found on Akenti web-site [1]. When an authorization decision is required, the resource gatekeeper asks a trusted Akenti server what access the user has to the resource. Then Akenti policy engine gathers all the relevant certificates for the user and for the resource from the local file system, LDAP servers and web servers, verifies and validates them, and responses the user's rights in respect to the requested resource. Akenti assumes secure SSL/TLS connection between peers and resource through the resource gateway which provides authentication using X.509 identity certificate. Authorization algorithm in Grid using Akenti is very similar to PERMIS and has the following stages:

1 A resource provider authenticates a requestor and validates his identity as well as possibly some additional attributes.

2 The resource provider receives and parses the user's request.

3 The resource provider forwards the user's identity, attributes, and requests to trusted Akenti server to authorize user (i.e., whether request should be granted

or denied)

4 Finally, Akenti returns a decision to the resource provider

2.2.4 Shibboleth

Shibboleth [116, 5] , is an Internet2/MACE project implementing cross-domain single sign-on and attribute-based authorization for systems that require inter-institutional sharing of web resources with preservation of end user privacy. The main idea of Shibboleth is that instead of having to login and be authorized at any restricted site, users authenticate only once at their local site, which then forwards the user's attributes to the restricted sites without revealing information about user identity.

The main components of the Shibboleth architecture are: Handle Service which authenticates users in conjunction with a local authentication service and issues an handle token; Attribute Service presents handle token when a user requests to access a resource. The resource, in turn, presents the user's handle token to the Attribute Service and requests the attributes of the user. Target Resource: includes shibboleth specific code to determine the user's home organization and, consequently, which Shibboleth attribute authority should be contacted for this user. A typical usage of Shibboleth is as follows:

1 The user authenticates to Shibboleth Handle Service (SHS);

2 SHS requests local Organizational Authentication Service by forwarding user authentication information to confirm his identity;

3 SHS generates a random handle and maps it to user identity. This temporal handle is registered at Attribute Service;

4 The handle is returned to the user and notified that he was successfully authenticated;

5 Then the user sends a request for a target resource with the previous handle;

6 The resource provider analyzes the handle to decide which Shibboleth service may provide the required user attributes to make authorization decision, and contacts it by forwarding the handle that identifies the user;

7 After validation checks on the handle have been done and the user's identity is known, the Attribute Release Policy is used to determine whether the user attributes can be sent to the resource provider;

8 The Shibboleth Attribute Authority casts the attributes in the form of a SAML attribute assertions and returns these assertions to the target resource;

9 After receiving the attributes, the target resource provider performs an authorization decision with regard to the user's request, attributes and resource access control policy.

Detailed specification of all Shibboleth's functional components like Identity Provider, Service Provider, etc. and used security protocol based on SAML can be found in [5]. GridShib [3, 26] is a currently going research project that investigates and provides mechanisms for integration Shibboleth into Globus Toolkit. The focus of the GridShib project is to leverage the attribute management infrastructure of Shibboleth, by transporting Shibboleth attributes as SAML attribute assertions to any Globus Toolkit PDP.

2.2.5 VOMS

The Virtual Organization Membership Service (VOMS), [12, 6], is an authorization service for Grid that has been developed by the EU projects DataGrid and DataTAG. VOMS has a hierarchical structure with groups and subgroups; a user in a VO is characterized by a set of attribute, 3-tuples of the form group, role, capability. The combined values of all these 3-tuples form a unique attribute, the Fully Qualified Attribute Name (FQAN), that is paired to the Grid user. VOMS is implemented as a push system, where the Grid user first retrieves and then sends to the Grid service the credentials embedding the attributes he want to exploit for the authorization process. The VOMS system consists of the following components:

- User Server: it is a front end to a data base where all the information about the VO users is kept. It receives requests from the client and returns information about the user;
- User Client: contacts the server presenting the certificate of a user and obtains the list of groups, roles and capabilities of that user;
- Administration Client: used by the VO administrators to add users, create new groups, changing roles, and so on;
- Administration Server: accepts the requests form the client and updates the Database.

To retrieve the authorization information the VO grants him, the Grid user exploits the VOMS User Client that contacts the VOMS User Server. The VOMS server returns a data structure, called VOMS *Pseudo-Certificate* or *Attribute Certificates*, embedding the user's roles, groups and capabilities. The pseudo-certificate is signed by the VOMS User Server and it has a limited time validity. If necessary, more than one VOMS User Server can be contacted to retrieve a proper set of credential for the Grid user. To access a Grid service the user creates a proxy certificate

containing the pseudo-certificates that he has previously collected from the VOMS Servers.

The Grid node, to perform the authorization process, extracts the Grid user's information from the user's proxy certificate and combines them with the local policy. Since using VOMS the Grid resource is accessed exploiting the Grid user name, i.e., the Distinguished Name in the user's certificate, the user name should be added in the gridmap file of each Grid resource and paired with a local account. To this aim, the Grid resource provider periodically queries VOMS databases to generate a list of VO users and to update the gridmap file mapping them to local accounts.

2.2.6 Cardea

Cardea is a distributed authorization system developed as part of the NASA Information Power Grid [73]. One of the key features of Cardea is that it evaluates authorization requests according to a set of relevant characteristics of the Grid resource and of the Grid user that requested the access, instead of considering the user's and resource's identities. Hence, the access control policies are defined in terms of relevant characteristics rather than in terms of identities. In this way, Cardea allows users to access Grid resources where they don't have existing local accounts. Moreover, Cardea is a dynamic system, because the information required to perform the authorization process are collected during the decision process itself. Any characteristic of the Grid user or of the Grid resource, as well of the ones of the current environment, can be taken into account in the authorization process. These characteristics are represented through SAML assertions, that are exchanged through the various components of the architecture.

From the architectural point of view, the Cardea system consists of the following components: a SAML Policy Decision Point (SAML PDP), one or more Attribute Authorities (AA), one or more Policy Enforcement Points (PEP), one or more references to an Information Service, an XACML context handler, one or more XACML Policy Administration Points, and a XACML Policy Decision Point (XACML PDP). The main component of the system is the SAML Policy Decision Point, that accepts authorization queries, performs the decision process and returns the authorization decision. To exploit Cardea in the existing Grid toolkits, proper connectors, e.g., an authorization handler in the case of the Globus toolkit, generate the authorization query in the format accepted by the SAML PDP. The SAML PDP, depending on the request, determine the correct XACML PDP to evaluate the request. The values of the attributes involved in the authorization request are retrieved querying the appropriate Attribute Authorities. Finally, the PEP is the component that actually enforces the authorization decision, and could even reside in a remote Grid node. Hence, the final authorization decision is transmitted by the SAML PDP to the appropriate PEP to be enforced.

The components of the Cardea system could be located on the same machine,

and in this case their interactions are implemented through local communication paradigms, or they could be distributed across several machines, and in this case they act as web services.

2.2.7 PRIMA

PRIMA (PRIvilege Management and Authorization) [78] is focused on management and enforcement of fine-grained privileges. PRIMA enables the users of the system to manage access to their privileges directly without the need for administrative intervention. The model uses on-demand accounts leasing and implements expressive enforcement mechanisms built on existing low-overheard security primitives of the operating systems.

PRIMA addresses the security requirements through a unique combination of three innovative approaches: *privileges*: unforgeable, self-contained, fine-grained, time limited representations of access rights externalized from the underlying operating system, privileges management is pushed down to the individuals in PRIMA; *dynamic policies*: a request-specific access control policy formed from the combination of user provided privileges with a resources access control policy; *dynamic execution environments*: a specifically provisioned native execution environment limiting the use of a resource to the rights conveyed by user-supplied privileges.

PRIMA authorization system could be divided in two parts. The first part is the privilege management layer which facilitates the delegation and selective use of privileges. The second part is the authorization and enforcement layer. The authorization and enforcement layers have two primary components. The first component is the PRIMA Authorization Module. The Authorization Module plays the role of a PEP. The second component is the PRIMA PDP which, based on policies made available to it, will respond to authorization requests from the PRIMA Authorization Module. These policies are created using platform independent language XACML. Two other components in the authorization and enforcement layer are the Gatekeeper and the Privilege Revocator. The Gatekeeper is a standard Globus Toolkit component for the management of access to Globus resources. It was augmented with a modular interface to communicate with the authorization components. The JobManager, also a standard component of the Globus Toolkit, has not been modified from the original Globus distribution. It is instantiated by the Globus Gatekeeper after successful authorization. It starts and monitors the execution of a remote users job. The Privilege Revocator monitors the lifetime of privileges that were used to configure execution environments. On privilege expiration, the Privilege Revocator removes access rights and de-allocates the execution environment automatically. No manual intervention from system administrators is required.

A typical access request in PRIMA authorization system goes as follows. Step 1, the delegation of privileges and provision of policies, happens prior to a request is issued. In step 2, subjects select the subset of privilege attributes they hold

for a specific (set of) Grid request(s) and group these privileges with their short lived proxy credential using a proxy creation tool. The resulting proxy credential is then used with standard Globus job submission tools to issue Grid service requests (step 3). Upon receiving a subjects service request, the gatekeeper calls the PRIMA authorization module (step 4). The PRIMA authorization module extracts and verifies the privilege attributes presented to the Gatekeeper by the subject. It then assembles all valid privileges into a dynamic policy. Dynamic policy denotes the combination of the users privileges with the resources security policy prior to the assessment of the users request. To validate that the privileges were issued by an authoritative source, the Authorization Module queries the privilege management policy via the PRIMA PDP. The multiple interactions between authorization module and PDP are depicted in a simplified form as a single message exchange (step 5 and 6). Once the privileges issuer authority is established, the PRIMA Authorization Module formulates an XACML authorization request based on the users service request and submits the request to the PDP. The PDP generates an authorization decision based on the static access control policy of this resource. The response will state a high-level permit or deny. In the case of a permit response, the Authorization Module interacts with native security mechanisms to allocate an execution environment (e.g., a UNIX user account with minimal access rights) and provision this environment with access rights based on the dynamic policy rules (step 7). Once the execution environment is configured, the PRIMA Authorization Module returns the permit response together with a reference to the allocated execution environment (the user identifier) to the Gatekeeper and exits (step 8). The following steps are unchanged from the standard Globus mechanisms. The Globus Gatekeeper spawns a JobManager process in the provided execution environment (step 9). The JobManager instantiates and manages the requested service (step 10). In the case of a deny response, the Authorization Module returns an error code to the Gatekeeper together with an informative string indicating the reason for the denied authorization. The Gatekeeper in turn will protocol this error in its log, return an error code to the Grid user (subject) and end the interaction. The Privilege Revocator watches over the validity period of dynamically allocated user accounts and all fine-grained access rights, revoking them when the associated privileges expire (step 11).

In summary, PRIMA mechanisms enable the use of fine-grained access rights, reduce administrative costs to resource providers, enable ad-hoc and dynamic collaboration scenarios, and provide improved security service to long-lived Grid communities.

2.2.8 Sandhu's Approach for Collaborative Computing

The authors of UCON recognized the usefulness of their model also for collaborative computing systems (and hence also for Grid) and published an initial work in the area [125, 126].

The access and usage control framework was designed to protect a shared trusted storage for a collaborative management of an application code by several developers from different locations. It was enhanced to address attribute mutability (e.g. developer's location is a mutable attribute) and continuity of control. A security policy stated exactly the need of continuous control and access rights were determined by authorization predicates, condition predicates and obligations.

Authors provides an architecture and initial implementation of a reference monitor. The reference monitor consists of user platforms, resource providers (RPs), and an attribute repository (AR). AR is a centralized trusted service to store and push mutable subject and system attributes to PDP. Object attributes are stored in a usage monitor (UM) on each RP side. A usage session is initialized when a user submits an access request from its platform to RP (step 1). Persistent subject attributes are pushed with the access request to the PDP (step 2). After receiving the request, PDP contacts AR and retrieves mutable attributes of the requesting subject (steps 3 and 4) and the object attributes from UM (step 5). Since the user may have interest on showing good values for mutable attributes, the PDP exploit a push model for immutable while the pull model for mutable attributes to ensure that PDP has always fresh information. By the way, this hybrid scenario is time sensitive and can pose security overhead. An access control decision is issued by PDP after collecting all related information (subject, object, and system attributes) and evaluation of security policies. The access decision is forwarded to PEP and enforced in the execution environment of RP (step 6). As the side effect of making the access decision, attribute updates are preformed by PDP according to corresponding security policy. Updated attributes are sent back to AR (step 7), and object attributes - to UM (step 8). When the access is in progress, any update of subject or object attributes and any change of system conditions triggers the reevaluation of the policy by PDP according to the ongoing usage session and may result in revocation of the ongoing usage or updates of attributes.

The prototype was implemented as the server-side reference monitor (both PDP and PEP were placed on RD side). The implementation did not realize all capabilities of the model and was limited to handle a continuity of control, access revocation, ongoing attribute updates. The reference monitor enforced usage control policy written in XACML policy language [119]. By the way, XACML is not expressive enough to define the original UCON model completely. It was noted in [125, 126], that XACML is only capable to specify attribute requirements before usage and possible updates after the usage, but not during the usage.

Chapter 3

Access and Usage Control Model in Grid

This Chapter described our access and usage control model for Grid services based on the UCON model. The UCON model is considered as a good candidate to express comprehensive usage scenarios occurring in Grid due to the UCON's peculiarities, a *continuity of control* and *mutability of attributes*. In fact, Grid services are long-lived and initial conditions which grant to a requestor some access rights over the Grid service can change during the service execution. This requires a continuous access reevaluation and the policy violation can lead to the access revocation and the service termination.

That Chapter covers the *policy level* and introduces Grid model abstraction which outlines objects to be protected on coarse- and fine-grained levels of control (section 3.1). Section 3.2 describes U-XACML *policy specification language* which enhances XACML in order to express core UCON access and usage control scenarios. Section 3.3 shows a *policy formal model* based on POLPA process algebra language. This Chapter ends giving several examples of security policies on coarse- and fine-grained levels of control (subsection 3.4).

3.1 Grid Model Abstraction

Grid is a collaborative computational environment which is operated by several participants: Grid user, resources providers and Grid administrators. To satisfy security requirements of Grid participants, our access and usage control framework proposes two levels of control: a *coarse-grained level* of control that manages access and usage to Grid service instances, and a *fine-grained level* that monitors the usage of underlying resources allocated for these service instances. Both levels are extended with the UCON's peculiarities, a continuity of control for long-lived services and mutability of attributes as a side effect of the usage process.

3.1.1 Coarse-grained Level of Control

A computational Grid exposes a set of Grid services to facilitate efficient execution of jobs. The coarse-grained level of control in our framework enforces security policies over *Grid services* and also specifies an authorized *workflow of Grid services*.

A Grid service implementation should be compliant with WS-Resource Framework specification and it extends the concept of a Web service with the notion of a *state-full service* [41]. Indeed, a Grid service is composed of a Web service and a state-full resource. The Web service part provides interfaces to access the resource, while the resource is assigned to perform some computational tasks. The resource can be created, destroyed and accessed several times during its life-cycle. Figure 3.1 gives a representation of a Grid service.

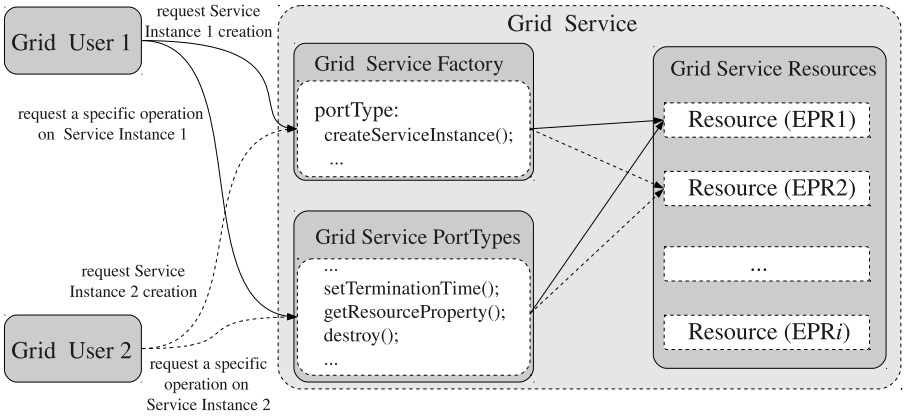

Figure 3.1: Grid Service

When a Grid user wants to execute a job in Grid, it invokes a factory of a computational Grid service. The factory service creates the new service instance, i.e. a state-full resource. The service instance models a process that executes the submitted job. During the execution, the service instance can be contacted using Grid Service PortTypes to check the status of the execution. The service instance sends a notification to the Grid user once it has finished executing. Then, the service instance can be destroyed. If access to create and use the Grid service instance is based on mutable attributes which might change during the service instance execution, the continuous control is required.

Each Grid service instance is uniquely identified. It is identified by the end point reference (EPR) which consists of Grid Service Handler (GSH) - abstract name, and Grid Service Reference (GSR) - used to communicate with the named Grid service instance.

All main services used to facilitate execution of jobs in computational Grid, i.e. GRAM, MDS, and GridFTP service, are state-full services. Instances of these services are basic *objects* to be protected on the coarse-level of control. Grid users

are *subjects*, and *access rights* are permissions to create Grid service instances and invoke their methods.

A computational Grid is formed by a number of distinct Grid service instances running together. Grid users might compose Grid services on their discretion to achieve some computational goals (e.g. see the usage example of the computational Grid given in section 1.1 and Figure 1.3). Composition of Grid services instances on behalf of a Grid user is named as a *service workflow*. In computational Grids the service workflow usually helps to facilitate the execution of heavy computational jobs. Each job can be decomposed on several parts and each part executed in parallel by a single computational service. Service workflows are another type of objects to be controlled on the coarse-grained level of control.

3.1.2 Fine-grained Level of Control

Each Grid service instance runs on its own processor, i.e. a Grid resource provider machine, and has an internal process control. The fine-grained level of control in our framework is enhanced with a flexible and efficient continuous control over RPs resources and allows to specify limited access rights to resources which host Grid service instances.

Grid users invoking Grid services hosted on RP's resources are *subjects* on this level of control. RP's resources like software installed on the RP platform, data files, storage, CPU, memory, network connectivity are *objects*. *Access rights* allow an invocation of software, an opening a file, a creation of a network socket, etc. Execution of each access rigts self is considered as a long-standing activity (e.g. an action which allows to make a socket connection). If access to perform such actions is based on mutable attributes which might change during the action execution, the continuous control is required.

Since a Grid computational service executes jobs submitted by remote (and possibly malicious) Grid users on the local resource, the fine-grained level of control addresses also a history-based security. Instead of considering the execution of a job as a single atomic action, we split down the monitoring in basic actions performed by a job during its execution. In particular, since we are interested in the interactions with the underlying resource, the actions we monitor are the system calls that jobs invoke on the operating system level. Hence, the sequence of actions performed by a job during its execution defines the behavior of the job itself. This sequence is not deterministic, because it may depend on various factors, such as specific input values. The model is history-based because the actions that a job is allowed to perform at a given point of its execution depend on its past behavior, i.e. on the sequence of actions previously executed by the job itself. Hence, a given action a could be allowed only if some other actions have (not) been already executed.

3.1.3 Integrating Two Levels of Control with Shared Attributes

Two levels of control are integrated by sharing attributes used to built security policies. A security policy that specifies mapping rules between attribute on different level of control is called a *property policy*. This policy is written as normal logic programs [14] and discussed in section 4.1.1.

Examples of coarse-grained attributes are Grid user location, reputation, balance, Grid roles, number of Grid service instances created by a specific Grid user, high-level attributes of Grid resource providers, etc. Examples of fine-grained attributes are number of resources consumed by a specific Grid service instance, e.g. number of created files, number of bytes transmitted via network, etc.

Attributes obtained at the coarse-grained level are taken into account for access decisions at the fine-grained level and vise versa. From security point of view, the integration of the two levels guarantees that Grid users exploit fine-grained resources in compliance with access rights granted on the coarse-grained level. Continuous control on both levels imposes that a violation of a security policy either on coarse- or fine-grained level triggers a termination of a Grid service instance.

The enforcement of security policies depends on types of attributes used in the policy specification. For these purposes, we classify attributes based on *mutability reason* and *provenance*. The mutability reason answers why attribute can change its value and launches the following attribute types:

- *Immutable*: these attributes are static during the usage and can be change only by the administrative actions;
- *Enforceable mutability*: these attributes are changed as the result of a policy enforcement only, and this changes are encoded into the policy through *update* actions;
- *Observable mutability*: these attributes are mutable by their nature, but how they change is not stated explicitly in a security policy. Reference monitor enforcing the security policy may only observe these attributes.

The provenance denotes where attributes were issued and is divided on:

- *Local*: these attributes are created in a security domain where the policy is enforced;
- *Remote*: these attributes are created in one security domain, but can be used in others.

In fact, coarse-grained attributes are usually considered as *remote* and with *observable mutability* regarding the fine-grained level of control and vise versa. By the way, this might hold for any attribute on any level of control, e.g. an attribute

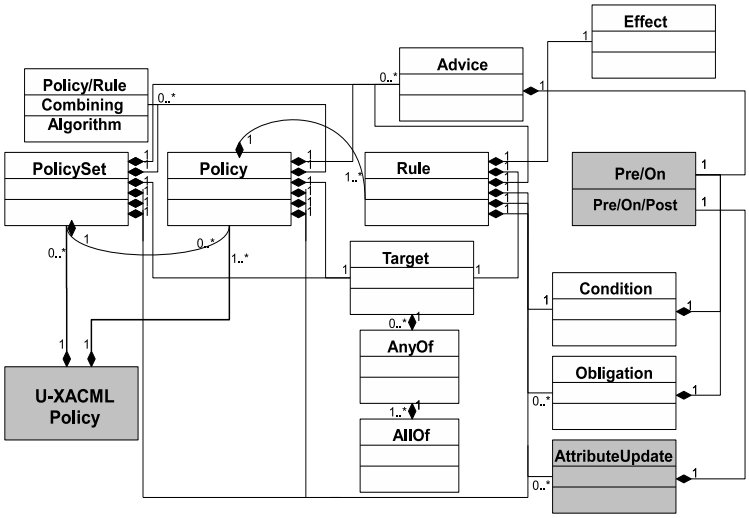

Figure 3.2: U-XACML Policy Meta-model

which specifies a location of a Grid user also is *remote* and with *observable mutability* regarding the coarse-grained level of control. In order to produce credible access decisions, reference monitor should understand the semantics of such attributes and trust to authorities issuing them.

3.2 Policy Specification Language: U-XACML

In this section, we propose U-XACML policy specification language, an enhancement of the XACML with features of usage control, i.e continuity of an access decision evaluation and mutability of attributes. U-XACML is capable to specify core UCON scenarios.

XACML [119] is the most widely-used example of the common purpose access control policy language. Attributes of arbitrary types can be expressed which makes the XACML language application-independent and extensible to accommodate specific requirements of specific applications and domains. Currently, the OASIS standards organization is working on the XACML specification, and several implementations have been presented by third-party vendors[1].

XACML is facilitated to express traditional access control models, where an access decision is evaluated only once when a request to access a resource comes. Also, the XACML policy can express obligations, a set of actions performed in conjunction with the access decision enforcement. However, the current version of XACML has insufficient facilities to express a continuous control afterwards the access was granted and started.

[1]http://sunxacml.sourceforge.net, http://mvpos.sourceforge.net

`subject`, `object` and `access right` of the UCON model are represented in XACML (and U-XAMCL) by `subject`, `resource` and `action` respectively. Subjects, resources and environment are associated with a set of attributes described with the following elements: `<AttributeDesignator>` (states to whom and by whom the attribute is issued), `<AttributeSelector>` (states where the attribute can be found), and `<AttributeValue>` (contains an attribute value).

The U-XACML policy meta-model slightly extends the original XACML and is drawn in Figure 3.2. The top-level policy elements are `<PolicySet>`, `<Policy>`, and `<Rule>`.

The `<Rule>` has three main parts: `<Target>` which denotes rule's applicability to an access request, `<Condition>`s which are predicates over attributes, and `<Effect>` is the result of the access decision evaluation. It returns either "Permit" or "Deny" if the rule is satisfied and "Non Applicable" if the `<Target>` and/or `<Condition>`s are not satisfied. Authorizations and conditions proposed in UCON are modeled in U-XACML by means of the `<Target>` and `<Condition>` elements. `<Target>` element puts constraints on the immutable attributes only, while `<Condition>` elements cover mutable attributes of a subject, object and environment.

The XACML (and U-XACML) `<Policy>` consists of one or more `<Rule>` elements. An access decision implied by the `<Policy>` is a combination of the result of the evaluation of each `<Rule>` it contains. XACML identifies several combining algorithms: deny-overrides, permit-overrides, first-applicable and only-one-applicable. The access decision produced by the `<Policy>` is accompanied with a set of `<Obligations>`.

The `<PolicySet>` is an optional element which provides the resulting policy through the combining of several `<Policy>` elements applicable to the access request.

In the UCON access decision should be evaluated not only before granting the access but also continuously when the access is in progress. The U-XACML policy specifies when the access decision is made by the `DecisionTime` in the `<Obligation>` and `<Condition>` elements. For example, the `<Condition>` element should include the following:

```
<xs:element name="Condition"/> <xs:complexType
name="ConditionType">
    ...
    <xs:attribute name="DecisionTime" type="xs:string" use="required"/>
    ...
</xs:complexType>
```

`DecisionTime` has values `pre` and `on`. For the `pre` models, the conditions are evaluated only once, while for the `on`-going models they have to hold continuously.

To represent attribute updates, we define a new element, `<AttrUpdates>`, that contains a collection of `<AttrUpdate>` elements:

```
 <xs:element name="AttrUpdates"
type="u-xacml:AttrUpdatesType"/> <xs:complexType
```

```
name="AttrUpdatesType">
    ...
    <xs:sequence>
        <xs:element ref="u-xacml:AttrUpdate" maxOccurs="unbounded"/>
    </xs:sequence>
    ...
</xs:complexType>
```

Each <AttrUpdate> refers to a distinct attribute and defines a single update action:

```
<xs:element name="AttrUpdate" type="u-xacml:AttrUpdateType"/>
<xs:complexType name="AttrUpdateType">
    ...
    <xs:sequence>
        <xs:element ref="u-xacml:UpdateExpression" minOccurs="0"/>
    </xs:sequence>
    ...
    <xs:attribute name="UpdateTime" type="xs:integer" use="required"/>
    ...
    <xs:element ref="TriggerOn"/>
    ...
</xs:complexType>
```

The UpdateTime element defines when an update action must be preformed and has values 1, 2, or 3 that denote, respectively, pre-, on- and post-updates. The <UpdateExpression> element is a specific update function which is used to compute a new value of the attribute.

When access is in progress, an invocation of obligation and update actions is triggered when some conditions hold. The <ObligationExpression> and <AttrUpdate> elements include the <TriggerOn> element which identifies conditions which trigger the update and obligation actions:

```
<xs:element name="ObligationExpression"/> <xs:complexType
name="ObligationExpressionType">
    ...
    <xs:element ref="TriggerOn"/>
    ...
</xs:complexType>
```

The <TriggerOn> has the same syntax as the <Condition> element.

These are the minimal required modifications to the XACML language specification in order to capture attribute updates, continuous policy evaluation, and to place conditions triggering ongoing attribute updates and obligations. These results are promising and show how relatively easily the existing standards in access and usage control have to be extended to express the UCON model.

3.3 Policy Formal Model: POLPA

The U-XACML policy specification language is capable to express core UCON scenarios which control the execution of a *single* long-standing access action. In fact,

our access and usage control model surmises to specify also history-based security control, i.e. sequences of actions, where each action is also the long-standing execution. Thus, we exploited POLPA policy language (rf. section 2.1.2) to express formally security policies on two levels of control. The POLPA language represents a security policy as a sequence of actions and is powerful to express UCON novelties too.

We follow a similar approach to [127] and consider a set of actions that model the potential activities involved in the UCON process. Every action refers to an access request where a subject s wants to access an object o through an operation that requires the right r. Given that the triple (s, o, r) represents the access request, we consider the following actions:

- `tryaccess(s,o,r)`: performed by subject s when performing a new access request (s, o, r);
- `permitaccess(s,o,r)`: performed by the reference monitor when granting the access request (s, o, r);
- `denyaccess(s,o,r)`: performed by the reference monitor when rejecting the access request (s, o, r);
- `revokeaccess(s,o,r)`: performed by the reference monitor when revoking an ongoing access (s, o, r);
- `endaccess(s,o,r)`: performed by the execution environment or the subject when ending the access (s, o, r);
- `update(attr)`: performed by the reference monitor to update the attribute;
- `Pa` ($\overline{P_a}$): performed by the reference monitor when authorization predicates based on attributes are (not)satisfied;
- `Pb` ($\overline{P_b}$): performed by the reference monitor when obligation actions are (not)fulfilled;
- `Pc` ($\overline{P_c}$): performed by the reference monitor when condition predicates based on environmental attributes are (not)satisfied.

An access request to create and use a computational Grid service on the coarse-grained level of control is specified via

```
tryaccess(gridUser,GRAMservice,createManagedJob)
```

and action

```
endaccess(gridUser,GRAMservice,createManagedJob)
```

is used to represent that the execution of the submitted job has ended and the corresponding service instance is destroyed.

A fine-grained security policy, that regulates the interactions with the local resources of jobs executed by the computational service instance on the behalf of the remote Grid user, specifies the access request via

```
tryaccess(app_id,socket,accept(sd,addr,addrlen,newsd))
```

where the job `app_id` tries to access the `socket` resource to execute the operation `accept` to wait for an incoming network connection. An attribute of the job `app_id` is the distinguished name of the Grid user that submitted it to the computational resource. Instead, the action

```
endaccess(app_id,socket,accept(sd,addr,addrlen,newsd))
```

is used to represent that the execution of the access previously described has been terminated.

3.4 Security Policy Examples

This subsection presents several examples of security policies specified using the POLPA formal language. The usage control scenarios encoded in these policies are based on experiences from previous work in access and usage control in Grid and interactions with the EU GRIDTRUST project participants. The GRIDTRUST project was launched to improve Grid security and clearly revealed the necessity of more sophisticated security services to adequately support continuous control over Grid services in a variety of collaborative scenarios.

3.4.1 Example 1. Coarse-grained Security Policy

The table 3.1 gives an example of the course-grained policy informally expressed in the introduction Chapter. The usage of a Grid service instance `service1` created by an `user1` is allowed only for 20 seconds and if the `user1` in the meanwhile never tries to access the service `s2`.

The first line of the policy states the access request received from the `user1` to call the method `createManagedJob` of the `service1`. The predicate in the second line checks if the `service2` is not invoked by the `user1`. The `invokedServ` is the `user1`'s (subject) attribute containing a set of active services running on behalf of the `user1`. If the predicate is satisfied, the set of active services should be updated by inserting the `service1` identifier (line 3). The predicate evaluation and the attribute update are run before granting the access. Further, the system creates the `service1` instance and the ongoing usage control phase starts, the `permitaccess(user1,service1,createManagedJob)` action is performed. The service execution `time` is bounded by 20 seconds what is stated by on-authorization

`tryaccess(user1,service1,createManagedJob).`	line 1
[`(service2` $\notin$ `user1.invokedServ)`].	line 2
`update(user1.invokedServ = user1.invokedServ` $\cup$ `service1).`	line 3
`permitaccess(user1,service1,createManagedJob).`	line 4
`( ([(service1.time` $>$ `20s) or (service2` $\in$ `user1.invokedServ)`].	line 5
`revokeaccess(user\1,service1,createManagedJob)`)	line 6
`or`	line 7
`endaccess(user1,service1,createManagedJob))`	line 8
`).`	line 9
`update(user1.invokedServ = user1.invokedServ` $\setminus$ `service1)`	line 10

Table 3.1: Coarse-grained Security Policy 1

predicate in the line 5. Also, it specifies that during `service1` usage, the `user1` is forbidden to call any method of the `service2`. Otherwise, the access should be revoked and execution of the `service1` terminated (line 6). If the policy is not violated during the usage, the user can end the service normally (line 8). After the usage, the `invokedServ` attribute is updated since the `service1` instance does not exist anymore. Notice, that the `invokedServ` attribute is a multi-set and may contains several equal elements, while the post-update action removes just one entry of the `service1`.

3.4.2 Example 2. Coarse-grained Security Policy

The table 3.2 gives an example of a policy which contains 5 rules:

- Grid user's reputation should be higher the threshold value before the usage (pre-authorization). If it is below the threshold during the execution, the access should be revoked and the application terminated (on-authorization). After the usage, the resource provider updates Grid user's reputation (post-update). If the service was ended normally by the user, the reputation should be increased, while if the access was revoked by the system, than the reputation should be decreased. This rule refers to **onA3** model;

- No more that 5 applications can run on behalf of the user on the resource provider node. If the number exceeded 5, than the access should be denied (pre-authorization). The update of the number of running applications is required before starting a service (pre-update) and when the usage is over (post-update). This rule refers to **preA13** model;

- Grid user has to sign an agreement, e.g. that application is not malicious, before submitting it to the execution. This rule refers to **preB0** model;

- The application submitted for execution can exploit computational resources for a particular time quota. If during the usage, application's execution time

```
tryaccess(user1,service1,createManagedJob).                line 1
[(user1.reputation > 10) ∧ (user1.numOfAppl ≤ 5)].          line 2
Pb('user1 should sign agreement').                         line 3
update(user1.numOfAppl += 1).                              line 4
permitaccess(user1,service1,createManagedJob).             line 5
Z.                                                         line 6
update(user1.numOfAppl -= 1)                               line 7

where

Z = (                                                      line 8
  ((                                                       line 9
    ({[(service1.timeQuota + 1h > environment.currTime)]. line 10
    Pb('send notification')})                              line 11
  par                                                      line 12
    ({[(user1.credit > 0)].                                line 13
    update(service1.timeQuota++,user1.credit=0)})          line 14
  ).Z)                                                     line 15
or                                                         line 16
  (endaccess(user1,service1,createManagedJob).             line 17
  update(user1.reputation += 1))                           line 18
or                                                         line 19
  ([(user1.reputation ≤ 10) ∨                              line 20
  (service1.timeQuota < environment.currTime)].            line 20
  revokeaccess(user1,service1,createManagedJob)            line 21
  update(user1.reputation -= 1))                           line 22
)                                                          line 23
```

Table 3.2: Coarse-grained Security Policy 2

is 1 hour to reach the quota value, the ongoing obligation is triggered. This obligation informs the user, that allowed execution time is elapsing and the credit is required to proceed the execution. This rule refers to **onB0** model;

- If application's execution time exceeded the quota and no credit was submitted by the user, the system should terminate the execution of the application (on-authorization). Otherwise, if the credit is presented, the system triggers attribute updates (on-update). It doubles the execution quota time limit and sets the credit value to zero. The iterative prolonging of the usage quota can be performed unbounded number of times. This rule refers to **onA2** model;

3.4.3 Example 3. Coarse-grained Security Policy

Table 3.3 gives an example of services workflow policy (similar to one given in Figure 1.3). This policy is enforced on the coarse-grained level and control the sequence of services invocation allowed in the Grid. The security policy allows a Grid application

```
tryaccess(user1,service1,allocateResource).              line 1
permitaccess(user1,service1,allocateResource).           line 2
endaccess(user1,service1,allocateResource);              line 3
(                                                        line 4
  (tryaccess(user1,service2,createManagedJob).           line 5
  permitaccess(user1,service2,createManagedJob).         line 6
  endaccess(user1,service2,createManagedJob))            line 7
or                                                       line 8
  (tryaccess(user1,service3,createManagedJob).           line 9
  permitaccess(user1,service3,createManagedJob).         line 10
  endaccess(user1,service3,createManagedJob))            line 11
);                                                       line 12
tryaccess(user1,service4,storeResults).                  line 13
permitaccess(user1,service4,storeResults).               line 14
endaccess(user1,service4,storeResults)                   line 15
```

Table 3.3: Coarse-grained Security Policy 3

to call service `service1` (lines 1-3), than to invoke any of the computational services (i.e. either service `service2` or service `service3` but not both (lines 4-12)). If a sequence of services `service1` - `service2` or `service1` - `service3` is finished the Grid application is granted access to the service `service4` (lines 13-15).

3.4.4 Example 4. Fine-grained Security Policy

```
tryaccess(app_id,free_mathlib,open).                     line 1
[property(user,non_profit)].                             line 2
permitaccess(app_id,free_mathlib,open).                  line 3
endaccess(app_id,free_mathlib,open).                     line 4
  i(tryaccess(app_id,free_mathlib,read).                 line 5
  permitaccess(app_id,free_mathlib,read).                line 6
  endaccess(app_id,free_mathlib,read)).                  line 7
tryaccess(app_id,free_mathlib,close).                    line 8
permitaccess(app_id,free_mathlib,close).                 line 9
endaccess(app_id,free_mathlib,close)                     line 10
```

Table 3.4: Fine-grained Security Policy 4

Table 3.4 gives an example of the fine-grained security policy. It specifies the the job `app_id` running on the behalf of the Grid user is allowed to access the utility library `free_mathlib` if the user is qualified as `non_profit`. This attribute is mined from the set of attributes presented on the coarse-grained level of control. The property policy regulates mapping rules between coarse- and fine- grained attributes and example of such policy is given in Figure 4.6.

If the access is granted , the file should be opened first, read several times, and eventually closed.

Chapter 4

Enhancing Expressiveness of the Model

This Chapter covers the *policy level* and presents the enhancements of access and usage control model in Grid. These enhancements address the problems occurring during enforcement of usage security policies in distributed settings of Grid environment.

Section 4.1 introduces a formal model to describe and collect attributes and evaluate authorization predicates. Attributes are considered as a sensitive data, and Grid entities participating in access and usage control iteratively disclose their attributes to gain a certain level of trust to each other. This section describes *trust negotiation in Grid*, a policy-based technique that provides entities with the possibility to protect their own attributes and to negotiate with other entities access to those attributes. Attributes disclosed by a Grid user during trust negotiation are used to evaluate authorization predicates.

Section 4.2 examines uncertainties associated with mutable attributes used to produce an access decision. These uncertainties occur in distributed environment where notification mechanisms about new attribute values are not available. Thus, the reference monitor operates using possibly out-of-date attributes. *The risk-aware access and usage control model* introduces a probabilistic model to compute uncertainties and based on the computed value to make a credible access decision - to grant or deny the access. The foremost task of the risk-aware access and usage control is to approximate a continuity of policy enforcement. The system generates an efficient scheduler for querying fresh attribute values only when the further enforcement of the policy becomes too risky. The scheduler foresees when attributes probably should change and, moreover, this change is going to violate a policy.

4.1 Trust Negotiation in Grid

Security policies are based on attributes. Attributes are usually pushed with the access request or pulled by the reference monitor enforcing the security policy. In fact, attributes are sensitive data and Grid users want to disclose a minimum set of attributes which solve the access control problem. In opposite side, the reference monitor is interested in preserving privacy of security policies and is suspicious to disclose the set of required attributes.

Trust negotiations are suggested by our access and usage control model to establish trust between Grid users and resource providers without previous interactions. Trust negotiation is a policy-driven process allowing for automated on-the-fly disclosure and exchange of sensitive attributes and security policy statements. We integrate in Grid the trust negotiation model proposed by Koshutanski et al. [60, 59, 61]. This model was selected since it allows to express comprehensive negotiation scenarios and has security overhead of the same level regarding the existing analogues [71].

4.1.1 Attribute-based Access Control

Security policies considered here are based on *remote* and *immutable* attributes, protect the access to Grid service instances, and model **pre**-authorization UCON scenarios on the coarse-grained level of control. The security policy is formed mainly by authorization predicates. Table 4.1 gives a basic security policy example.

`tryaccess(s,o,r).`	line 1
`Pa.`	line 2
`permitaccess(s,o,r).`	line 3
`endaccess(s,o,r)`	line 4

Table 4.1: preA0 Security Policy

This security policy represents a usage scenario where an access decision is evaluated only once before the usage session starts and the access decision is done by checking *authorizations*. In the following, we refer to this security policy by considering only authorizations specified via $\mathcal{P}_{\mathcal{A}}$. The reference monitor derives the access decision from presented attributes and authorizations using inference rules of the formal model expressing these concepts [104].

In the following, we examine the syntax and semantics of authorizations. These $\mathcal{P}_{\mathcal{A}}$ policies are written as normal logic programs [14]. A logic program is a set of rules of the form:

$$A \leftarrow B_1, \ldots, B_n \tag{4.1}$$

A is called the head of the rule, each B_i is called a positive literal, whereas the conjunction of B_i is called the body of the rule. If the body is empty the rule is called a fact.

In the model we also have constraints that are rules with an empty head.

$$\leftarrow \quad B_1, \ldots, B_n \tag{4.2}$$

A constraint (4.2) is used to rule out from the set of acceptable models situations in which all B_i are true and all C_j are false.

Below we list the core predicates defined for the logical model representation at the coarse-grained level of control.

- $\texttt{grant}(User,\ \mathsf{Service}:s,\ \mathsf{Action}:p,\ \mathsf{Resource}:r)$ a predicate denoting that a $User$ is granted to invoke an action p on a Grid service s which exploits an underlying resource r.

- $\texttt{cred}(User,\ \mathsf{Attr}:a,\ \mathsf{Issuer}:i)$ a predicate denoting a credential token of a $User$ having attribute a issued by i.

Notice, that $User$ corresponds to a *subject*, $\mathsf{Service}:s$ and $\mathsf{Resource}:r$ - to an *object*, and $\mathsf{Action}:p$ - to *access rights*. Since Grid is an open system with a priori unknown clients we do not have a predefined set of client identities in the model. We denote that with a variable $User$ specifying untyped entity value in the respective fields of the predicates above.

The access is permitted if the logical implication of the $\texttt{grant}(User,\ \mathsf{Service}:s,\ \mathsf{Action}:p,\ \mathsf{Resource}:r)$ predicate can be drawn using presented attributes $\texttt{cred}(User,\ \mathsf{Attr}:a,\ \mathsf{Issuer}:i)$. Otherwise, the access is denied.

4.1.2 Feedback on Missing Attributes

The intuition behind the coarse-grained access control model is to provide Grid users with a feedback on missing credentials in cases of not enough access rights. Also the work by [54, 20] identifies the need of a feedback on access control requirements for open systems. The underlying access control model [62] is data-driven by two logic reasoning services: *deduction* and *abduction* [105]. We use deduction logic reasoning when taking decisions on whether a Grid user has enough access rights to get a Grid service, and abduction on logic programs as a core reasoning when computing a feedback on missing credentials. This subsection illustrates the essence of the interactive access control process as a core element for negotiation. We refer to [62] for details on the two reasoning services and their deployment in an interactive access control model.

Each Grid security domain has a security policy for access control $\mathcal{P}_\mathcal{A}$ and a *security policy for disclosure control* $\mathcal{P}_\mathcal{D}$. $\mathcal{P}_\mathcal{A}$ protects Grid's resources by stipulating what credentials a Grid user must satisfy to be authorized for a particular resource

while, in contrast, $\mathcal{P}_\mathcal{D}$ defines which credentials among those occurring in $\mathcal{P}_\mathcal{A}$ are disclosable so, if needed, can be demanded from a Grid user.

Each Grid user has a profile of active credentials $\mathcal{C}_\mathcal{A}$ available to a Grid security domain during a negotiation process. The reference monitor keeps user's set of active credentials for the duration of the user's request and its execution in Grid. The credential profile is also accessible by the fine-grained monitoring level.

The reference monitor also keeps a set of declined credentials $\mathcal{C}_\mathcal{N}$ that keeps track of what credentials a Grid user has declined to provide within a negotiation session. Declined credentials are internal to the negotiation model and are kept only for the duration of a current authorization process. They are not used at fine-grained level. The purpose of the declined credentials is to avoid loops in a negotiation process and to guarantee successful interactions in presence of alternative solutions.

`AccessDecisionWithFeedback`(*request*, $\mathcal{P}_\mathcal{A}$, $\mathcal{P}_\mathcal{D}$, $\mathcal{C}_\mathcal{A}$, $\mathcal{C}_\mathcal{N}$)

1: **if** *request* is a consequence of $\mathcal{P}_\mathcal{A}$ and $\mathcal{C}_\mathcal{A}$ **then** `grant`

2: **else**

3: **compute** a set of disclosable credentials $\mathcal{C}_\mathcal{D}$ entailed by $\mathcal{P}_\mathcal{D}$ and $\mathcal{C}_\mathcal{A}$. Remove from $\mathcal{C}_\mathcal{D}$ all presented and declined credentials, i.e. $\mathcal{C}_\mathcal{D} = \mathcal{C}_\mathcal{D} \setminus (\mathcal{C}_\mathcal{N} \cup \mathcal{C}_\mathcal{A})$,

4: **compute** a set of missing credentials $\mathcal{C}_\mathcal{M}$ such that
(i) $\mathcal{C}_\mathcal{M} \subseteq \mathcal{C}_\mathcal{D}$,
(ii) $\mathcal{P}_\mathcal{A}$ together with $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ grant *request*,
(iii) $\mathcal{C}_\mathcal{A}$ and $\mathcal{C}_\mathcal{M}$ preserve $\mathcal{P}_\mathcal{A}$ consistent.

5: **if** no set found **then** `deny` **else** ***ask***$(\mathcal{C}_\mathcal{M})$.

Figure 4.1: Coarse-grained Access Decision with Feedback on Missing Credentials

Figure 4.1 shows the core access decision algorithm with feedback on missing credentials. Input to the decision process is the service request, access policy, disclosure policy, user's set of active and declined credentials. First step checks if the user has enough access rights to access the resource according to its active credentials and Grid's access policy. If the check succeeds the function returns grant.

In case of not enough access rights, the algorithm performs two steps to compute a feedback on missing credentials. It first computes a set of disclosable credentials inferred form user's active credentials and the Grid's disclosure policy, and from the resulting set, it removes the already presented and declined credentials. Second, the algorithm uses an abduction reasoning to compute a set of missing credentials, out of the disclosable ones, that is sufficient to unlock the requested resource. The abduction reasoning guarantees that if a solution exists then it is consistent with the access policy and user's active credentials. If a solution set is found it is returned back, else a denial message is returned instead.

The coarse-grained access control process is implemented by using the DLV system [72] as a back-end engine for the deductive and abductive computations.

4.1.3 Negotiation-based Decision Making

We first define the three policies that Grid users (specified as *clients*) and Grid resource providers (specified as *servers*) have:

- $\mathcal{P}_{\mathcal{A}}$ a policy for protecting opponent's *own* resources based on *foreign* credentials
- $\mathcal{P}_{\mathcal{AC}}$ a policy for protecting opponent's *own* credentials based on *foreign* credentials
- $\mathcal{P}_{\mathcal{D}}$ a policy for disclosure the need of (missing) *foreign* credentials

Figure 4.2 shows the negotiation protocol. The protocol runs on both client and server side. The meaning of $\mathcal{C}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{N}}$and $\mathcal{C}_{\mathcal{M}}$ is read as the set of presented foreign credentials, the set of declined foreign credentials and the set of missing foreign credentials, respectively. We also denote with $\mathcal{O}$ a set of own credentials with respect to a negotiation opponent. We also defined the notion of suspended credential requests to handle the fact that during a negotiation process entities may start to request each other credentials that are already in a negotiation. The set $\mathcal{O}_{neg}$ keeps track of the opponent's own credentials that have been requested and which are still in negotiations. Hence, if a request for a credential already in a negotiation the protocol suspends the request until the respective negotiation thread is finished. When the original thread returns an access decision the protocol resumes all threads awaiting on the requested credential with the decision.

A negotiation process has the following main steps:

1. A client, Alice, sends a service request *request* and (optionally) a set of credentials $\mathcal{C}_p$ to a server, Bob.
2. Bob's negotiation dispatcher receives the requests, checks if it is a service request and runs the negotiation protocol in a new thread with new negotiation session.
3. When the protocol is run, it updates opponent's set of active credentials with the newly presented ones and checks if the request is already being in a negotiation (steps 1 and 2).
4. If Alice's request is not to be suspended then Bob looks at *request* and if it is a request for a service he calls for an access decision with his *policy for access to resources* $\mathcal{P}_{\mathcal{A}}$, his policy for disclosure of foreign credentials $\mathcal{P}_{\mathcal{D}}$, the set of Alice's active $\mathcal{C}_{\mathcal{A}}$ and declined $\mathcal{C}_{\mathcal{N}}$ credentials (step 9).

Negotiation session: $\mathcal{C}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{N}}$ and $\mathcal{O}_{neg}$. Initialization: $\mathcal{C}_{\mathcal{A}}$= $\mathcal{C}_{\mathcal{N}}$= $\mathcal{O}_{neg}$= $\emptyset$.
`NegotiationDispatcher`{
`OnReceiveRequest` $\langle request, \mathcal{C}_p \rangle$ do

1: **if** `isService`($request$) **then**
2: `reply` $resp$ = `NegotiationProtocol`($request$, $\mathcal{C}_p$); // in a new session thread.
3: **else**
4: `reply` $resp$ = `NegotiationProtocol`($request$, $\mathcal{C}_p$); // in a new thread under the original session.

`OnSendRequest` $\langle request, \mathcal{O}_p \rangle$ do

1: $result$ = `invoke NegotiationProtocol`($request$, $\mathcal{O}_p$)@`Opponent`; // in a new session thread.

}
`NegotiationProtocol`($request$, $\mathcal{C}_p$){

1: $\mathcal{C}_{\mathcal{A}} = \mathcal{C}_{\mathcal{A}} \cup \mathcal{C}_p$;
2: **if** $request in \mathcal{O}_{neg}$ **then**
3: suspend and await for the $result$ on $request$'s negotiation;
4: return $result$ when resumed;
5: **else**
6: $\mathcal{O}_{neg} = \mathcal{O}_{neg} \cup \{request\}$;
7: **repeat**
8: **if** `isService`($request$) **then**
9: $result$ = `AccessDecisionWithFeedback`($request$, $\mathcal{P}_{\mathcal{A}}$, $\mathcal{P}_{\mathcal{D}}$, $\mathcal{C}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{N}}$);
10: **else**
11: $result$ = `AccessDecisionWithFeedback`($request$, $\mathcal{P}_{\mathcal{AC}}$, $\mathcal{P}_{\mathcal{D}}$, $\mathcal{C}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{N}}$);
12: **if** $result$ == `ask`($\mathcal{C}_{\mathcal{M}}$) **then**
13: `AskCredentials`($\mathcal{C}_{\mathcal{M}}$);
14: **until** $result$ == `grant` **or** $result$ == `deny`;
15: $\mathcal{O}_{neg} = \mathcal{O}_{neg} \setminus \{request\}$;
16: resume all processes awaiting on $request$ with the $result$ of the negotiation;
17: return $result$;
18: **end if**

}
`AskCredentials`($\mathcal{C}_{\mathcal{M}}$){

1: **parfor** each $c \in \mathcal{C}_{\mathcal{M}}$ **do**
2: $response$ = `invoke` i`AccessNegotiation`(c, $\emptyset$)@`Opponent`;
3: **if** $response$ == `grant` **then** $\mathcal{C}_{\mathcal{A}} = \mathcal{C}_{\mathcal{A}} \cup \{c\}$ **else** $\mathcal{C}_{\mathcal{N}} = \mathcal{C}_{\mathcal{N}} \cup \{c\}$;
4: **end parfor**
5: **while** $\mathcal{C}_{\mathcal{M}} \not\subseteq (\mathcal{C}_{\mathcal{A}} \cup \mathcal{C}_{\mathcal{N}})$ **do** `wait`();

}

Figure 4.2: Negotiation Protocol

5. If *request* is a request for a credential then Bob calls for an access decision with his *policy for access to own credentials* $\mathcal{P}_{\mathcal{AC}}$, his policy for disclosure of foreign credentials $\mathcal{P}_{\mathcal{D}}$ and Alice's active $\mathcal{C}_{\mathcal{A}}$ and declined $\mathcal{C}_{\mathcal{N}}$ credentials (step 11).

6. In the case of computed missing credentials $\mathcal{C}_{\mathcal{M}}$ (steps 12 and 13) Bob transforms $\mathcal{C}_{\mathcal{M}}$ into single requests for credentials and awaits until receives all responses (steps 1–5 of `AskCredentials` function). At this point Bob acts as a client, requesting Alice the set of missing credentials. Alice runs the same protocol with swapped roles.

7. When Bob receives all responses, he restarts the loop and consults for a new access decision.

8. When a final decision of grant or deny is taken, the respective response is returned back to Alice.

4.1.4 Negotiation Schema Implementation

We adopted a thread-based negotiation of missing credentials by transforming the need of missing credentials into a sequence of single requests each asking for a foreign credential from the missing set. Each request for a credential spurs a new negotiation thread that negotiates access to this credential.

One of the technical issues in the protocol is in the way the server requests missing credentials back to the client. We use the keyword `parfor` for representing that the body of the loop is run each time in a parallel thread. Thus, each missing credential is requested independently from the requests of the others. At that point of the protocol, it is important that each of the finished threads updates presented and declined sets of credentials properly without interfering with other threads. We note that each credential request marks (updates) the requested foreign credential as declined after a session time expires.

The thread based implementation with shared $\mathcal{C}_{\mathcal{A}}$ and $\mathcal{C}_{\mathcal{N}}$ is necessary to allow for a polynomial execution time of the trust negotiation protocol with respect to the number of queries to the abduction algorithm. Indeed, without a shared memory of received credentials it is possible to structure policies in a way that a credential would be asked many times. In this way, the protocol queries for credentials are bounded by the number of credentials occurring in the policy $\mathcal{P}_{\mathcal{AC}}$.

Declining a credential in a negotiation process is when an entity is asked for it and the same entity replies to the request with answer deny. When an entity is asked for a credential and there is a counter request for additional credentials then the thread started the original request awaits for the reply and treats the requested credential as not yet released.

When a trust negotiation module is initially loaded it internally loads an application server and sets the dispatcher module resident in the memory awaiting

on requests. When the server receives a request it automatically redirects the request to the dispatcher which in turn transforms it from raw data to a high-level representation.

The negotiation dispatcher is an essential component to the negotiation protocol. The dispatcher has a role of a *negotiation server* that manages entities requests and their negotiation sessions. Whenever a request arrives the dispatcher runs the negotiation for that request in a new thread that shares same session variables $\mathcal{C}_{\mathcal{A}}$, $\mathcal{C}_{\mathcal{N}}$ and $\mathcal{O}_{neg}$ with other threads running under the same negotiation session.

On each received request the dispatcher analyzes the session data from the request and its local database and acts as following. If no session data is specified in the request (and request for a service) then the dispatcher generates new session information ($\mathcal{C}_{\mathcal{A}}$=$\mathcal{C}_{\mathcal{N}}$= $\mathcal{O}_{neg}$=$\emptyset$) and runs the negotiation protocol with the new session information. If a session exists and the session data correctly maps to the corresponding one in dispatcher's local database then the dispatcher runs the negotiation protocol in a thread under the existing session. If the specified session does not match to any internal session then deny message is returned back.

Figure 4.3 shows the architecture and communications of the negotiation module. The architecture logically splits a negotiation process in two levels: negotiation

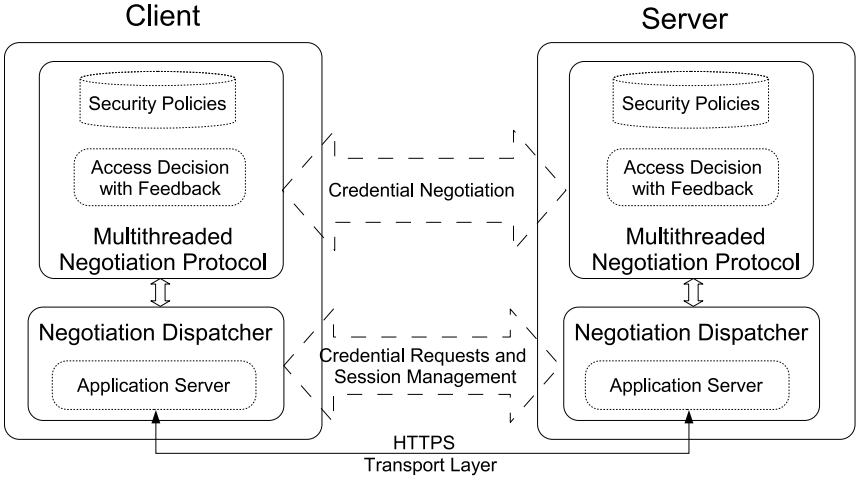

Figure 4.3: Negotiation Framework Message Interoperation

requests and session management; and pure credential negotiation. The division is driven by the goal of making efficient and scalable negotiations for a multi-user environment such as Grid.

4.1.5 Security Policies for Trust Negotiation

Assume, a computing center decides to share part of its computational GRAM services on a Grid platform to acquire new users. The center offers a set of free java libraries that can be invoked by submitted user's jobs for efficient computing

of mathematical functions, (e.g. the Fast Fourier Transform, Matrix inversion, file format conversion, etc). According to an internal policy of the computing center, these libraries can be used by researchers or students of universities and by members of non profit organizations. At the same time, the computing center also offers a commercial version of the same libraries but used only by users that have paid a given fee, or that belong to a set of associations (e.g., IEEE members).

Moreover, the development staff of the computing center continuously work to improve the performances of the free libraries and the beta versions of the libraries are tested directly by the computing center's users. Since the beta version of the libraries could include some errors that could, in principle, allow unfamiliar users intentionally or unintentionally to breach a system functionality, the computing center decides to allow only well-behaved users to access beta libraries.

COMPUTING CENTER SECURITY POLICIES:
Access Policy $\mathcal{P}_{\mathcal{A}}$
`grant`(User, gramService, create, free_mathlib) ← `cred`(User, studentPhD, universityMalaga).
`grant`(User, gramService, create, free_mathlib) ← `cred`(User, researchSenior, universityMalaga).
`grant`(User, gramService, create, devel_mathlib) ← `grant`(User, gramService, create, free_mathlib).
`grant`(User, gramService, create, comm_mathlib) ← `grant`(User, gramService, create, free_mathlib), `cred`(User, ieeeEnrollment, ieeeInc).
`grant`(User, gramService, create, comm_mathlib) ← `cred`(User, visaCard, bankRoma), `cred`(User, ssn, governmentAuth).
Credential Policy $\mathcal{P}_{\mathcal{AC}}$
`cred`(computerCenter, affiliation, governmentAuth) ← `cred`(User, employee, anEmployer).
`cred`(computerCenter, visaConfirmed, visaEurope) ← .
Disclosure Policy $\mathcal{P}_{\mathcal{D}}$
`cred`(User, studentPhD, universityMalaga) ← .
`cred`(User, researchSenior, universityMalaga) ← .
`cred`(User, ieeeEnrollment, ieeeInc) ← .
`cred`(User, visaCard, bankRoma) ← .
`cred`(User, ssn, governmentAuth) ← .
`cred`(User, employee, anEmployer) ← .

USER SECURITY POLICIES:
Credential Policy $\mathcal{P}_{\mathcal{AC}}$
`cred`(marioRossi, visaCard, bankRoma) ← `cred`(Rprovider, visaConfirmed, visaEurope).
`cred`(marioRossi, ssn, governmentAuth) ← `cred`(Rprovider, affiliation, governmentAuth).
`cred`(marioRossi, employee, anEmployer) ← .
Disclosure Policy $\mathcal{P}_{\mathcal{D}}$
`cred`(Rprovider, visaConfirmed, visaEurope) ← .
`cred`(Rprovider, affiliation, governmentAuth) ← .

Figure 4.4: Example of Trust Negotiations Policies

Figure 4.4 shows the coarse-grained security policies of the computing center and a user underpinning our negotiation scenario. Figure presents access, credential, and disclosure policies of the computing center, as well as, credential and disclosure policies of the user.

We use a term starting with a capital letter (e.g., User, Rprovider, etc) to refer to a variable that represents any value in its field. The variable is referable from other predicates within a same rule. Terms starting with lower case letters represent constants in a policy (e.g., marioRossi, visaConfirmed, etc).

The coarse-grained access policy $\mathcal{P}_{\mathcal{A}}$of the computing center governs access to a

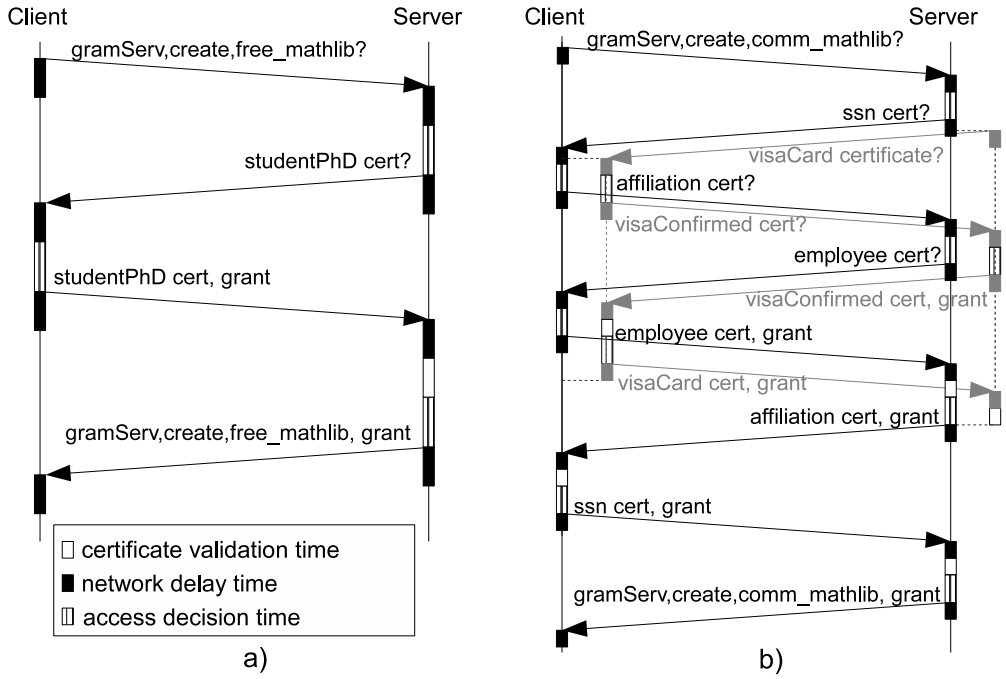

Figure 4.5: Grid Trust Negotiation Scenarios

computational service and its underlying resources. Access permissions vary based on what underlying resources a user claims to utilize during execution of his submitted job. For example, the invocation of the free version of the java library is granted to senior researchers and PhD students at the University of Malaga, given that they submitted a credential attesting their positions. It also states that access to the beta version of the library is given to those who have access to the free version.

Students and researches at the University of Malaga officially enrolled to IEEE community may utilize facilities of the commercial version of the computational library without fees. Otherwise, to access the commercial version of the library a payment transaction is required. How the payment transaction goes in Grid is out of the scope of our model. We present an access policy governing a disclosure of information needed by the computing center to complete a payment transaction. For instance, a visa card credential could contain information embossed on the user's physical payment card. A ssn (social security number) credential could be asked additionally by the computing center to validate the user's account. In user's turn, the user could request a visa confirmed credential from the computing center to escape frauds and prove that the computing center is eligible to complete payment transactions.

The credential policy of the computing center allows access to computing center affiliation credential to users that have presented a credential attesting their employee status. Access to the visa confirmed credential of the computing center is

Property Policy:
property(User, non_profit) ← cred(User, studentPhD, universityMalaga).
property(User, non_profit) ← cred(User, researchSenior, universityMalaga).
property(User, commercial) ← property(User, non_profit), cred(User, ieeeEnrollment, ieeeInc).
property(User,commercial) ← cred(User, visaCard, bankRoma).

Figure 4.6: Example of Property Policy

not protected and given on request. Although this credential could be additionally protected we omit it simplifying the negotiation scenario.

The disclosure policy discloses the need of all credentials involved in the access policy. We do not focus on negotiation strategies and how to disclose sensitive credentials, we structure the policy as all credentials are disclosable at any request for negotiation.

Looking at the user side, the credential policy controls an access to the social security number and visa card credentials of the user. The policy states that the user allows access to his visa card credential only if a computing center proves his official agreement with Visa Inc. Similarly, the social security number credential is sent during negotiations if the computing center is allowed to collect such information by an appropriate government affiliation. In contrary, the credential stating user's employment record is disclosed freely on demand.

Figure 4.5 draws trust negotiation scenarios in time settings. In the first scenario (a), the user requests the access to the GRAM service instance and $free\_mathlib$, in the second scenario (b) - to the GRAM service instance and *comm_mathlib*. In both scenarios, no attributes are pushed with the initial request.

Attributes acquired during trust negotiations might be required for the fine-grained monitoring, e.g. like in the fine-grained policy example given in subsection 3.4.4. The *property policy* regulates mapping rules between coarse- and fine- grained attributes and example of such policy is given in **??**. Notice, that this policy is expressed using the same formalism as for the disclosure and credential policies.

4.2 Risk-aware Access and Usage Control

This section addresses the enforcement of security policies based on *remote* attributes with *observable mutability* [36, 64, 63, 65].

Since attributes may change in time, fresh values of attributes are essential for a correct decision making and policy enforcement [95, 87]. Continuity of control requires permanent awareness about the current attribute values. Decision making mechanisms should be supplemented with security mechanisms responsible for timely acquisition and delivery of attributes.

Distributed settings of Grid environment hardens the design of such security mechanisms. In fact, *remote* attributes with *observable mutability* are always associated with some uncertainties. Attributes can be uncertain because of two types

of causes: *unintentional* and *intentional*. Unintentional causes appear because of distributed settings of Grid and are always present (e.g. delays in delivery, noise, loss of connection, etc.). Intentional causes are connected with deliberate alteration of attributes. For instance, attributes are assertions claimed by third parties (i.e. by local certification authorities of Grid entities) which are possibly partially trusted.

Novel security mechanisms are required to count and tolerate uncertainties associated with attributes. This section proposes to enhance the *policy level* by specifying in the security policy the allowed uncertainty level. Uncertainties associated with attributes are paired with a *risk* and its semantics denotes a probability of a security policy failure multiplied by an impact of this failure. Analyzing the current risk value, the system weights pros and cons of granting or revoking access using a *cost matrix* and makes the most rational access decision. The formal (mathematical) model of risk-aware access and usage control is based on the cost-utility theory, risk management, Markov chains, and semi-ring computational models.

The section is structured as follows. Subsection 4.2.1 points to access and usage control scenarios covered in this section. Subsection 4.2.2 enlists all types of uncertainties associated with attributes. Subsection 4.2.3 describes decision making in risk-aware access and usage control. Subsection 4.2.4 calculates probabilities used to make a decision under risk. Subsection 4.2.5 outlines a risk-aware policy enforcement. Subsection 4.2.6 summarizes this section with discussions of the proposed approach and related works.

4.2.1 Access and Usage Control Scenario

Attribute Acquisition Models

Security policies considered here are based on *remote* attributes with *observable mutability*. An attribute is modeled as variable a which might take a value from (in)finite domain of attribute values $ATTR$. *Remote* means that the attribute is managed by a remote Attribute Provider which is not under control of Reference Monitor enforcing the policy (see Figure 4.7). *Observable mutability* means that Reference Monitor observes only partially how the attribute changes in time, and moreover it cannot influence (or even block) the attribute modifications.

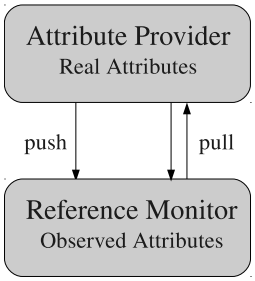

Figure 4.7: Attribute Acquisition Model

An attribute might change in discrete points of time and this process is modeled via a finite sequence

$$RealAttr = \{(a_i^{real}, t_i) | a_i^{real} \in ATTR, t_i \in T, \forall i : t_i < t_{i+1}\}$$

where each element (a_i^{real}, t_i) specifies the attribute value a_i changed at time t_i. T is a countable infinite set of natural numbers which models time ticks. During time interval $|t_i - t_{i+1}|$ the attribute does not change, and the attribute value at t_{curr} equals a_i^{real}, where $t_i < t_{curr} < t_{i+1}$.

Only Attribute Provider can see *RealAttr*, while Reference Monitor operates with a finite sequence of observed attributes specified via

$$ObservedAttr = \{(\langle a_j^{obs}, t_j \rangle, \tilde{t}_j) | a_j^{obs} \in ATTR, t_j \in T, \forall j : t_j \leq \tilde{t}_j\}$$

where $\langle a_j^{obs}, t_j \rangle$ corresponds to elements (a_i^{real}, t_i) of *RealAttr*, i.e. $\forall j = i, \langle a_j^{obs}, t_j \rangle = (a_i^{real}, t_i)$. $\tilde{t}_j$ specifies the time point when the attribute was delivered to Reference Monitor and used to evaluate policy predicates. Attribute delivery and access decision making are time-consuming operations in Grid, thus $\tilde{t}_j$ is usually bigger than t_j (time when attribute was issued). Notice, that Attribute Provider and Reference Monitor share the same trusted clocks which start to work at $t_{try} = 0$.

In access and usage control two attribute acquisition models exist: *push* and *pull* (see Figure 4.7).

Definition 4. (**Attribute Acquisition Push Model**) *Attribute Acquisition Push Model defines a scenario when each new attribute value is pushed from Attribute Provider to Reference Monitor. Formally, this means that ObservedAttr and RealAttr have the same number of elements. i.e.*

$$|ObservedAttr| = |RealAttr|$$
$$\forall i,\ (a_i^{real}, t_i) \in RealAttr\ \exists j = i,\ (\langle a_j^{obs}, t_j \rangle, \tilde{t}_j) \in ObservedAttr$$

where $|S|$ specifies a number of elements in a sequence S

Definition 5. (**Attribute Acquisition Pull Model**) *Attribute Acquisition Pull Model defines a scenario when Reference Monitor queries Attribute Provider to give the current attribute value.*

There could be the case when Reference Monitor queries too often (more frequently than the attribute changes), and as a result ObservedAttr contains ***redundant*** *elements, i.e.*

$$|ObservedAttr| \geq |RealAttr|$$
$$RealAttr = \{ ...,\ (a_i^{real}, t_i),\ (a_{i+1}^{real}, t_{i+1}),\ ... \},$$
$$ObservedAttr = \{ ...,\ (\langle a_j^{obs}, t_j \rangle, \tilde{t}_j),\ ...,\ (\langle a_{j+k}^{obs}, t_{j+k} \rangle, \tilde{t}_{j+k}),\ ... \},$$
$$\forall i, j\ \exists k \geq 0\ :\ t_i = t_j = ... = t_{j+k} < t_{i+1}$$

In opposite, Reference Monitor might query too rare (less frequently than the attribute changes), and as a result ObservedAttr contains ***insufficient*** *elements, i.e.*

$$|ObservedAttr| \leq |RealAttr|$$
$$RealAttr = \{ ..., (a_i^{real}, t_i), (a_{i+1}^{real}, t_{i+1}), ..., (a_{i+k}^{real}, t_{i+k}), ... \},$$
$$ObservedAttr = \{ ..., (\langle a_j^{obs}, t_j \rangle, \tilde{t}_j), (\langle a_{j+1}^{obs}, t_{j+1} \rangle, \tilde{t}_{j+1}), ... \},$$
$$\forall i, j \; \exists k \geq 1 \; : \; t_i = t_j, \; t_{i+k} = t_{j+1}$$

Obviously, the attributes acquisition pull model with redundant elements is as expressive as the push model, i.e. each change of the attribute will be eventually captured by Reference Monitor. On another side, the pull model with redundant attributes imposes implementation difficulties since often attribute queries increase security overhead and might slowdown the overall system performance.

As example, assume attribute a specifying location of a Grid User. Domain of a is $ATTR = \{locA, locB, locC\}$. The attributes observed by Attribute Provider and Reference Monitor in presence of various attributes acquisition models (push and pull insufficient) are given in Figure 4.8.

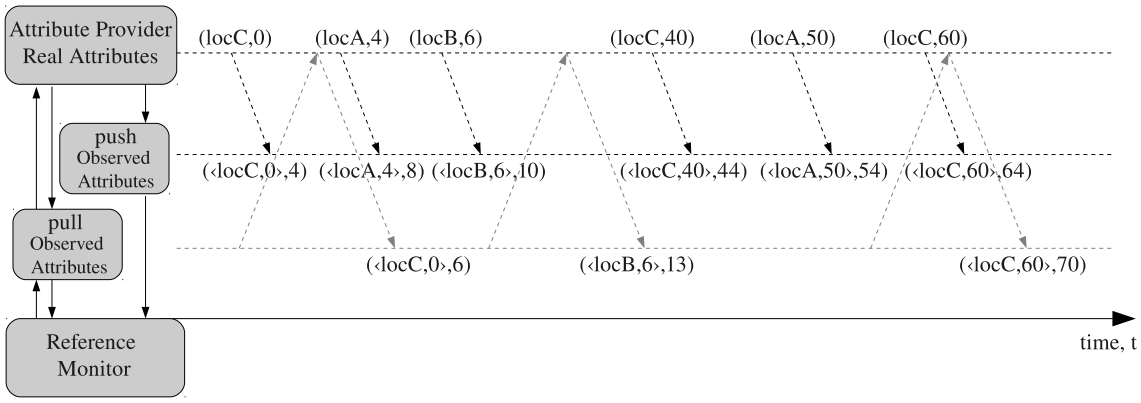

Figure 4.8: Attribute Acquisition Pull and Push Models

Notice, that the average time spent for the attribute delivery and decision making, i.e. $\triangle\, t_{processing} = (\tilde{t}_j - t_j)$ for each element in the observed attributes sequence, is always bigger for the pull model than for the push model. This is because the pull model needs two round of network traverse - from Reference Monitor to Attribute Provider and back.

Correct Policy Enforcement

Risk-aware access and usage control concerns **pre** and **on** UCON usage scenarios and is realized on the coarse-grained level of control. A security policy is formed

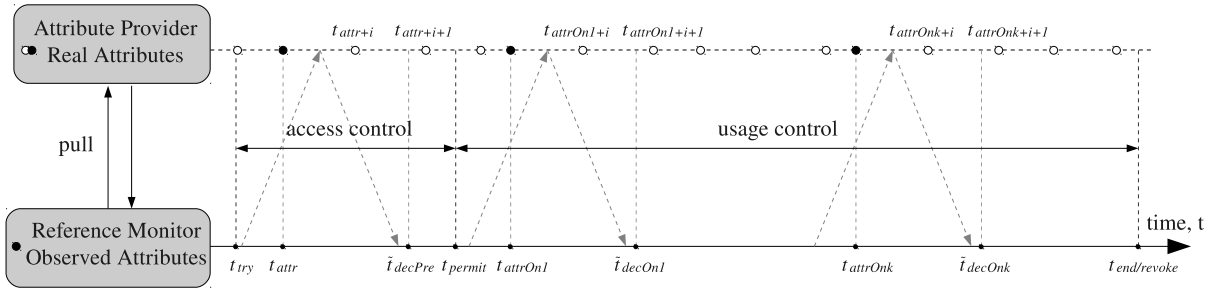

Figure 4.9: Security Policy Enforcement with Pull Acquisition Model

mainly by authorization and condition predicates. For simplicity, each predicate is assumed to constrain only single attribute, and the security policy consists of at most two predicates: first specifies access control (pre-authorization), and second - usage control (ongoing authorization). Table 4.2 gives a typical security policy example. The security policy requires that the Grid User accessing and using some Grid resources should remain in *locB*, or *locC*.

`tryaccess(s,o,r).`	line 1
`Ppre(a = locB or a = locC).`	line 2
`permitaccess(s,o,r).`	line 3
`( endaccess(s,o,r)`	line 4
`or`	line 5
($\overline{Pon}$`(a = locB or a = locC).`	line 6
`revokeaccess(s,o,r) ) )`	line 7

Table 4.2: preOnA0 Policy

Figure 4.9 shows how the policy enforcement evolves in time. It starts at t_{try}, and proceeds with querying the current attribute value. Either push or pull model is used for acquisition of attribute, Reference Monitor processes only the first received value and evaluates `Ppre` only once.

Definition 6. (**Correct Enforcement of Access Control**) *Reference Monitor correctly enforces the security policy during pre-authorization and it grants access at* $\tilde{t}_{decPre}$ *only if the following holds (see Figure 4.9)*

$$\texttt{Ppre}(a^{real}_{attr+i}) = true \wedge \texttt{Ppre}(a^{real}_{attr}) = true$$

where real and observed attribute sequences are

$$RealAttr = \{\ (a^{real}_{try}, t_{try}),\ ...,\ (a^{real}_{attr}, t_{attr}),\ ..., (a^{real}_{attr+i}, t_{attr+i}),\ (a^{real}_{attr+i+1}, t_{attr+i+1})\ ...\ \}$$

$$ObservedAttr = \{\ (\langle a^{obs}_{attr}, t_{attr}\rangle, \tilde{t}_{decPre})\ \}$$

and

$$t_{attr+i} \leq \tilde{t}_{decPre} \leq t_{attr+i+1},\ i \geq 0$$

For convenience, the notation $G_{t_{attr}}$ specifies that the access decision should be granted at t_{attr}, i.e. $\texttt{Ppre}(a^{real}_{attr}) = true$, while $\overline{G}_{t_{attr}}$ specifies the opposite. $G_{\tilde{t}_{decPre}}$ specifies that $\texttt{Ppre}(a^{real}_{attr+i}) = true$ holds. In case of the correct enforcement of access control, both events $G_{t_{attr}}$ and $G_{\tilde{t}_{decPre}}$ should be presented.

The correct policy enforcement implies that having the observed attribute sequence, the policy is enforced exactly in the same fashion as it would be enforced with the real attribute sequence. In fact, these sequences are not usually equal. Reference Monitor operates only with the observed attributes and knows only $G_{t_{attr}}$. Reference monitor needs additional knowledge about how the attribute changes in time to deduce $G_{\tilde{t}_{decPre}}$. For instance, if Reference Monitor knows that the attribute changes as some deterministic function of time, it could compute the real value of the attribute at any time and show definitely whether $G_{\tilde{t}_{decPre}}$ holds or not.

The initial assumption that the attribute is *remote* and has *observable mutability* imposes that Reference Monitor possesses incomplete or *uncertain* knowledge about real attribute vales. Assume, that this knowledge is probabilistic and

$$\mathbf{Pr}[G_{\tilde{t}_{decPre}} \mid G_{t_{attr}}]$$

specifies the probability computed by Reference Monitor that the access should be granted at time $\tilde{t}_{decPre}$ knowing that the security policy was satisfied at t_{attr}. Notice, the correct enforcement of access control requires that the access is granted if the predicates are satisfied exactly at t_{attr} and $\tilde{t}_{decPre}$ and the policy might be violated between t_{attr} and $\tilde{t}_{decPre}$.

As example, let's consider the enforcement of access control for real and observed attribute sequences given in Figure 4.8. Assume, that $t_{try} = 0$ and $t_{permit} = 7$. In this case real and observed attributes are following:

$$RealAttr : \{(locC, 0), (locA, 4), (locB, 6)\}$$
$$ObservedAttr(push) : \{(\langle locC, 0\rangle, 4)\}$$
$$ObservedAttr(pull) : \{(\langle locC, 0\rangle, 6)\}$$

Interestingly, either push or pull model is presented the same attribute value $(locC, 0)$ is used by Reference Monitor for the evaluation. In both cases, the predicate $\texttt{Ppre}(locC) = true$ and Reference Monitor might grant the access. For the pull model, the grant access decision is correct, because at the time of the decision enforcement, i.e. $\tilde{t}_{decPre} = 6$, the real attribute value is $a = locB$, and this value also satisfies `Ppre`. For the push model, the grant access decision is erroneous at the time of the decision enforcement, i.e. $\tilde{t}_{decPre} = 4$, the real attribute value is $a = locA$, and this value violates `Ppre`. Thus, Reference Monitor enforcing the security policy using incomplete (uncertain) knowledge might produce wrong access decisions.

Pre-authorization ends at t_{permit}, and usage control phase begins and Reference Monitor continuously evaluates the predicate `Pon`. Since the attribute changes in

discrete moments of time, the predicate should be evaluated every time the attribute changes. If the push model is not presented in the system for the attribute acquisition, Reference Monitor should pull the attribute several times during the usage.

Definition 7. (**Correct Enforcement of Usage Control**) *Reference Monitor correctly enforces the security policy and does not revoke the access after evaluating the policy k-th time, if the following holds (see Figure 4.9)*

$$\texttt{Pon}(a_m^{real}) = true,\ \forall m \in \{1,\ ...,\ attrOnk + i\}$$

where real and observed attribute sequences are

$$RealAttr = \{(a_1^{real}, t_1),\ ...,\ (a_{attrOnk}^{real}, t_{attrOnk}),\ ...,$$
$$(a_{attrOnk+i}^{real}, t_{attrOnk+i}),\ (a_{attrOnk+i+1}^{real}, t_{attrOnk+i+1}),\ ...\ \}$$
$$ObservedAttr = \{\ ...,\ (\langle a_{attrOnk}^{obs}, t_{attrOnk}\rangle, \tilde{t}_{decOnk}),\ ...\ \}$$

and

$$t_{permit} = t_1,\ t_{attrOnk+i} \leq \tilde{t}_{decOnk} \leq t_{attrOnk+i+1},\ i \geq 0$$

or revoke the access if the following holds

$$\forall attrOnk,\ s.t.\ (a_{attrOnk}^{real}, t_{a_{attrOnk}^{real}}) \in RealAttr,\ \mathbf{Pon}(a_{attrOnk}^{real}) = false,$$
$$\exists(\langle a_{attrOnk}^{obs}, t_{attrOnk}\rangle, \tilde{t}_{decOnk}) \in ObservedAttr,\ \tilde{t}_{decOnk} = t_{attrOnk}$$

For convenience, the notation $G_{|t_{permit} - \tilde{t}_{decOnk}|}$ specifies that the access decision should be granted and the usage session should be continued at t_{decOnk}. This means that the policy has never been violated since t_{permit}, i.e. each attribute change was checked and the predicate `Pon` was always satisfied.

In usage control the policy is evaluated every time when the attribute changes. In case of the pull attribute acquisition model some attribute changes might be unnoticed. Thus, Reference Monitor has less information to prove that $G_{|t_{permit} - \tilde{t}_{decOnk}|}$ holds and to enforce the policy correctly at $\tilde{t}_{decOnk}$. Assume, that knowledge of Reference Monitor about the predicate satisfaction in this interval is probabilistic and

$$\mathbf{Pr}[G_{(t_{permit}:\tilde{t}_{decOnk})} \mid G_{t_{attrOn1}},\ G_{t_{attrOn2}},\ ...,\ G_{t_{attrOnk}}]$$

specifies the probability computed by Reference Monitor that the usage session should be continue at time $\tilde{t}_{decOnk}$ knowing that the predicate was satisfied at time points when observed attributes were issued. Notice, that the interval (t_{permit} : $\tilde{t}_{decOnk}$) concerns all real attributes changes between t_{permit} and $\tilde{t}_{decOnk}$ but excluding observed attribute changes.

As example, let's consider the enforcement of usage control for real and observed attribute sequences given in Figure 4.8. Assume, that $t_{permit} = 6$ and real and observed attributes are following:

$$RealAttr : \{(locB, 6), (locC, 40), (locA, 50), (locC, 60)\}$$
$$ObservedAttr(push) : \{(\langle locB, 6\rangle, 10), (\langle locC, 40\rangle, 44), (\langle locA, 50\rangle, 54), (\langle locC, 60\rangle, 64)\}$$
$$ObservedAttr(pull) : \{(\langle locB, 6\rangle, 13), (\langle locC, 60\rangle, 70)\}$$

For the push model, Reference Monitor terminates the usage session when the attribute $(\langle locA, 50\rangle, 54)$ is received and processed at $\tilde{t}_{decOnk} = t_{revoke} = 54$. Although, the access is revoked as soon as possible there were still 4 time ticks when the policy was violated. This is the inevitable *minimal policy violation time* used for the attribute delivery and access evaluation.

For the pull model, Reference Monitor observes only attributes which satisfy the predicate `Pon` and never revokes access until the requester ends the usage session normally at $t_{end} = 80$. This is erroneous enforcement because at $t = 50$ the requestor was in $locA$ and violated the security policy. Thus, Reference Monitor enforces the security policy using incomplete (uncertain) knowledge and might produce wrong decisions.

Proposition 1. *The correct enforcement of access and usage control is possible if the attribute acquisition push model is presented in the system and*

$$\forall j(\langle a_j^{obs}, t_j\rangle, \tilde{t}_j) \in ObserverdAttributes,\ \tilde{t}_j = t_j,\ i.e.\ \triangle\, t_{processing} = 0$$

Proof This comes from the definition.

4.2.2 Intentional and Unintentional Uncertainties

Neither push no pull model in Grid can guarantee the correct enforcement of access and usage control. This subsection summarizes uncertainties associated with the attribute acquisition and discusses two types of uncertainties: *unintentional* which corresponds to a *freshness and correctness of attributes*, and *intentional* which corresponds to a *trustworthiness of attributes*.

Freshness and Correctness of Attributes

The following types of unintentional uncertainty are identified: *freshness I*, *II*, *III*, and *correctness*.

Freshness I corresponds to usage control scenarios where exists only the *attribute acquisition pull model with insufficient elements*:

$$|ObservedAttr| < |RealAttr|$$
$$\forall(\langle a_j^{obs}, t_j\rangle, \tilde{t}_j) \in ObservedAttr,\ \exists(a_i^{real}, t_i) \in RealAttr$$
$$s.t.\ \langle a_j^{obs}, t_j\rangle = (a_i^{real}, t_i),\ and\ t_i = t_j = \tilde{t}_j,\ i.e.\ \triangle\, t_{processing} = 0$$

An attribute value can be acquired on demand but the procedure costs some resources, and thus, only a part of attribute changes can be detected. Time required to pull attribute and evaluate predicates is negligible.

As example, assume the network of sensors provides the current location of the Grid user for Reference Monitor enforcing the policy given in Table 4.2. Sensors have limited resources (power, bandwidth, memory), and consequently, Reference Monitor pulls a fresh value of the location attribute only once per hour. Even if the attribute does not satisfy the policy during this hour, Reference Monitor will make the incorrect access decision and allow to continue the resource usage.

From the prospective of Reference Monitor, unintentional uncertainty of type freshness I means that

$$\mathbf{Pr_{freshI}}[G_{(t_{permit}:\tilde{t}_{decOnk})} \mid G_{t_{attrOn1}},\ G_{t_{attrOn2}},\ ...,\ G_{t_{attrOnk}}] < 1$$

i.e. always exists a possibility of a policy violation between access control checks despite the fact that all pulled attributes satisfy the policy.

Freshness II implies that an attribute may change during inevitable delays $\triangle\ t_{processing} > 0$. Inevitable time delays are paid for attribute delivery and decision making. Delivery from Attribute Provider to Reference Monitor is time costly due to a distributed nature of Grid and a network latency. Decision making can also affect the enforcement of policies based on volatile attributes.

From the prospective of Reference Monitor, unintentional uncertainty of type freshness II means that

$$\forall(\langle a_j^{obs}, t_j\rangle, \tilde{t}_j) \in ObservedAttr:\ t_j < \tilde{t}_j$$
$$for\ access\ control:\ \mathbf{Pr_{freshII}}[G_{\tilde{t}_j} \mid G_{t_j}] < 1$$
$$for\ usage\ control:\ \mathbf{Pr_{freshII}}[G_{(t_j:\tilde{t}_j)} \mid G_{t_j}] < 1$$

Freshness III introduces a scenario where a current value of an attribute is uncertain since some update queries are pending and might not be committed by the time when an access decision should be made. In this case, Attribute Provider sends to Reference Monitor two attributes: the first is the last certain attribute value and, the second contains *some additional information* on how the current attribute value varies from the given value:

$$(a_i^{real}, t_i), (\triangle\ a_i^{real}, t_{i+1}),\ where\ t_i < t_{i+1}$$

Accordingly, Reference Monitor receives:

$$(\langle a_i^{real}, t_i\rangle, \tilde{t}_j), (\langle \triangle\ a_i^{real}, t_{i+1}\rangle, \tilde{t}_j)$$

Time required to deliver attribute and evaluate predicates is negligible, i.e. $t_{i+1} = \tilde{t}_j$.

As an example, assume a policy which allows Grid users with a good reputation to submit a huge number of applications for execution in Grid. The reputation is updated only when the execution is ended and the system receives feedback from a

resource provider. Applications can run concurrently and each single execution can be long-lived and lasts days or even weeks. The access decision to submit a new job is based on the reputation value dated by the last registered feedback and on the number of applications currently running on the user's behalf. Indeed, the ongoing applications can be malicious but this fact can be discovered only afterwards. The only way to obtain the fresh reputation value is to block the access until all running applications terminate. Instead, the system has to be set up to make an access decision with some uncertainty on the current reputation of the Grid user. For the given example, $\triangle\, a_i^{real}$ specifies how many applications were submitted for execution in interval $(t_i : t_{i+1})$ and are currently ongoing.

From the prospective of Reference Monitor, unintentional uncertainty of type freshness III means that:

$$for\ access\ control:\ \mathbf{Pr_{freshIII}}[G_{\tilde{t}_j} \mid G_{t_i}] < 1$$
$$for\ usage\ control:\ \mathbf{Pr_{freshIII}}[G_{(t_i:\tilde{t}_j)} \mid G_{t_i}] < 1$$

Unintentional uncertainties related to freshness of attributes can be seen as particular cases of *timeliness* and *currency* factors from Bouzeghoub and Peralta [24]. Freshness of the first type relates to the problem of defining the frequency of updates (timeliness), while freshness of the second and third types is caused by natural delays in delivery of the authorization information (currency).

Correctness of real attribute values is affected by additive noises that usually exist in case of non-accurate measurements. For example, the location attribute can be calculated only with a given precision. Thus, observed attributes might differ from the corresponding real counterparts:

$$\forall(\langle a_j^{obs}, t_j\rangle, \tilde{t}_j) \in ObservedAttr,\ \exists(a_i^{real}, t_i) \in RealAttr$$
$$s.t.\ a_j^{obs} \neq a_i^{real},\ and\ t_i = t_j = \tilde{t}_j,\ i.e.\ \triangle\, t_{processing} = 0$$

From the prospective of Reference Monitor, unintentional uncertainty of type correctness means that:

$$\forall j, \mathbf{Pr_{corr}}[G_{t_j}] < 1$$

i.e. there exists the probability that the observed attribute $(\langle a_j^{obs}, t_j\rangle, \tilde{t}_j)$ does not satisfy the security policy even at time of issuance t_j.

Trustworthiness of Attributes

Intentional uncertainties impose that observed attributes do not correspond to the real attributes, and trustworthiness of attributes used to produce the access decision is not guaranteed. These uncertainties appeared as a result of altering attributes by Attribute Provider or as the result of attacks occurred during delivery, storing, etc.

Current approaches guarantee only integrity of an attribute by validating a signature of the entity which signs the attribute assertion, but this does not guarantee trustworthiness of the attribute value.

Notice, that uncertainties related to trustworthiness assume that either an attribute value, or a time of issuance, or both can be modified by Attribute Provider. Indeed, on each attribute pull request Attribute Provider might respond with the same attribute value which always satisfies a security policy.

From the prospective of Reference Monitor, unintentional uncertainty of type trustworthiness means that:

$$\forall j, \mathbf{Pr_{trust}}[G_{t_j}] < 1$$

i.e. there exists the probability that the observed attribute a^{obs} does not satisfy the security policy at any time.

4.2.3 Risk-aware Decision Making

Reference Monitor has to envisage and tolerate the impact of making access decisions in the case of uncertain attributes. Making the access decision, Reference Monitor tries to choose between two *alternatives* (grant access and deny/revoke access) only one which is as good as possible. Good means that Reference Monitor grants access to legible requestors if the security policy is satisfied, and forbids access to unauthorized entities if the security policy is violated. Unfortunately, when real attribute values are unknown, Reference Monitor is unable accurately to infer whether the security policy is violated or satisfied based on observed attributes, and, thus, to choose the good alternative. There are four possible scenarios of how Reference Monitor can act processing only observed attributes:

- *True Positive*: to grant access when the security policy is really satisfied;
- *False Negative*: to grant access when the security policy is really violated;
- *False Positive*: to deny access when the security policy is really satisfied;
- *True Negative*: to deny access when the security policy is really violated.

True Positive and *True Negative* correspond to a good-chosen alternative, while *False Negative* and *False Positive* are erroneous access decisions. Each scenario is associated with a utility, i.e. possible losses and benefits, which Reference Monitor loses/gains when the scenario happens. The utilities are assigned during the security policy design and can be either qualitative or quantitative. Here only quantitative utilities are considered and are named as *costs*. For a better representation, Table 4.3 shows a utility (cost) matrix where alternatives are represented as rows of the matrix, real states of the world as columns and cells of the matrix represent costs.

An access decision is made by comparing the benefits of granting or denying access and choosing the alternative which surmises the best profit. For instance, in

	Policy is satisfied	Policy is violated
Grant access	$C_{00} = 20$	$C_{01} = -100$
Deny access	$C_{10} = -5$	$C_{11} = 10$

Table 4.3: Utility (Cost) Matrix

the case of a correct policy enforcement (i.e. when observed attribute values are the same as real attribute values and the policy is satisfied), the benefits of granting access are 20, whereas losses to deny access are -5. Definitely, Reference Monitor should choose the *'grant access'* alternative.

Usually, observed and real attributes are not equal, and Reference Monitor has insufficient information to reason about the security policy satisfaction or violation. If Reference Monitor is able to treat uncertainties associated with observed attributes by computing the *probability* **Pr** of the security policy violation (satisfaction), it means that the access decision is made under *risk*.

Definition 8. (Risk) *Risk is the possibility of suffering harm or loss [10] and is defined as follows :*

$$risk_{ij} = \mathbf{Pr}[A] \times cost_{ij}$$

where

$$A \in \Omega = \{\text{“}G : \textit{ policy is satisfied}\text{”}, \text{“}\overline{G} : \textit{ policy is violated}\text{”}\}, \textit{ and } \mathbf{Pr}[\Omega] = 1$$

A probability-weighted utility theory [46] provides a method to make an access decision under risk. The idea behind this method is almost the same as already has been presented and Reference Monitor compares benefits and losses of each alternative but *weighted* on a probability of a security policy violation/satisfaction.

Definition 9. (Risk-aware access decision) *Let Reference Monitor possesses a probability of a security policy satisfaction (or violation) and a cost matrix. Reference Monitor*

$$\textit{grants access if } -risk_{grant} < -risk_{deny}$$

or

$$\textit{denies/revokes access if: } -risk_{grant} \geq -risk_{deny}$$

where

$$risk_{grant} = \mathbf{Pr}[G] \cdot C_{00} + (1 - \mathbf{Pr}[G]) \cdot C_{01}$$
$$risk_{deny} = \mathbf{Pr}[G] \cdot C_{10} + (1 - \mathbf{Pr}[G]) \cdot C_{11}$$

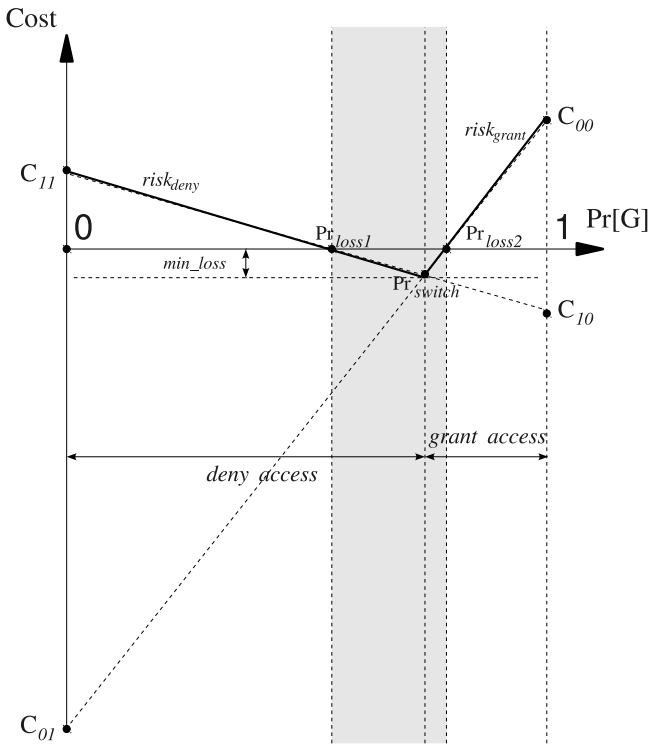

Figure 4.10: Risk-aware access decision

Cost Matrix

The *policy level* of our framework assumes that a policy designer is responsible for specifying a cost matrix for each security policy which is engineered using remote attributes with observable mutability. The cost matrix entries can be updated only by some administrative actions of the policy designer (administrator).

The cots matrix does not depend on usage time and specifies the exact benefits and losses the system eventually gains (from a complete usage session, from the beginning till the usage is over). Thus, semantics of a cost matrix corresponds to 'pay-per-usage' attributes. For instance, if a resource provider donates resources to Grid, the maximum losses C_{01} are equal to costs of the resources.

It is very difficult to determine a cost matrix for every security policy and every object it protects. Some heuristics can be used on selecting costs. For instance, losses caused by allowing access to malicious users are surmised to be the severest, wherefore Table 4.3 contains $C_{01} = -100$. Moreover, if Reference Monitor behaves correctly, the profit should be positive, i.e. $C_{00} > 0, C_{11} > 0$ and negative in the case of the erroneous access decisions, i.e. $C_{01} \leq 0, C_{10} \leq 0$. Finally, in the most cases where a resource provider does not guarantee any quality of service, $C_{10} = 0$ and each equivocal request is better to deny without any possible harm.

There are some other interesting parameters that can be mined from the cost

matrix. Figure 4.10 shows how a risk-aware access decision is made. Intersection of alternatives identifies a probability value when Reference Monitor switches an access decision from deny to grant. Reference Monitor grants access if $\mathbf{Pr}[G] \in (\mathbf{Pr_{switch}}, 1)$, and denies otherwise. For the cost matrix in Table 4.3 this probability is $\mathbf{Pr_{switch}} = 0.81$.

Assume, that the probability of the security policy satisfaction calculated by Reference Monitor and used to produce risk-aware access decision is correct and always remains in $\mathbf{Pr}[G] \in (\mathbf{Pr_{loss1}}, \mathbf{Pr_{loss2}})$. In this case, for the given cost matrix the resource provider will always receive losses in *average*. This interval can be reduced or eliminated by tuning initial costs or remains if the resource provider is motivated to have losses but to guarantee a certain quality of service.

Average losses or benefits accumulated by the resource provider are specified as *average_loss* and can be estimated as follows. First, assume the scenario when the probability of the security policy satisfaction calculated by Reference Monitor always remains the same and equals to $\mathbf{P_{switch}}$. Accordingly to our model, Reference Monitor should deny each access request where $\mathbf{Pr}[G] = \mathbf{Pr_{switch}}$. Thus, the outcome of each policy enforcement would be C_{11} if access really was requested by an unauthorized user or C_{10} - if the user was eligible but Reference Monitor denied access to the resource. Assuming, that the probability of the security policy satisfaction is calculated correctly, the outcome of a single policy enforcement can be considered as a random variable Y and mean of this variable $E[Y]$ equals to *average_loss*

$$average\_loss = E[Y]$$

$$E[Y] = \sum_{\forall y} y \cdot \mathbf{Pr}[y] = C_{11} \cdot (1 - \mathbf{P_{switch}}) + C_{10} \cdot \mathbf{Pr_{switch}}$$

For this particular case, *average_loss* is the minimum loss the resource provider might suffer during risk-aware access and usage control. In Figure 4.10 *average_loss* is specified as *min_loss*.

Usually, the probability of the security policy satisfaction $\mathbf{Pr}[G]$ calculated by Reference Monitor for the given access request might varies. Assume, that this probability can be considered as a random variable X taking any possible value from 0 to 1. In this case, the outcome Y can be expressed through X and *average losses* can be calculated as follows:

$$Y = \begin{cases} C_{11} \cdot (1 - X) + C_{10} \cdot X, & where\ X \leq \mathbf{Pr_{switch}} \\ C_{01} \cdot (1 - X) + C_{00} \cdot X & where\ X > \mathbf{Pr_{switch}} \end{cases}$$

$$E[Y] = C_{11} + C_{01} - (C_{11} - C_{10}) \cdot E[X \leq \mathbf{Pr_{switch}}] + (C_{01} - C_{00}) \cdot E[X > \mathbf{Pr_{switch}}]$$

Probability of Correct Policy Enforcement

The probability that the policy is satisfied and enforced correctly for ***access control*** scenarios is given by (see Definition 6)

$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr_{pre}}[G_{t_{attr}} \cdot G_{\tilde{t}_{decPre}}]$$

or using conditional probabilities

$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr_{pre}}[G_{t_{attr}}] \cdot \mathbf{Pr_{pre}}[G_{\tilde{t}_{decPre}} \mid G_{t_{attr}}] \tag{4.3}$$

This formula shows what the probability $\mathbf{Pr_{pre}}[G_{t_{attr}}]$ corresponds to trustworthiness and correctness of attributes, and $\mathbf{Pr_{pre}}[G_{\tilde{t}_{decPre}} \mid G_{t_{attr}}]$ - to freshness of attributes. Usually, all types of uncertainties might exist in the system and summing these uncertainties corresponds to multiplication of their corresponding probabilities. How to calculate a probability corresponding to a specific uncertainty will be discussed later.

The probability that the policy is enforced correctly by time $\tilde{t}_{decOnk}$ for ***usage control*** scenarios is given by (see Definition 7)

$$\mathbf{Pr_{on}}[G] = \mathbf{Pr_{on}}[G_{|t_{permit} - \tilde{t}_{decOnk}|}]$$

Suppose that observed attributes by Reference Monitor are

$$G_{t_{attrOn1}},\ G_{t_{attrOn2}},\ ...,\ G_{t_{attrOnk}}$$

then

$$\mathbf{Pr_{on}}[G] = \mathbf{Pr_{on}}[G_{(t_{permit}:t_{attrOn1})} \cdot G_{t_{attrOn1}} \cdot G_{(t_{attrOn1}:t_{attrOn2})} \cdot G_{t_{attrOn2}} \cdot ... \cdot G_{t_{attrOnk}} \cdot G_{(t_{attrOnk}:\tilde{t}_{decOnk})}]$$

or

$$\mathbf{Pr_{on}}[G] = \mathbf{Pr_{on}}[G_{(t_{permit}:\tilde{t}_{decOnk})} \cdot G_{t_{attrOn1}} \cdot G_{t_{attrOn2}} \cdot \ ... \ \cdot G_{t_{attrOnk}}]$$

and using conditional probabilities

$$\mathbf{Pr_{on}}[G] = \mathbf{Pr_{on}}[G_{(t_{permit}:\tilde{t}_{decOnk})} \mid G_{t_{attrOn1}},\ G_{t_{attrOn2}},\ ...,\ G_{t_{attrOnk}}] \cdot \mathbf{Pr_{on}}[G_{t_{attrOn1}} \cdot G_{t_{attrOn2}} \cdot \ ... \ \cdot G_{t_{attrOnk}}]$$

This formula shows what the first probability corresponds to freshness of attributes, and the second probability - to correctness and trustworthiness of attributes.

If trustworthiness and correctness remains constant in time, e.g. the attribute is measured always with the same precision, the probability of the correct policy enforcement by time $\tilde{t}_{decOnk}$ is given by

$$\mathbf{Pr_{on}}[G] = \mathbf{Pr_{fresh}}[G_{(t_{permit}:\tilde{t}_{decOnk})} | G_{t_{attrOn1}}, G_{t_{attrOn2}}, ..., G_{t_{attrOnk}}] \cdot (\mathbf{Pr_{corr-trust}}[G_t])^k \tag{4.4}$$

Important result shown by this formula is that the probability of correct policy enforcement (in presence of all type of uncertainties) decays in exponential time in number of policy checks.

Definition 10. (**Effective Enforcement of Usage Control**) *Reference Monitor effectively enforces the security policy under uncertainties by time* $\tilde{t}_{decOnk}$ *if the probability of the correct enforcement of the security policy depends only on the time passed since the last attribute was observed:*

$$\mathbf{Pr_{on}}[G] = \mathbf{Pr_{on}}[G_{(t_{attrOnk}:\tilde{t}_{decOnk})} \mid G_{t_{attrOnk}}]$$

Proposition 2. *If only uncertainties of type freshness II or freshness III do exist in the system, then Reference Monitor enforces the security policy effectively.*

Proof This comes from the definition.

4.2.4 Computing Probabilities

A model of how an attribute changes in time is required to compute a probability of a policy satisfaction. The attribute mutability in our approach is modeled as a *stochastic process*, and Reference Monitor knows parameters of this process. Based on this information, this subsection shows how to compute the probability of the correct policy enforcement in the presence of uncertainties of several types.

Probabilistic Model of Attribute Mutability

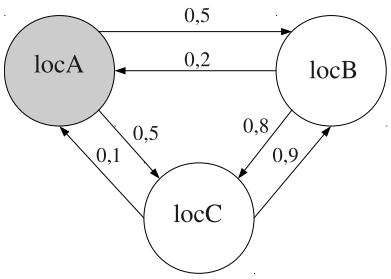

Figure 4.11: Markov Chain for Location Attribute

Discrete-time Markov chain [11, 86] is applied to model changes of remote attributes with observable mutability. States in this chain are possible values of the considered attribute and transitions are possible changes of the attribute. The next attribute value depends only on the current value.

A Markov chain for the location attribute (see the security policy example given in Table 4.2) is depicted in Figure 4.11. The attribute domain consists of three values $ATTR = \{locA, locB, locC\}$, and the Markov chain has three states - $\{A, B, C\}$, accordingly.

Suppose, (a_i, t_i) corresponds to the current attribute value, and the consecutive attribute value is described by (a_{i+1}, t_{i+1}). The one-step transition probability specifies the probability that the current attribute $a_i = locB$ will change the value to $a_{i+1} = locA$ and is denoted by

$$Pr_{BA}^{(i,i+1)} = Pr[a_{i+1} = locA \mid a_i = locB]$$

and it equals to 0.2 for the Markov chain given in Figure 4.11.

The overall possible changes of the attribute are described by the one-step transition matrix which is formed by arranging one-step transition probabilities into the matrix

$$\mathbf{Prob}^{(i,i+1)} = \begin{pmatrix} Pr_{AA}^{(i,i+1)} & Pr_{AB}^{(i,i+1)} & Pr_{AC}^{(i,i+1)} \\ Pr_{BA}^{(i,i+1)} & Pr_{BB}^{(i,i+1)} & Pr_{BC}^{(i,i+1)} \\ Pr_{CA}^{(i,i+1)} & Pr_{CB}^{(i,i+1)} & Pr_{CC}^{(i,i+1)} \end{pmatrix}$$

Assume that one-step transition probabilities do not change during some period of time, i.e. the Markov chain is *time-homogeneous* [50]

$$\forall i : \mathbf{Prob}^{(i,i+1)} = \mathbf{Prob}_{location} = \begin{pmatrix} 0 & 0.5 & 0.5 \\ 0.2 & 0 & 0.8 \\ 0.1 & 0.9 & 0 \end{pmatrix}$$

The n-step transition probability specifies the probability that the current attribute $a_i = locB$ will change the value to $a_{i+n} = locA$ after n changes and is denoted by

$$Pr_{BA}^{(n)} = Pr[a_{i+n} = locA \mid a_i = locB]$$

By analogy to the one-step case, the n-step transition probability matrix is denoted by $\mathbf{Prob}^{(n)}$ and for time-homogeneous Markov chains this matrix can be calculated using Kolmogorov-Chapman equation

$$\mathbf{Prob}^{(n)} = \mathbf{Prob}^n$$

where $\mathbf{Prob}^n$ specifies the matrix in power n.

Additionally to the transition matrix, it is worth to know the amount n of attribute changes occurred in the time period between t_i and t_j. Our approach uses the Poisson distribution to determine the number of attribute changes in the time interval

$$Pr_p(n) = \frac{\lambda^n e^{-\lambda}}{n!}$$

where $\lambda = (t_j - t_i) * n_{mean}$, and n_{mean} is the average amount of transitions in a unit of time.

One might observe that the discrete-time Markov chain, describing the attribute changes, together with the Poisson distribution, describing the number of changes in the time interval, do form the *continuous-time Markov chain*. But for the sake of convenience, this explicit separation is used here.

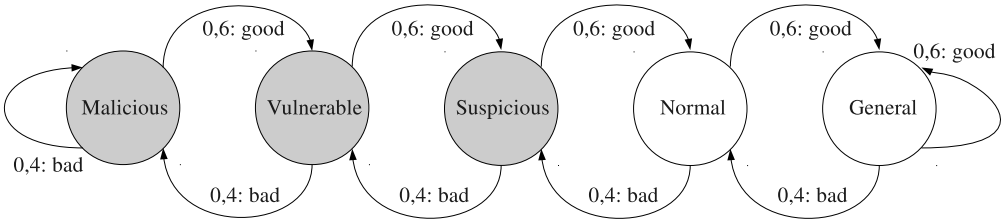

Figure 4.12: Markov Chain for Reputation Attribute

Another example of the Markov chain for the reputation attribute (see the example for the freshness III in subsection 4.2.2) is depicted in Figure 4.12. The Markov chain has five states and the state 0 corresponds to the *malisious* attribute value, 1 - *vulnerable*, 2 - *suspicious*, 3 - *normal*, and 4 - *general*.

Initially, the Grid user has *normal* reputation which will be updated to *general* if the user's application will be executed in Grid successfully and will never violate security policies of the resource provider, or the reputation will become *suspicious* - if the application execution will be terminated due to the security policy violation. In long term prospective, the reputation of the Grid user might take any of the five possible values.

Suppose, that the probability of a successful termination of a singe application remains constant and equals to 0.6. In this case, the transition probability matrix of the Markov chain describing the reputation attribute is given by

$$\mathbf{Prob}_{reputation} = \begin{pmatrix} 0.4 & 0.6 & 0 & 0 & 0 \\ 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}$$

Obtaining the transition probability matrices for all wanted attributes is a hard task in practice. Assume, that these matrices can be estimated using statistics of past interactions with Grid users. The best way is to find the statistics for a specific user and derive the probabilities about it directly. This approach is useful if long-term interactions with the Grid user do exist. Another way is to get the value from other sources (e.g. other data providers) or central authorities (similar to credit bureaus in banking). Finally, if interactions with users are very short then similar users can be grouped and statistics can be collected to the whole group.

Access Control in Presence of Freshness III Uncertainty

Suppose, Reference Monitor knows the Markov chain modeling behavior of the attribute reputation. This information and observed attributes are used to calculate

the probability of the security policy satisfaction in the presence of the uncertainty of type freshness III. All other uncertainties are negligible, i.e. the attribute value is correct and trustworthy.

```
tryaccess(s, o, r).                                    line 1
Ppre(reputation = 'normal' or 'general').              line 2
permitaccess(s, o, r).                                 line 3
endaccess(s, o, r)                                     line 4
```

Table 4.4: preA0 Policy

The security policy example for access control scenario **preA0** is given in Table 4.4. The attribute values, i.e. states of the corresponding Markov chain, which violate the security policy are denoted as grey circles in Figure 4.12.

Assume, the Grid user requests the Grid service and Attribute Provider pushes the user's reputation with the initial request. Reference Monitor receives and processes two attributes

$$(\langle a^{real}_{attr}, t_{attr}\rangle, \tilde{t}_{decPre}), (\langle \triangle\ a^{real}_{attr}, \tilde{t}_{decPre}\rangle, \tilde{t}_{decPre})$$

The fists attribute states that at t_{attr} the attribute value was a^{real}_{attr} and, suppose, $a^{real}_{attr} = `general'$. The second attribute shows how many applications $\triangle\ a^{real}_{attr} = n$ were submitted by the Grid user and are currently running.

Based on this information, Reference Monitor should calculate the probability of the policy satisfaction at $\tilde{t}_{decPre}$ (see equation 4.3)

$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr_{freshIII}}[G_{\tilde{t}_{decPre}} \mid G_{t_{attr}}] =$$
$$= \mathbf{Pr_{freshIII}}[a^{real}_{decPre} = `normal' \vee a^{real}_{decPre} = `general' \mid a^{real}_{attr} = `general', \triangle\ a^{real}_{attr} = n]$$

i.e. the probability that starting at the state 4 ($`general'$) after n-steps transition the Markov chain will be in the states 3 or 4.

Suppose, the vector $\mathbf{S}_t$ specifies the probabilities distribution over all states and the element $\mathbf{S}_t[j]$ denotes the probability that the Markov chain at time t is in the state j. For the attribute value $a^{real}_{attr} = `general'$, the vector equals to $\mathbf{S}_{t_{attr}} = (0, 0, 0, 0, 1)$. The value of the vector at time $\tilde{t}_{decPre}$ after n transitions can be found using the Kolmogorov-Chapman equation and will be

$$\mathbf{S}_{\tilde{t}_{decPre}} = \mathbf{S}^T_{t_{attr}} \cdot \mathbf{Prob}^n_{reputation}$$

When the state probability vector $\mathbf{S}_{\tilde{t}_{decPre}}$ is found, the required probability of the security policy satisfaction can be found by summing up the values which belong to states $`normal'$ and $`general'$

$$\mathbf{Pr_{pre}}[G] = \sum_{j \in \{3,4\}} \mathbf{S}_{\tilde{t}_{decPre}}[j] \tag{4.5}$$

As example, consider that the number of transitions sent in the second attribute equals to $n = 2$

$$\mathbf{S}_{\tilde{t}_{decPre}} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}^T \cdot \begin{pmatrix} 0.4 & 0.6 & 0 & 0 & 0 \\ 0.4 & 0 & 0.6 & 0 & 0 \\ 0 & 0.4 & 0 & 0.6 & 0 \\ 0 & 0 & 0.4 & 0 & 0.6 \\ 0 & 0 & 0 & 0.4 & 0.6 \end{pmatrix}^2 = \begin{pmatrix} 0 \\ 0 \\ 0.16 \\ 0.24 \\ 0.6 \end{pmatrix}^T$$

and the probability of the policy satisfaction

$$\mathbf{Pr_{pre}}[G] = \mathbf{S}_{\tilde{t}_{decPre}}[3] + \mathbf{S}_{\tilde{t}_{decPre}}[4] = 0.24 + 0.6 = 0.84$$

The probability $\mathbf{Pr_{pre}}[G] > \mathbf{Pr_{switch}}$ where $\mathbf{Pr_{switch}} = 0.81$ for the cost matrix given in Table 4.3 and consequently Reference Monitor grants the access.

If the number of transitions equals to $n = 4$, the probability of the policy satisfaction will be $\mathbf{Pr_{pre}}[G] = 0.26 + 0.5 = 0.76 < \mathbf{Pr_{switch}} = 0.81$. In this case, Reference Monitor denies the access.

Access Control in Presence of Freshness II Uncertainty

This subsection addresses how to calculate the probability of the security policy satisfaction in the presence of uncertainty of type freshness II. The security policy considered here is the same as in the case of the freshness III.

Meanwhile, in this scenario Reference Monitors observes and processes only one attribute $(\langle a^{real}_{attr}, t_{attr}\rangle, \tilde{t}_{decPre})$, and also should calculate the probability of the policy satisfaction at $\tilde{t}_{decPre}$ (see equation 4.3)

$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr_{freshIII}}[G_{\tilde{t}_{decPre}} \mid G_{t_{attr}}]$$
$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr_{freshIII}}[a^{real}_{decPre} = \text{`}normal' \vee a^{real}_{decPre} = \text{`}general' \mid a^{real}_{attr} = \text{`}general']$$

The number of transitions (e.g. attribute changes) between t_{attr} and $\tilde{t}_{decPre}$ is unknown but has the Poison distribution. Thus, the probability of the security policy satisfaction is the product of probabilities computed for the case with the freshness III but weighed on the probability that exactly n transitions has occurred

$$\mathbf{Pr_{pre}}[G] = \sum_{n=0}^{\infty}(Pr_p(n) \cdot \sum_{j\in\{3,4\}} \mathbf{S}_{\tilde{t}_{decPre}}[j])$$
$$\mathbf{Pr_{pre}}[G] = \sum_{n=0}^{\infty}(\frac{\lambda^n e^{-\lambda}}{n!} \cdot \sum_{j\in\{3,4\}} (\mathbf{S}^T_{t_{attr}} \cdot \mathbf{Prob}^n_{reputation})[j])$$

where $\lambda = (\tilde{t}_{decPre} - t_{attr}) * n_{mean}$, and n_{mean} is the average amount of transitions in a unit of time (assume that Reference Monitor knows this parameter).

It is interesting to compare result for freshness III and freshness II. Assume that $\lambda = 2$, e.g. the number of expected transitions in the interval between t_{attr} and $\tilde{t}_{decPre}$. In this case, the probability of the policy satisfaction $\mathbf{Pr_{pre}}[G] = 0.90 > \mathbf{Pr_{switch}}$ and Reference Monitor grants the access. If $\lambda = 4$, the probability of the policy satisfaction will be $\mathbf{Pr_{pre}}[G] = 0.79 < \mathbf{Pr_{switch}} = 0.81$. In this case, Reference Monitor denies the access. By the way, in both scenarios the probability of the policy satisfaction is higher in the case of the uncertainty of type freshness II.

Usage Control in Presence of Freshness I Uncertainty

This subsection addresses how to calculate the probability of the security policy satisfaction in the presence of uncertainty of type freshness I. The security policy considered here is given in Table 4.2 but only part of usage control is addressed.

In this scenario, usage control starts at t_{permit} and Reference Monitor pulls the attribute value. Suppose, it receives the attribute $(\langle a^{real}_{attrOn1}, t_{permit}\rangle, \tilde{t}_{decOn1})$.

If the observed attribute is trustworthy and correct the probability of the policy satisfaction starting from t_{permit} till $\tilde{t}_{decOn1}$ is given by (see also equation 4.4)

$$\mathbf{Pr_{on1}}[G] = \mathbf{Pr_{freshI}}[G_{(t_{permit}:t_{attrOn1})} \cdot G_{t_{attrOn1}} \cdot G_{(t_{attrOn1}:\tilde{t}_{decOn1})} \mid G_{t_{attrOn1}}]$$

According to the definition of uncertainty of type freshness I $t_{attrOn1} = t_{permit} = \tilde{t}_{decOn1}$ and supposing that $a^{real}_{attrOn1}$ satisfies the policy predicate (e.g. the Grid user is in the location C), this probability equals to 1 and Reference Monitor grants the access.

The reasonable question arises at what time $t_{attrOn2}$ the next attribute should be pulled and the security policy reevaluated? In terms of risk-aware usage control, the attribute should be pulled when the probability of the security policy satisfaction becomes too low

$$\mathbf{Pr_{on1}}[G] = \mathbf{Pr_{switch}}$$

The probability of the policy satisfaction starting from t_{permit} till $t_{attrOn2}$ is given by

$$\mathbf{Pr_{on1}}[G] = \mathbf{Pr_{freshI}}[G_{(t_{permit}:t_{attrOn2})} \mid G_{t_{attrOn1}}] = \mathbf{Pr_{switch}}$$

and $t_{attrOn2}$ can be derived from this equation.

In the following, it is more appropriate to consider the probability of the policy violation rather then satisfaction. This probability in terms of the location attribute values equals to

$$\mathbf{Pr_{freshI}}[a^{real}_t = \text{`}locA', \forall t \in t_{permit} : t_{attrOn2} \mid a^{real}_{attrOn1} = \text{`}locC'] = 1 - \mathbf{Pr_{switch}}$$

This probability tells whether the Markov chain, which models the attribute behavior and starts in the state $a^{real}_{attrOn1}$, can reach the set of states which violate the security policy. If the Markov chain has entered such state, it could never leave

it. These states are called *absorbing* and suppose Abs specifics the set of absorbing states. For the location attribute it equals to $Abs = \{locA\}$.

A new Markov chain should be constructed to find the probability that the original Markov chain has entered one of the absorbing states. In order to convert the original chain to the new one, the probabilities in each row l of the original transition matrix should be changes to values $Pr_{lm} = 0$ if $l \in Abs$, $l \neq m$, and $Pr_{lm} = 1$ if $l \in Abs$, $l = m$. The transition matrix of the new Markov chain for the location attribute is as specified

$$\overline{\mathbf{Prob}}_{location} = \begin{pmatrix} 1 & 0 & 0 \\ Pr_{BA} & Pr_{BB} & Pr_{BC} \\ Pr_{CA} & Pr_{CB} & Pr_{CC} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0.2 & 0 & 0.8 \\ 0.1 & 0.9 & 0 \end{pmatrix}$$

Using the new Markov chain, the probability of the policy violation is calculated in the similar way to the freshness II problem

$$\sum_{n=0}^{\infty}(Pr_p(n) \cdot \sum_{j \in Abs} \mathbf{S}_{t_{attrOn2}}[j]) = 1 - \mathbf{Pr_{switch}}$$

$$\sum_{n=0}^{\infty}(\frac{\lambda^n e^{-\lambda}}{n!} \cdot \sum_{j \in Abs} (\mathbf{S}^T_{t_{attrOn1}} \cdot \overline{\mathbf{Prob}}^n_{location})[j]) = 1 - \mathbf{Pr_{switch}}$$

Since Abs contains only the attribute value $locA$, the time $t_{attrOn2}$ showing when ***the next attribute should be pulled*** can be found from equations

$$\sum_{n=0}^{\infty} \frac{\lambda^n e^{-\lambda}}{n!} \cdot (\mathbf{S}^T_{t_{attrOn1}} \cdot \overline{\mathbf{Prob}}^n_{location})[0] = 1 - \mathbf{Pr_{switch}}$$

$$\lambda = (t_{attrOn2} - t_{attrOn1}) * n_{mean}$$

Obviously, to find $t_{attrOn2}$ some numerical solving methods should be applied.

At this time the policy enforcement becomes too risky and it is very probably that the Grid user has changed his initial location to one which violates the security policy. In order to proceed the usage session, Reference Monitor queries fresh values of the location attribute. Suppose, Reference Monitor can execute two type of queries:

I to pull all changes of the attribute occurred within the interval $(t_{permit} : t_{atrOn2})$

II to pull only the current attribute value $(\langle a^{real}_{attrOn2}, t_{attrOn2} \rangle, \tilde{t}_{decOn2})$

In the case of *the query of the first type*, Reference Monitor reevaluates the security policy at $\tilde{t}_{decOn2}$ by recomputing the probability of the policy satisfaction in the interval from t_{permit} till $\tilde{t}_{decOn2}$

$$\mathbf{Pr_{on2}}[G] = \mathbf{Pr_{freshI}}[G_{(t_{permit}:\tilde{t}_{decOn2})} \mid G_{t_{attrOn1}},\, G_{t_{attrOn1+1}},\, ...,\, G_{t_{attrOn1+i}},\, ...,\, G_{t_{attrOn2}}]$$

where $t_{attrOn2} = \tilde{t}_{decOn2}$.

The first type of query allows to infer whether the policy was actually satisfied all the interval. In fact, this probability will be 0 if there does exist at least one value among pulled attributes which violates the security policy or 1 otherwise. If the policy was satisfied all the interval $(t_{permit} : \tilde{t}_{decOn2})$ *the next attribute query should be scheduled* at $t_{attrOn3}$. The time of this query and all consecutive queries is computed in the same fashion as the time $t_{attrOn2}$. Indeed, such policy enforcement gives an example of *the effective enforcement of usage control*.

The query of the second type allows to infer whether $\mathbf{Pr}[G_{t_{attrOn2}}] = 1$, i.e. the policy is actually satisfied at the querying time. But, there is no evidence that the security policy holden also within the interval $(t_{permit} : \tilde{t}_{decOn2})$. In fact, the Grid user might violated the policy at some time point within the interval and the query of the second type is inadequate to reason about this. Thus, the access decision taken at $\tilde{t}_{decOn2}$ should be aware about the risk of the policy violation within the passed interval.

The probability of the policy satisfaction in interval from t_{permit} till $\tilde{t}_{decOn2}$ for the query of the second type is given by

$$\mathbf{Pr_{on2}}[G] = \mathbf{Pr_{freshI}}[G_{(t_{permit}:\tilde{t}_{decOn2})} \mid G_{t_{attrOn1}},\, G_{t_{attrOn2}}]$$

Assume, that the attribute value queried at $t_{attrOn2}$ equals to $a_{t_{attrOn2}} = locC$. Thus, the Grid user initially was in the location C and at time of the second attribute query he remains in the same location.

The number of transitions n, i.e. attribute changes, in the interval $(t_{permit} : \tilde{t}_{decOn2})$ is unknown. If n is odd, the probability of the policy violation will be 1 because there is no any path for the Markov chain (see Figure 4.11) which starts and finishes in '$locC'$, makes the odd number of transitions, and never visits '$locA'$. Indeed, all paths which contains the location A do violate the security policy. If n is even, then the probability of the policy violation equals to the probability that the Markov chain absorbs in n steps to the location A. Combining two cases, the probability $\mathbf{Pr_{on2}}[G]$ will be

$$\mathbf{Pr_{on2}}[G] = \sum_{n=0}^{\infty} \mathbf{Pr_n},\ \mathbf{Pr_n} = \begin{cases} 0, & if\ n\ is\ odd \\ \dfrac{\lambda^n e^{-\lambda}}{n!} \cdot (\mathbf{S}^T_{t_{attrOn1}} \cdot \overline{\mathbf{Prob}}^n_{location})[0] & if\ n\ is\ even \end{cases}$$

Obviously, the recalculated probability $\mathbf{Pr_{on2}}[G]$ will be less than the probability $\mathbf{Pr_{on1}}[G]$ computed before querying the attribute. Accordingly to the risk-aware usage control theory - the access should be revoked after receiving the new attribute despite the fact that this observed attribute satisfies the security policy too. These scenario is described in Figure 4.8 where the security policy holds when the attributes are pulled but is violated in-between. This interesting result is the consequence of

the characteristics of the Markov chain modeling the attribute changes, and the attempt to preserve the correctness of the policy enforcement.

Obviously, such policy enforcement is rigorous but not effective. Let's consider another approach to estimate the impact of the security policy violation in the interval $(t_{permit} : \tilde{t}_{decOn2})$. Instead of computing the probability of the policy violation, Reference Monitor computes the average time the Grid user might spent in '$locA'$, i.e. in the location which violates the security policy. Assume, it equals to $T^{locA}_{(t_{permit}:\tilde{t}_{decOn2})}$.

Then, to leverage the growth of probabilities and make the risk-aware usage control *efficient* the following rule is used

$$(\mathbf{Pr}[G_{t_{attrOn1}}] = 1) \wedge (\mathbf{Pr}[G_{t_{attrOn2}}] = 1) \wedge (\frac{T^{locA}_{(t_{permit}:\tilde{t}_{decOn2})}}{(\tilde{t}_{decOn2} - t_{permit})} < 1 - \mathbf{Pr_{switch}}) \rightarrow \mathbf{Pr_{on2}}[G] = 1$$

i.e. if the relative policy violation time is less than $(1 - \mathbf{Pr_{switch}})$ and every observed attribute satisfies the security policy, Reference Monitor continues the usage session and calculates the time of the next attribute query assuming that the policy was satisfied in the interval $(t_{permit} : \tilde{t}_{decOn2})$.

Resuming, the risk-aware usage control can be used to pull attribute queries only when they are really needed. Such efficient schedule saves computational and communication resources, on the one hand, and prevents unnoticed failures of policies on the other.

Access Control in Presence of Uncertainty: Trustworthiness

Usually, trust management models deal with *intentional* uncertainties and propose to assign trust values to attributes. Trust models varies on trust metrics, the semantics of trust, how trust is calculated and how a policy based on trust should be enforced.

The approach proposed in [36] introduces a trust model for a role-based trust management framework (RTML) implemented on a mobile platform. RTML is especially suitable for large-scale, distributed systems with decentralized attribute authorities. RTML introduces delegation of credentials[1], linked and parameterized credentials, etc.

Original semantics of RTML credentials was extended with trust weights expressing a quantitative experience-based trust put by the issuer on the assertion. For instance, the credential encoded as $A.f(v) \longleftarrow D$ states that a principal A trusts a principal D on a property with a degree v, where $v \in (0, 1)$. Due to transitive and distributed model of authority supported by RTML framework, the approach allows to infer attributes of a peer and estimate their trustworthiness based on known attributes and relations between them.

[1] Credential is a digitally signed attribute

Suppose a security policy which allows to execute jobs in Grid certified by ACM with a trust value above 0.80. The reference monitor can reason to execute a submitted job having the following credentials. The first states that a principal A (e.g. a requestor and the application producer) guarantees the job trustworthiness $A.isTrustedApplication(1.00) \longleftarrow JavaJob$; the second constitutes that a principal B has some collaborations with the principal A on producing trustworthy jobs with a degree of 0.70, $B.collaborateWith(0.70) \longleftarrow A$; and, finally, the Grid service provider knows that the principal B is the ACM member with a degree 0.99, $ACMauthority.ACMmember(0.99) \longleftarrow B$. The system is able to infer the required credential by combining the chain of the presented credentials: $ACMauthority.isTrustedApplication(0.67) \longleftarrow JavaJob$.

Algorithm to calculate trust weights is based on a semi-ring algebraic structure where two operators are defined to calculate overall trust: linking (multiplication of trust weights) and aggregation (maximum of given trust weights). Therefore, in our example the resulting trust weight is a multiplication of trust weights in the chain and equals to $1.00 * 0.70 * 0.99 = 0.67 < \mathbf{Pr_{switch}}$, and the reference monitor deduces that the job execution should be forbidden and denies the access.

4.2.5 Risk-aware Policy Enforcement

This subsection studies how the security policy enforcement is affected by the presence of uncertain attributes taking into account the main novelty of usage control, i.e. continuity of control.

Non-oblivious Enforcement

Suppose, **preA0** policy (see Table 4.5) based on an immutable attribute but whose trustworthiness is uncertain. The policy grants the access to the Grid storage service if the Grid user provides the self-signed assertion that information he wants to store does not contain data circulation of which is illegal (i.e. it violates third party copyrights).

```
tryaccess(gridUser,gridStorageService,methodStore).                line 1
[(dataIsLegal == true)].                                           line 2
permitaccess(gridUser,gridStorageService,methodStore).             line 3
endaccess(gridUser,gridStorageService,  methodStore)               line 4
```

Table 4.5: The **preA0** Policy with Immutable Attribute

The required attribute *dataIsLegal* is pushed with the initial request, and assume that only trustworthiness of the attribute is uncertain. Reference Monitor calculates the trustworthiness of the attribute and suppose it equals 0.82. The access decision is evaluated only once and assuming that the security policy is accompanied with the cost matrix given in Table 4.3 the access is granted since

$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr_{trust}}[G] = \mathbf{Pr_{trust}}[dataIsLegal = true] = 0.82 > \mathbf{Pr_{switch}}$$

Since the access is granted, the system imposes no further control and the usage session can be ended only on the Grid user demand. But imagine, that during the ongoing access Reference Monitor receives the evidence from some trusted party, that the data stored by the Grid user is illegal. In this case, Reference Monitor understands that the initial decision was incorrect and the bad alternative was chosen. To mitigate this post-completion error, Reference Monitor *recalculates the risk value every time when new knowledge appear* in the system about circumstances under which risk-aware access decisions were taken. This is the crucial feature of the risk-aware access and usage control which imposes that Reference Monitor is not oblivious to every access decision made using uncertain attributes.

For the given example, assume that:

- E denotes the evidence that the stored data is illegal;
- $\mathbf{Pr}[E|G]$ denotes the probability of seeing the evidence E if the data is actually correct, i.e. the evidence is false positive. Suppose this value equals to 0.05;
- $\mathbf{Pr}[E|\overline{G}]$ denotes the probability of seeing the evidence E if the data is actually illegal. This probability always equals to 1;
- $\mathbf{Pr}[G \mid E]$ denotes the probability that the stored data is illegal if the evidence of such violation is present.

Reference Monitor revises previous estimates of the attribute trustworthiness using Bayesian inference and the re-evaluated probability of the policy satisfaction will be

$$\mathbf{Pr_{pre}}[G] = \mathbf{Pr}[G \mid E] = \frac{\mathbf{Pr}[E \mid G] \cdot \mathbf{Pr_{trust}}[G]}{\mathbf{Pr}[E \mid G] \cdot \mathbf{Pr_{trust}}[G] + \mathbf{Pr}[E \mid \overline{G}] \cdot \mathbf{Pr_{trust}}[\overline{G}]} = 0.19$$

In fact, $\mathbf{Pr}[G \mid E] < \mathbf{Pr_{switch}}$, and Reference Monitor revokes access immediately upon receiving the evidence since keeping the usage session active becomes too risky. Notice, that the risk revision could be forward, and backward, i.e. the evidence could state that the stored content is perfectly good. Meanwhile, the approach is the same - once Reference Monitor gets new information, it revises previous risk estimates and access decisions.

In fact, such model of the risk-aware policy enforcement imposes that all policy statements which use remote attributes with observable mutability are complemented with the mutable *risk attribute*, $risk = risk_{deny} - risk_{grant}$. The key point is that the risk is the system-controlled attribute, and the system is always aware about its freshness, correctness and trustworthiness and can guarantee the correct enforcement of the policy based on the risk attribute. Thus, instead of the **preA0** policy Reference Monitor enforces the **preOnA0** policy (see Table 4.6).

```
tryaccess(gridUser,gridStorageService,methodStore).                    line 1
[(system.risk < 0)∧(gridUser.dataIsLegal == true)].                    line 2
permitaccess(gridUser,gridStorageService,methodStore).                 line 3
( endaccess(gridUser,gridStorageService,methodStore)                   line 4
  or                                                                   line 5
( [(system.risk ≥ 0)].                                                 line 6
  revokeaccess(gridUser,gridStorageService,methodStore ) ) )           line 7
```

Table 4.6: The **preOnA0** Policy Example with Risk Attribute

Risk Mitigation

In usage control, a security policy violation triggers a revocation of access. Risk-aware usage control implies that instead of the immediate policy revocation some other system actions can be executed, e.g. querying Attribute Provider the current attribute value. In order to preserve the coherence of the usage control model, these actions are included into the policy specification. These actions, executed in the result of violation of the predicate based on the risk value, are called the *risk mitigation strategies*. These actions could be treated as the special case of *obligations* and are engineered using atomic actions from a set of affordable *countermeasures*. The policy example given in Table 4.2 assumes the following countermeasure actions: `pullAttr(a)` (the *enforcement level* action which pulls the current attribute value from Attribute Provider), and `revokeaccess(s,o,r)` (the *policy level* action which terminates the usage). The policy and its risk mitigation strategy are presented in Table 4.7.

```
tryaccess(s,o,r).                                                      line 1
[(system.risk < 0)∧(a = locB or a = locC)].                            line 2
permitaccess(s,o,r).                                                   line 3
Z;                                                                     line 4

Z =( (endaccess(s,o,r))                                                line 5
  or                                                                   line 6
  ( {[(system.risk ≥ 0)]. mitigate(system.risk) }.Z )                  line 7
  or                                                                   line 8
  ( {[(a = locA)].                                                     line 9
  revokeaccess(s,o,r)} ) )                                             line 10

mitigate(system.risk)::                                                line 11
pullAttr(a)                                                            line 12
if  (system.risk ≥ 0)                                                  line 13
  revokeaccess(s,o,r)                                                  line 14
```

Table 4.7: Risk Mitigation

Depending on the system design, other countermeasures can be presented, e.g.

a suspension of a usage session, sending a notification to the policy administrator, etc.

4.2.6 Discussion and Related Work

The main focus of risk-aware access and usage control is to approximate the continuous control. The presented approach is generic and counts all possible uncertainties associated with observed attributes in the unified model. It assumes a probabilistic model of the attribute changes and gives mathematical model to calculate all types of uncertainties.

So far, the risk-aware access and usage control is focused on attributes used to evaluate predicates. Besides, the security policy also contains *actions*, e.g. attribute updates and obligations whose fulfillment can be uncertain. The risk-aware access and usage control should extended to capture all kind of uncertainties.

Obviously, some drawbacks do exist. First of all, the approach is based on the assumption that the uncertainty can be modeled by the probability of the policy violation and this probabilities are known whereas generally the system possesses only partial knowledge. Moreover, there are inevitable difficulties on assigning the initial cost matrix, difficulties on determining probabilities, implementing risk-mitigation strategies, etc. However, many real attributes (e.g. reputation, location) do behave as a random process with the Markov property.

There are several related work on risk in access and usage control. Aziz et al. [15] assess policies considering different types of risk - operational, combinatorial and conflict of interest. The approach is focused on reconfiguration of policy in a way to reduce its risk and save its strength. Han et al. [45] describe the approach to pre-evaluate security of policy using risk before enforcement. We don't consider composing of policies and assume that they are created in a secure way. Instead, our approach discusses peculiarities of collecting uncertain attribute values and problems connected with this issue.

Several approaches [123, 34, 74] use risk assessment to analyze cost of possible outcomes of access and employ a cost-benefit analysis to make an access decision. These methods consider a static decision making process while our approach analyzes the dynamic behavior of the system.

Few methods describe trustworthiness of policy arguments and update mechanisms. Skalka et al. [106] discussed the approach to evaluate credentials for distributed authorization with risk. Nauman et al. [87] determined trustworthiness of update mechanism analyzing and verifying its behavior. Next to paying attention to trustworthiness of attributes our approach is also focused on their freshness and also explicitly shows how to make a decision using risk assessment.

The approach proposed in [65] empowers UCON model with risk assessment. This paper describes an approach for selection of service providers (data consumer) in a service oriented architecture (SOA). The model of risk-aware usage policy enforcement is devoted to another problem: enforcement of policies by a resource

provider rather by a requestor and making a rational decision about further accesses.

An examples of dealing with uncertain attribute values in UCON is given in [9]. Each remote attribute is associated with a security label which represents the trusted status of the attribute, and could change as the result of the attribute update. Since updates can run on a remote host, the behavior identifies whether the current value of the attribute is trusted within a specific platform. The model examines how to ensure the correct enforcement of the UCON policy particularly if Reference Monitor is placed on the requestor's side.

Chapter 5

Enforcement of Access and Usage Control in Grid

This Chapter details the *enforcement level* of our access and usage control model.

The enforcement level proposes an architecture of a state-full server-side reference monitor which serves as a collection of security mechanisms collaborating together to enforce security policies. Each mechanism is represented by a component in the architecture.

This Chapter presents the overall component-based architecture of the reference monitor on coarse- and fine-grained levels of control (see section 5.1) and interactions between its main components during security policy enforcement (see section 5.2). The rest of this Chapter outlines security requirements which should be counted during the design of the reference monitor (see section 5.3).

5.1 Component-based Architecture of Reference Monitor

This section describes a detailed high-level component-based architecture of the reference monitor. The important aspect here is to identify the main state-full authorization components necessary to enforce coarse-grained and fine-grained access and usage control scenarios and adhere their functionality to UCON peculiarities, i.e. continuity of control and mutuality of attributes. Also, this section outlines supplementary components needed to facilitate security policies enforcement, e.g. components for the efficient attributes management, risk-aware access and usage control, and trust negotiation.

We start by symbolically dividing the reference monitor in two operational parts (see Figure 5.1): coarse-grained c-reference monitor which protects access and usage of Grid service instances; and fine-grained f-reference monitor which performs runtime monitoring of jobs submitted for execution by means of Grid computational services. The objects monitored on the coarse-grained level are Grid services hosted

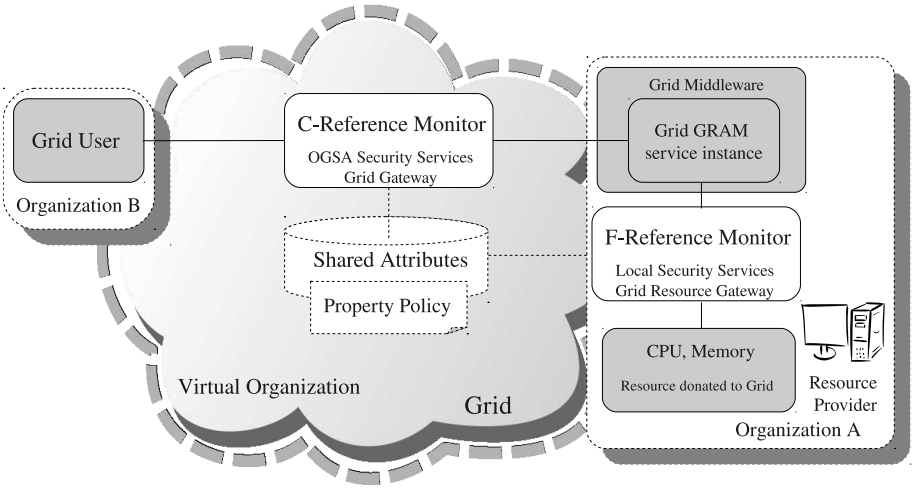

Figure 5.1: Reference Monitor

by computational Grid, while access rights grant access to the methods exposed by these services.

Two levels of control are integrated by sharing attributes used to produce access decisions. Property policy specifies mapping rules among attributes available on different levels. Besides, two levels of control are synchronized on the access revocation, i.e. the Grid service instance terminated on the fine-grained level is automatically revoked on the coarse-grained level and vise versa.

5.1.1 Coarse-grained Level of Control

Figure 5.2 shows the overall architecture of the coarse-grained reference monitor. It consists of two blocks representing a Grid user and Grid-level security services which form the coarse-grained reference monitor.

Grid User

Grid user acts as a subject in access and usage control scenarios and initiates *access requests* to Grid service instances. Since Grid service instances are long-lived resources, at some point of time the Grid user on its discretion might stop the execution of the Grid service instance by sending the *end-access* message.

Besides this basic functionality, the Grid user hosts security services to support trust negotiation with Grid resource providers. **Trust Negotiation Engine** is responsible for carrying out the negotiation protocol specified in Figure 4.2. The protocol runs over a protected channel and utilizes credential negotiation in order to enforce (on the fly) user's and resource provider's mutually satisfiable requirements. Trust Negotiation Engine also maintains security polices relevant to the trust negotiation, i.e. credential and disclosure policies.

Credential Manager complements the functionalities of the trust negotiation engine. It provides interfaces for: (i) validation and verification of credentials received from other parties during trust negotiation, (ii) credential transformation from transport format to logic predicates suitable for policies evaluation.

User Attributes represent a repository where all Grid user's attributes are stored. These attributes can be access by the trust negotiation engine and by other parties. Disclosure of attributes is regulated by the disclosure policy.

Coarse-grained Reference Monitor

The coarse-grained reference monitor receives the service request from remote Grid users, performs the configured security checks, and starts the requested service. The coarse-grained reference monitor consists of (see Figure 5.2): Policy Enforcement Point (C-PEP), Policy Decision Point (C-PDP), Attribute Manager (C-PIP), Sensors, Authorizations Manager, Obligations Manager, Conditions Manager, Trust Negotiation Engine, Risk Manager and Policy Administration Point (PAP).

C-PEP component is able to intercept invocations of security-relevant operations (access requests), suspend them before starting, and interrupt them while in progress. The $tryaccess(s, o, r)$ is transmitted by C-PEP to C-PDP, when it intercepts and suspends a security-relevant operations waiting for the access decision produced by C-PDP. Grid user attributes might be pushed with the initial access request. C-PEP forwards these attributes to Attribute Manager by the $push(attr)$.

C-PEP is responsible for the enforcement of taken access decisions. If C-PDP responds with the $permitaccess(s, o, r)$ the C-PEP creates the Grid service instance and sends the end point reference to this instance to the requesting Grid user. If C-PDP responds with the $denyaccess(s, o, r)$ the C-PEP aborts the access request by notifying the requesting Grid user.

Moreover, once an access request (s, o, r) has been permitted and is in progress, C-PEP should be able to detect when the access to Grid service instance terminates normally. Either Grid user or system environment where the Grid service instance is hosted might stop the execution of the Grid service instance by sending the *endaccess* message to C-PEP. Afterwards, C-PEP issue the $endaccess(s, o, r)$ to C-PDP. If C-PEP receives the $revokeaccess(s, o, r)$ from C-PDP when the access is in progress, it should be able to terminate the usage session by destroying the Grid service instance and notifying the requesting Grid user about the policy violation.

C-PEP might also manage meta-information associated with the usage session, e.g. session id and the current state of the enforced coarse-grained security policy.

C-PDP produces access decisions according to the coarse-grained security policy. When C-PDP receives the access request $tryaccess(s, o, r)$ coming from C-PEP, C-PDP loads the coarse-grained security policy applicable to the given access request.

The coarse-grained security policy is written in POLPA language and represents a process specified via a sequence of allowed actions. C-PDP builds security automa-

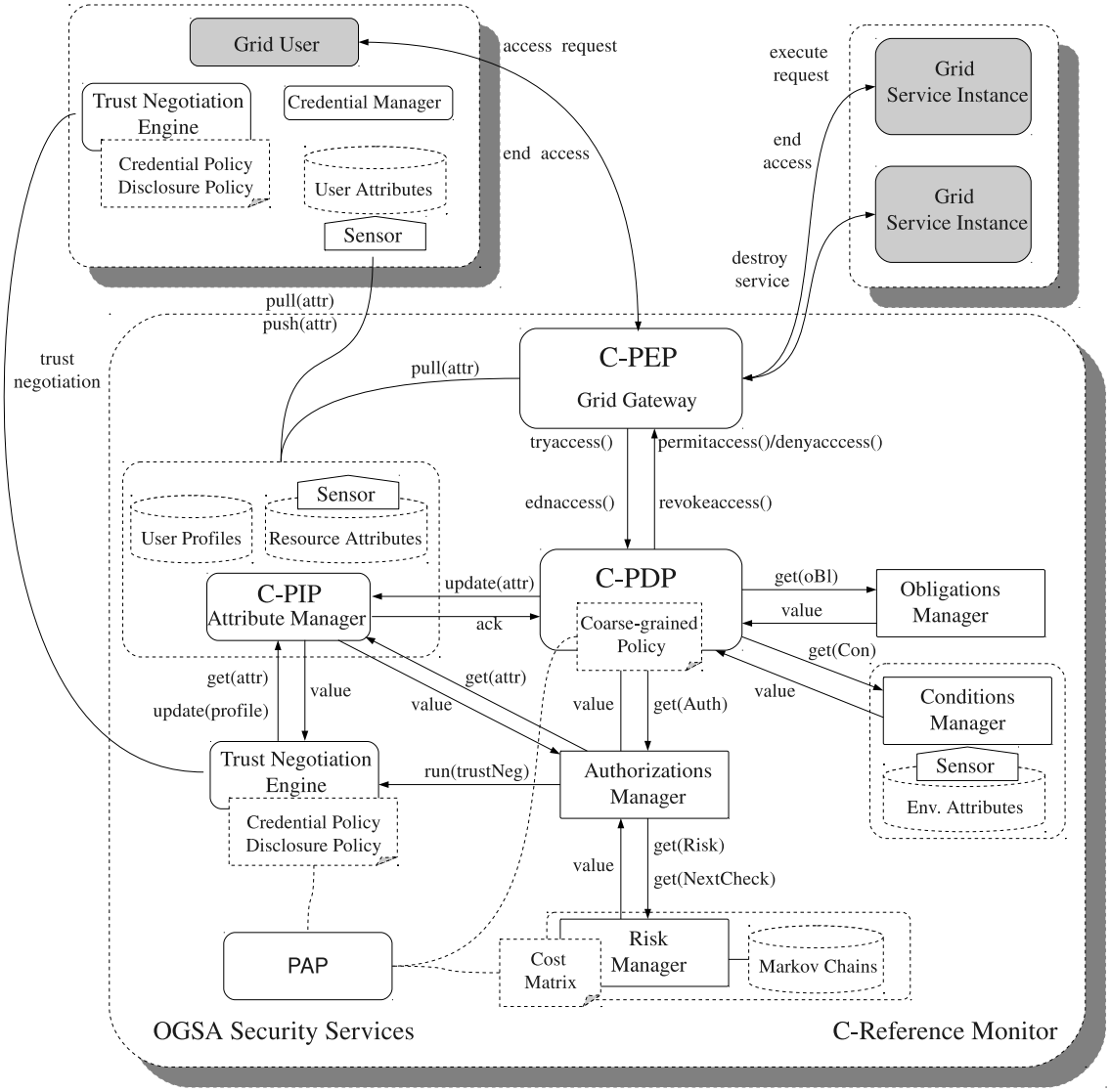

Figure 5.2: Reference Monitor Architecture on Coarse-grained Level

ton that represents the security policy and corresponds to admitted behavior of the system. As an example, assume the security policy represented by the sequence of actions $\alpha.\beta.\gamma$. When the action α is received, C-PDP allows the execution of α, updates the state of the policy and remembers that α has been already executed. Next action C-PDP expects is β.

Initial action consumed by the automaton is the $tryaccess(s, o, r)$ and the pre-authorization starts. Next transitions occur as the result of the performing actions corresponding to checking authorization and condition predicates, to enforcing attribute update and obligation actions. C-PDP contacts Authorizations Manager by the $get(auth)$ to check authorization predicates. If the predicates are satisfied, Authorizations Manager returns the $value = grant$ and C-PDP forces the transition of

the security automaton and execution of the next action. In the same fashion C-PDP interacts with Conditions Manager, Obligations Manager and Attribute Manager. If any of these actions fail, the security automaton cannot transit to the next state and C-PDP issues the $denyaccess(s, o, r)$ and aborts the policy enforcement. Otherwise, C-PDP sends the $pertimaccess(s, o, r)$ to C-PEP and starts the ongoing usage control.

The ongoing usage control phase is processed in the same manner. The security automaton halts when some security policy statements do not hold anymore or C-PDP receives the $endaccess(s, o, r)$ from C-PEP. If the security automaton halts as the result of the policy violation, C-PDP invokes C-PEP to terminate the usage session through the $revokeaccess(s, o, r)$ action. Notice, the $revokeaccess(s, o, r)$ action can be sent by C-PDP to C-PEP before that C-PEP sends the $endaccess(s, o, r)$ action to C-PDP. Active C-PDP and state-full interactions between C-PDP and C-PEP are a main novelty of the UCON model implemented in our framework. Usually, authorization engines in Grid assume a passive C-PDP, i.e. C-PDP that answers only once to access requests.

If there were no policy violations during the enforcement, the security automaton halts when the last action specified in the security policy is enforced. After this point, no further interactions between the authorization components are possible.

C-PIP: Attribute Manager serves as a repository of attributes used to evaluate security policies and its functionality consists in efficient attributes management is order to keep up-to-date attribute values. Attribute Manager interacts with C-PEP, C-PDP, Authorizations Manager, Trust Negotiation Engine, and Sensors.

C-PEP pushes to Attribute Manager initial set of attributes submitted with the access request by the $push(attr)$. C-PDP invokes Attribute Manager in order to update some attributes as a result of the coarse-grained policy enforcement by the $update(attr)$. If Attribute Manager is able to change attribute value, it replies with the *ack* message. Authorizations Manager queries attributes which are required for authorization predicates evaluation by the $get(attr)$. Also, Authorizations Manager might force Attribute Manager to pull fresh attributes from remote repositories, e.g. *remote* attributes of Grid users that have *observable mutability*.

In case of trust negotiation, Attributes Manager captures the functionality of Credential Manager component specified for the Grid user. Attribute manager keeps credentials for mutual trust negotiations and Grid users' profiles of active credentials, performs credentials validation and translation for appropriate format required by other components. Trust Negotiation Engine contacts Attribute Manager in order to update profiles of Grid users participating in negotiation by the $update(profile)$ and to get Grid disclosable credentials requested by Grid users during negotiation by the $get(attr)$.

Sensors are plugged into subject, object and environment attribute repositories. They are activated every time when attributes are changed. The new value of the attribute is forwarded to Attribute Manager if the push model of attributes acquisition exists in the system. Otherwise, sensors are used by Attribute Manager

to pull fresh attribute values.

Obligations Manager is invoked by C-PDP with the $get(oBl)$ and is responsible for the obligations fulfillment. If the obligation is controllable, Obligations Manager can ask the subject to perform it. Instead, if the obligation is observable only, Obligations Manager simply tests it. Obligations Manager returns $value = true$ to C-PDP if the obligation is satisfied, or false otherwise. As an example, an obligation could state that an email of agreement is sent to the obligation subject that must accept the agreement by replying to this mail. In this case, Obligations Manager sends the email of agreement and waits for the reply before returning true to C-PDP. Interactions between Obligations Manager and C-PDP are state-less.

Conditions Manager is invoked by C-PDP with the $get(Con)$ every time the coarse-grained policy requires the evaluation of condition predicates. Conditions Manager observes attributes of environment where the system operates. It is capable to capture any change of attributes done by the execution environment and exploits sensors to collect attributes. As example, Conditions Manager could retrieve the current time, version of Grid middleware installed on Grid user machine, and so on.

Authorizations Manager is invoked by C-PDP with the $get(Auth)$ every time the coarse-grained policy requires the evaluation of authorization predicates.

Authorizations Manager queries Attribute Manager to get fresh attributes values. If the set of received attributes is not sufficient to evaluate authorization predicates, Authorizations Manager asks Trust Negotiation Engine to resolve problem of missed credentials by starting trust negotiation. Notice, trust negotiation is applicable only for pre-authorization scenarios and is not supported for usage control scenarios, i.e. when the access is in progress. When trust negotiation is over, Authorizations Manager queries Attribute Manager again and if the set of attributes is still insufficient it returns $value = false$ to C-PDP. Otherwise, Authorizations Manager invokes Risk Manager with the $get(Risk)$ to calculate the uncertainties associated with the current attribute values. If the risk of the coarse-grained policy violation is low, Authorizations Manager replies with the $value = true$ to C-PDP.

During the ongoing usage control Authorizations Manager evaluates authorization predicates continuously, i.e. each change of attribute triggers the predicates reevaluation. In case of the push attribute acquisition model, new attribute values are pushed to Authorizations Manager by Attribute Manager each time attributes change value. In case of the pull attribute acquisition model, Authorizations Manager asks Risk Manager to schedule when the policy enforcement will be too risky and the next attribute query should be made. When this time elapses, Authorizations Manger forces Attribute Manager to query fresh attribute values. Authorizations Manager stays alive until the predicates are satisfied. When the predicates are violated, Authorizations Manager replies with the $value = false$ to C-PDP. Authorizations Manager might also be stopped during by C-PDP if the $endaccess(s, o, r)$ action was received by C-PDP from C-PEP.

Trust Negotiation Engine functions in the same manner as the corresponding component of the Grid User architecture. In order to achieve interoperability

of negotiations with Grid users, there should be an instance of Trust Negotiation Engine on the Grid side.

Risk Manager serves to compute intentional and unintentional uncertainties associated with remote attributes. Risk Manager is invoked by Authorizations Manager and returns $value = true$ if the current risk value is less than allowed by the *cost matrix*. Also, Risk Manager operates as a scheduler during ongoing usage control and estimates the time point when the policy enforcement becomes too risky and some countermeasures should be taken to minimize the risk. Authorizations Manager is responsible to execute such countermeasures.

C-PAP manages all security policies of Grid Administrator and other stakeholders on the coarse-grained level of control. In fact, it control the following policies:

- *Coarse-grained security policies* which are expressed in POLPA or U-XACML and specify access and usage control scenarios of Grid service instances, Grid services, Grid service workflows;
- *Property policies* which are expressed as first-order logic rules and map attributes between coarse- and fine-grained levels of control;
- *Credential policies* and *disclosure policies* which are expressed as first-order logic rules and used for trust negotiation of the coarse-grained level of control;
- *Cost matrix* which are expressed as an array of values. The cost matrix determines the allowed level of uncertainties associated with attributes used to produce access decision

In conclusion, notice that each architectural component implements the part of the security services functionality specified by OGSA and discussed in the introduction Chapter.

5.1.2 Fine-grained Level of Control

The fine-grained reference monitor performs run-time monitoring of jobs submitted for execution on the machine donating computational resources to Grid. The fine-grained reference monitor consists of (see Figure 5.3): Policy Enforcement Point (F-PEP), Policy Decision Point (F-PDP), Attribute Manager (F-PIP), Sensors, Authorizations Manager, Obligations Manager, Conditions Manager, and Policy Administration Point (PAP). Underlying logic and functionality of the fine-grained authorization components are the same an on the coarse-grained level of control. The components only differ in security policies loaded on each level of control and protect distinct set of objects.

The fine-grained reference monitor in our framework is designed to monitor the execution of Java-based computational jobs and is integrated into Java Virtual Machine (JVM). Objects on this level of control are files, sockets, etc. For example,

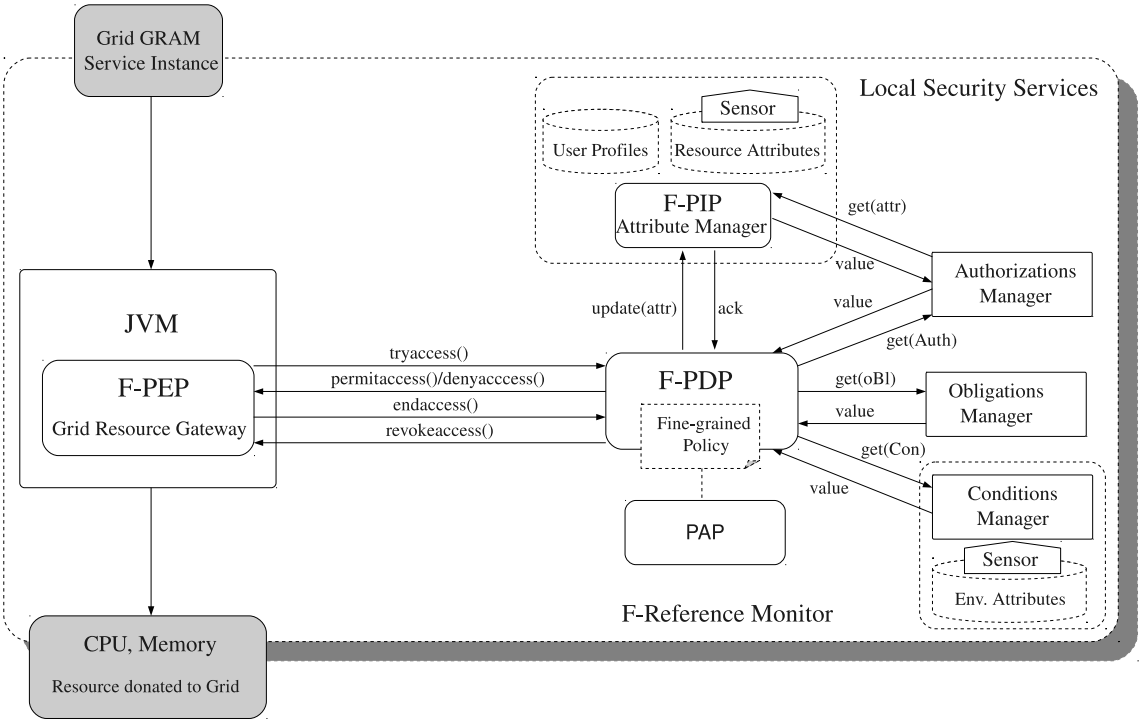

Figure 5.3: Reference Monitor Architecture on Fine-grained Level

if an object is a file, the open, read, write and close operations are the actions that should be monitored by the F-PEP. If the action is not allowed, the F-PEP terminates the job execution.

Besides, some functionality and components are omitted on the fine-grained level of control in order to make the job execution faster. Trust negotiation, risk evaluation are missed and only the push attribute acquisition model exists on the fine-grained level.

Grid resource providers should enforce the following security policies on the fine-grained level:

- *Fine-grained security policies* which are expressed in POLPA and specify access and usage control scenarios of low-level computational resources allocated for Grid service instances;

- *Property policies* which are expressed as first-order logic rules and map attributes between coarse- and fine-grained levels of control;

5.2 Interactions between Components

The message flow, or so-called *authorization protocol* between main components of the architecture is addressed in this section. The security policy enforcement can be

separated in two parts: pre-authorization (or access control scenarios), and ongoing continuous control (or usage control scenarios). Since the security policy enforcement varies slightly on two levels of control, the coarse-grained security policies are discussed.

5.2.1 Enforcement of Access Control Scenarios

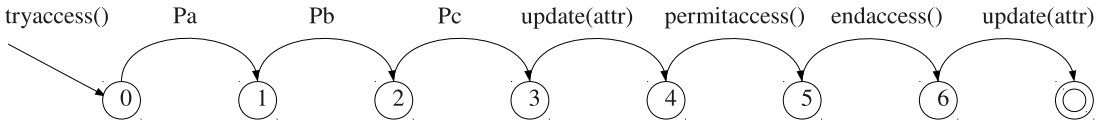

Figure 5.4: preABC13 Security Policy as Security Automaton

`tryaccess(s,o,r).`	line 1
`Pa.`	line 2
`Pb.`	line 3
`Pc.`	line 4
`updates(attr).`	line 5
`permitaccess(s,o,r).`	line 6
`endaccess(s,o,r).`	line 7
`update(attr)`	line 8

Table 5.1: preABC13 Security Policy

Suppose, the **preABC13** Security Policy whose authorization predicates are built on remote, i.e. Grid users, attributes. Such security policy written in POLPA language is given in Table 5.1 and it states that access should be granted if authorization and condition predicates are satisfied, and obligation action is fulfilled. Before granting access and when the usage session is ended some attribute updates are required.

The enforcement of this access control scenario goes as follows (see Figure 5.2):

1. *Step*: a Grid user requests access to a Grid service instance. Assume, no attributes are pushed with the initial request; C-PEP intercepts the request and sends the $tryaccess(s, o, r)$ to C-PDP. Afterwards, C-PEP suspends the access request and awaits for the C-PDP reply;

2. *Step*: C-PDP receives this action and loads the security policy from PAP which corresponds to the access request. Then, C-PDP creates the security automaton that models this policy (see Figure 5.4). Inial state of the automaton is 0 and in order to transit to the next state C-PDP initiates authorization predicates check by invoking Authorizations Manager and awaits for its reply;

3. *Step*: Authorizations Manager asks Attribute Manager attributes required for the predicates evaluation. Attribute Manager replies with the nil message because no attributes were pushed with the initial request. Then, Attribute Manager invokes Trust Negotiation Engine to run negations on the missed attributes and awaits for its reply;

4. *Step*: Trust Negotiation Engine loads from PAP credential and disclosure security policies and starts trust negotiation with corresponding authorization components of the Grid user. During negotiation process, Trust Negotiation Engine contacts Attribute Manager in order to update the Grid user profile with new acquired attributes. When the trust negotiation process is accomplished, Trust Negotiation Engine replies to Authorizations Manager suspended in the step 3;

5. *Step*: Authorizations Manager awakes and queries again Attribute Manager on attributes required for the predicates evaluation. It attributes received during trust negotiation are insufficient and do not satisfy authorization predicates, then the *step access denied* is taken. Otherwise, Authorizations Manager contacts Risk Manager to calculate uncertainties associated with negotiated attributes and awaits for its reply;

6. *Step*: Risk Manager loads the cost matrix from PAP, computes the probability of the authorization predicates violation, and produces the risk-aware access decision. The result is sent back to Authorizations Manager suspended in the step 5;

7. *Step*: Authorizations Manager forwards the result of authorization predicates evaluation to C-PDP suspended in the step 2. If predicates are satisfied, C-PDP forces the transition of the security automaton to the state 1. Then, C-PDP proceeds with invocation of Obligations Manager to fulfill required obligations and awaits for its reply;

8. *Step*: Obligations Manager forces the obligation subject to fulfill required obligation actions. When this is done, Obligations Manager replies to C-PDP suspended in the previous step;

9. *Step*: The next action initiated by C-PDP is checking of condition predicates. It is performed in the same way as obligations fulfilment. When all decision factor are satisfied, the security automaton remains in the state 3 and the next action - update of attributes. C-PDP contacts Attribute Manager to preform this update operation. After the update, C-PDP replies the $permitaccess(s, o, r)$ to C-PEP suspended in the step 1 and idles till the next action is received to trigger the automaton transition;

10. *Step*: C-PEP resumes the suspended access request, creates the Grid service instance and notifies about this the Grid user. Then, C-PEP also idles until the Grid service instance completes its tasks or the Grid user decides to stop the service. When this happens, C-PEP contacts again C-PDP suspended in the previous step with the $endaccess(s, o, r)$;

11. *Step*: C-PDP forces the security automaton to transit to the state 6 and the final action to be executed in this access control scenario is the attribute update. C-PDP asks Attribute Manager to perform it. Then, C-PDP notifies all components that the usage session is over and the access request processing has finished;

12. *Step Access Denied*: this step is taken if during steps 7-9 any of managers return that the particular policy statement does not hold. In this case, C-PDP immediately stops the policy enforcement, destroys the security automaton and notifies C-PEP with the $denyaccess(s, o, r)$. C-PEP forwards this message to the Grid user and rejects the access request. No further communications on this access request are possible.

5.2.2 Enforcement of Usage Control Scenarios

Enforcement of usage control scenarios differ from the enforcement of access control ones due to continuity of control and a need of revocation of violated sessions.

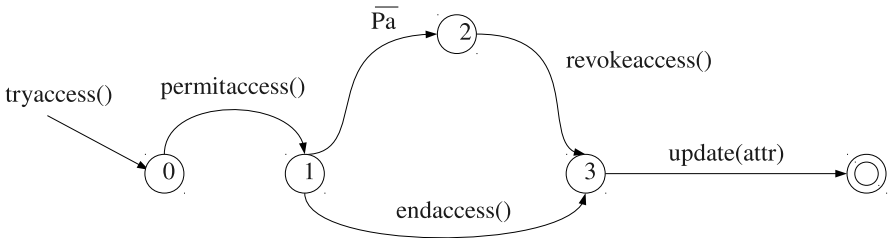

Figure 5.5: onA3 Security Policy as Security Automaton

`tryaccess(s,o,r).`	line 1
`permitaccess(s,o,r).`	line 2
`(endaccess(s,o,r) or`	line 3
$\overline{Pa}$`.revokeaccess(s,o,r));`	line 4
`update(attr)`	line 5

Table 5.2: onA3 Security Policy

Suppose, the **onA3** Security Policy whose authorization predicates are built on remote, i.e. Grid users, attributes with observable mutability. Such security policy

written in POLPA language is given in Table 5.5 and it states that access should be granted when requested. During the ongoing usage, the authorization predicates are evaluated continuously, and when they are not satisfied - the access should be revoked. Either the usage session ends normally or is destroyed as the results of the security policy violation, the attribute updates completes this usage control scenario.

The enforcement of this security policy goes as follows (see Figure 5.2):

1. *Step*: a Grid user requests access to a Grid service instance. C-PEP intercepts the request and sends the $tryaccess(s, o, r)$ to C-PDP. Afterwards, C-PEP suspends the access request and awaits for the C-PDP reply;

2. *Step*: C-PDP receives this action and loads the security policy from PAP which corresponds to the access request. Then, C-PDP creates the security automaton that models this policy (see Figure 5.5). Inial state of the automaton is 0. No any control is required in this usage control scenario and the access should be granted. C-PDP transits to the state 1 and responds with the $permitaccess(s, o, r)$ to C-PEP suspended in the previous step. C-PEP creates the Grid service instance and sends the end point reference of this instance to the requesting Grid user. Then, C-PEP idles. Simultaneously, C-PDP starts the ongoing usage control by invoking Authorizations Manager and waits for its reply.

   Authorizations Manager continuously evaluates authorization predicates. In case of the push attribute acquisition model, Authorizations Manager asks Attribute Manager to push fresh attribute values every time they change. When new attributes arrive, Authorizations Manager asks Risk Manager to compute uncertainties associated with attributes. If the policy enforcement is not too risky, Authorizations Manager awaits until new attributes will be pushed by Attribute Manager and iteratively repeats this procedure until the predicates are satisfied.

   In case of the pull attribute acquisition model, Authorizations Manager gets required attributes from Attribute Manager and asks Risk Manager to compute the time point when the policy enforcement will be too risky. When this time elapses, Authorizations Manager forces Attribute Manager to pull fresh attribute values from sensors. If new attributes satisfy authorization predicates, Authorizations Manager again asks Risk Manager to compute the time point of the next attribute query and predicates reevaluation. Attribute Manager iteratively repeats this procedure until the predicates are satisfied;

3. *Step*: On this step the policy enforcement depends on what happens first: (i) the Grid service instance completes its tasks or the Grid user decides to stop the service; (ii) Authorizations Manager detects the security policy violation.

   If the first alternative happens, C-PEP contacts C-PDP suspended in the previous step with the $endaccess(s, o, r)$, C-PDP stops Authorizations Manager

and consequently, Authorizations Manager terminates interactions with Attribute Manager and Risk Manager. The security automaton which model the security policy transits from the state 1 to the state 3.

If the second alternative happens, Authorizations Manager notifies C-PDP that authorization predicates do not hold anymore. C-PDP transits the security automaton from the state 1 to the state 2. Then, C-PDP issues a $revokeaccess(s, o, r)$ command to the C-PEP and the security automaton transits to the state 3. When C-PEP receives the $revokeaccess(s, o, r)$ from C-PDP, it destroys the Grid service instance and notifies the requesting Grid user about the policy violation.

4. *Step*: Either the first or the second alternative occurs after its execution, the security automaton remains in the state 3. Then, C-PDP forces the final action execution - the attribute update. C-PDP asks Attribute Manager to perform it. Then, C-PDP notifies all components that the usage session is over and the access request processing has finished.

5.3 Security Requirements for Reference Monitor

This section ends this Chapter by describing general design requirements which should be resolved during the implementation of components of the reference monitor.

The reference monitor should implement correct and efficient enforcement of security policies and ensures that unauthorized accesses are prevented and malicious users can not circumvent the policy enforcement. This claim is based on successful implementation of several design principals among which are [99]:

- *Complete mediation*: every access and security-sensitive action to every object must be checked for an authority. The reference monitor is non-bypassable on the way to the accessing resource;

- *Tamper-proof*: the reference monitor cannot be modified or influenced by other processes, i.e. which could approve actions that are not allowed by the security policy. Tamper-proof requirements also encompass the reference monitor which governs access to the security policy itself to guarantee that the enforced policy can not be modified too;

- *Reference validation correctness*: all access decisions do adhere to security policies, and these decisions are correctly implemented.

Chapter 6

Implementation of Access and Usage Control in Globus

This Chapter presents the *implementation level* of our access and usage control model. It discusses the implementation and integration of the *reference monitor* in Globus Toolkit 4.0, the mostly used middleware to setup computational Grid. We integrated our model exploiting the possibility to plugging third-party authorization services, called SAMLAuthzCallout.

The coarse-grained reference monitor is realized to continuously control access and usage of Grid service instances, while the fine-grained level is implemented for the GRAM computational service only. The performance evaluation is given to estimate the overhead posed by our framework. All experiments were executed on Pentium 4 with 2.8GHz and 1GB RAM running Linux. Implementation and results were presented in [30, 31, 58].

6.1 Implementation of Coarse-grained Reference Monitor

This section presents the implementation of the coarse-grained reference monitor. First, it briefly recalls the authorization framework employed in Globus Toolkit (subsection 6.1.1). Then, the realization of the trust negotiation engine is discussed (subsection 6.1.2). In the rest, the implementation of state-full interactions between C-PDP and C-PEP are shown (subsection 6.1.3). The state-full coarse-grained reference monitor is able to capture continuous control over Grid service instances and to revoke instances which violate prescribed security policies.

6.1.1 Native Authorization Framework of Globus

The main component of Globus Toolkit is Globus Container. This is a runtime environment that provides a message processing facilities between Grid users and

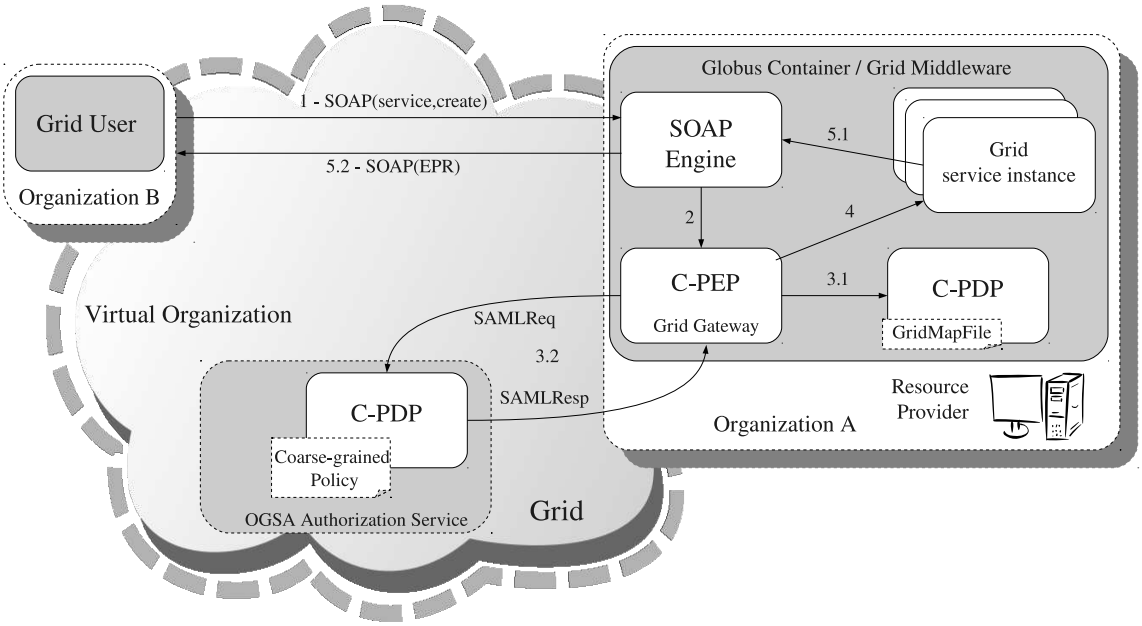

Figure 6.1: Access Control in Globus Toolkit

Grid services, creates and destroys Grid service instances, and controls access and usage of Grid services.

Globus Container processes default access control scenarios as follows (see Figure 6.1):

- *Step 1*: Grid user sends the SOAP[1] message that contains the access request `SOAP(service,create)` to create a new service instance;

- *Step 2*: The message is handled by SOAP Engine and is translated to a format understandable by Globus Container. Each request should be authorized before the execution and SOAP Engine contacts C-PEP for this task;

- *Step 3.1*: The basic access control approach in Globus Toolkit is based on GridMapAuthorization (rf. Chapter 1 and 2). The C-PEP asks C-PDP to check if the Distinguished Name of the Grid user requesting the service is among a predefined list;

- *Step 3.2*: GridMapAuthorization is a very coarse approach and, additionally, Globus Toolkit allows to define alternative authorization services that best suit access control needs. The Open Grid Forum Authorization working group has defined a standard for plugging third-party authorization services and specifies that the communication protocol between the authorization service and Globus Container must conform to the SAML protocol of requesting

[1]SOAP is a message exchange protocol adopted for interactions between Grid entities

authorization assertion and responding to them. Thus, C-PEP creates the authorization chain that includes the GridMapAuthorization and the third-party authorization service.

- *Step 4*: If the federated access decision is grant, C-PEP creates the requested service instance;
- *Step 5*: End Point Reference (EPR) of the created service instance is placed into the SOAP envelope and SOAP Engine sends it to the Grid user.

6.1.2 Integration of Trust Negotiation

We deployed Trust Negotiation Engine (with Credential Manager, rf. Figure 5.2) in both Grid user and Globus Container sides to make the negotiation framework interoperable. We exploited standard Globus Toolkit mechanisms for the seamless integration and plugged-in trust negotiations as the authorization service through SAMLAuthzCallout. Our authorization service is integrated by simply exploiting a Globus configuration feature, i.e. by configuring the Grid service security profile to exploit it. Realization of the authorization service is done in Java [2].

Figure 6.2 shows the negotiation-based authorization life cycle. It consists of the following steps:

- Grid user sends a service request (1) handled by Globus Container. Simultaneously, Trust Negotiation Engine of the Grid user side sends the same request (1') to the counterpart engine embedded into the authorization service. Once step (1') is done, the authorization service establishes an SSL channel (mutual authentication) with the user's Trust Negotiation Engine (1").
- The real trust negotiation starts when the Globus container has processed the service request (in step (1)) and sends a SAML authorization request to the authorization service, step (2). On its side, the authorization service compares the request to those requests received directly from Grid users and starts negotiating with the user that matches to the container's request. We note that step (2) is independent form the occurrence of step (1") but step (3) takes place only if steps (1'), (1") and (2) have taken place.
- The trust negotiation protocol (rf. Figure 4.2) runs over an SSL secure socket connection providing message confidentiality, step 3. The SSL connection with mutual authentication bootstraps a negotiation process with initial identity token exchange. Once the negotiation protocol is initialized the Grid user and authorization server negotiate on the missed attributes required to produce the access decision. Attributes are encoded as X.509 certificate tokens and trust negotiation phase includes exchange, validation and transformation of these certificates to logic predicates.

[2] http://www.interactiveaccess.org/index_v1_1.html

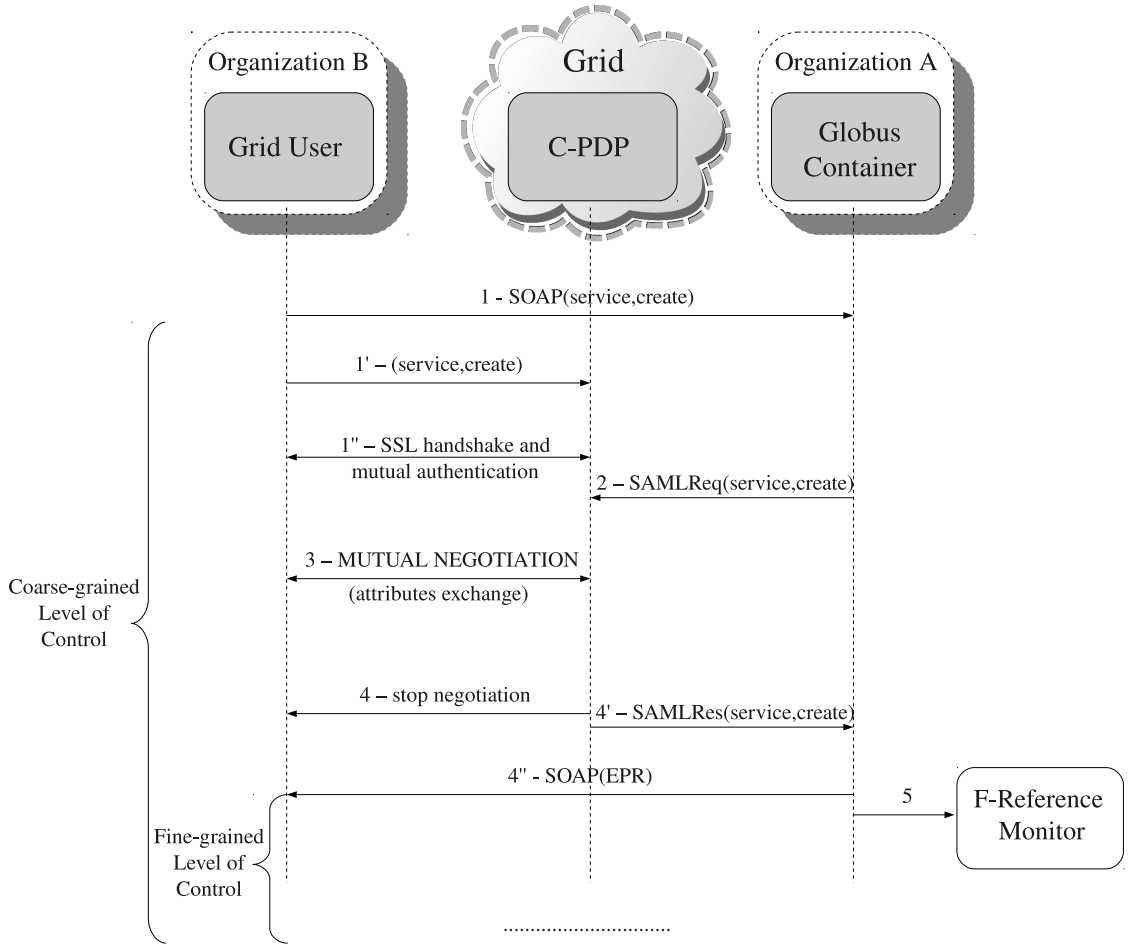

Figure 6.2: Negotiation-based Access Control

- When the negotiation is over and access decision has been taken, step (4), the authorization service sends back to the Globus container a SAML response with the access decision, step (4'). It is the starting point when the Globus container initiates the service creation, step (4"), and the fine-grained control, step (5).

The negotiation process including network support and messages delivery between the trust negotiation nodes was implemented apart from the Globus transportation channels and uses SSL socket connection. Message exchanges in the protocol were custom-defined and optimized for efficient message delivery.

We assume that communications over SSL are sufficiently secure for our model. Thus, Grid user and Globus Container achieve confidentiality and integrity of authorization messages and credentials transmitted during trust negotiations. Credentials encode attributes of peers, and are digitally signed by trusted authorities.

Credentials are compliant with X.509 (v3) attribute certificate standard. All credentials are associated with a particular identity (public key) presented for the

SSL handshake to proof their ownership, and vanish fake credentials. During a negotiation process some of the already presented credentials may expire before the negotiation is completed. To deal with that, Credential Manager keeps two sets of user profiles: raw X.509 certificates, and their logic based equivalent. The former one is re-evaluated on any negotiation step to be performed (i.e., before an access decision is taken), so that if any of the presented certificates expires Trust Negotiation Engine aborts the negotiation process.

Unfortunately, the trust negotiation process might become a subject of denial-of-the-service attacks since it is a relatively heavy computational process (due to its design nature). Anyway, one could define a short enough bounded session time for a round of negotiations to mitigate this kind of attacks. When the session time expires, the session automatically is terminated.

Performance Evaluation

We tested the negotiations overhead on the coarse-grained level of control based on scenarios and security policies presented in Chapter 4 (section 4.1.3 and Figures 4.4, 4.5), and by evaluating it with: (i) increased number of presented credentials, (ii) increased number of negotiated (disclosable) credentials and (iii) simulated two extreme cases of negotiation strategies giving us the boundaries of overall possible negotiation timing.

We will first illustrate typical negotiation exchanges and their time consumption against network delay, certificate validation and access decision making. Figure 4.5 shows two negotiation processes based on the presented Grid access control scenario.

We divide a trust negotiation overhead into (relatively independent) time cases:

- network delay for data transmission over SSL channel;
- certificate validation, verification and conversion to data structures suitable for logic access decision. Attributes are encoded as X.509 certificates;
- logic access decision evaluation based on security policies (authorization predicates checking).

Figure 4.5(a) presents a simple negotiation scenario with just few message exchanges. The client (Grid User) requests to obtain access rights to the free_mathlib library and the server (Computing Center, the Grid resource provider) requires a PhD student credential to grant access. The table below shows the negotiation time (in milliseconds) of the different cases.

network delay	83.2
certificate validation	14.0
access decision	101.5
overall time	287.0

The overall time is counted from the point when a client initiated a request till the time the server replied with a grant message. This time also includes the time for user profile management such as profile generation, update and deletion.

The logic engine was called for access decision evaluation three times - once on the client side and twice on the server side. The submitted certificate was validated once on the server side only.

We count overall negotiation time of all steps done by both server and client sides. However, to calculate time of concurrent processes (notice, that each missed credential is negotiation in the separate system process) we projected all processes on one time line in order to accurately deal with overlapping time. For instance, for two concurrent processes we update the overall time by the difference from the earliest process started till the latest that ended.

Figure 4.5(b) shows a negotiation case with two concurrent negotiations (in concurrent threads), where one of them is drawn in gray. The table below summarizes the measured time performance (in milliseconds).

network delay	182.1
certificate validation	39.0
access decision	380.1
overall time	707.6

In this scenario, the logic access decision was called 10 times, 5 on each side, and the client validated 2 certificates while the server 3.

In either cases, the certificates validation and verification had less time consuming part than the network delay. The conclusion from the first part of the experimentation is that the overall negotiation time is more sensitive to network delay time than to certificate validation. The more interactions during a negotiation the more influence the network delay will have on the overall negotiation. Absolute value of network delay time depends also on the size of the data to be processed. There was no direct way to manage and reduce this value for the given scenario.

The access decision time had the most impact on the overall negotiation process. Looking in Figure 4.1, the access decision depends on two main computations: deduction part (based on the access policy and client's set of credentials) and abduction part (based on the access policy, client's set of credentials and the set of disclosable credentials). We performed two sets of experiments to analyze the access decision behavior with respect to the two logic computations. We used the Grid access control scenario in Figure 4.5(b) and its security policies (rf. Figure 4.4) for the experiments.

First set of experiments focused on measuring access decision time of deduction computation versus an increased number of pushed certificates to be verified, validated and transformed to logic facts. Technically, a client pushes *visaCard* and *ssn*

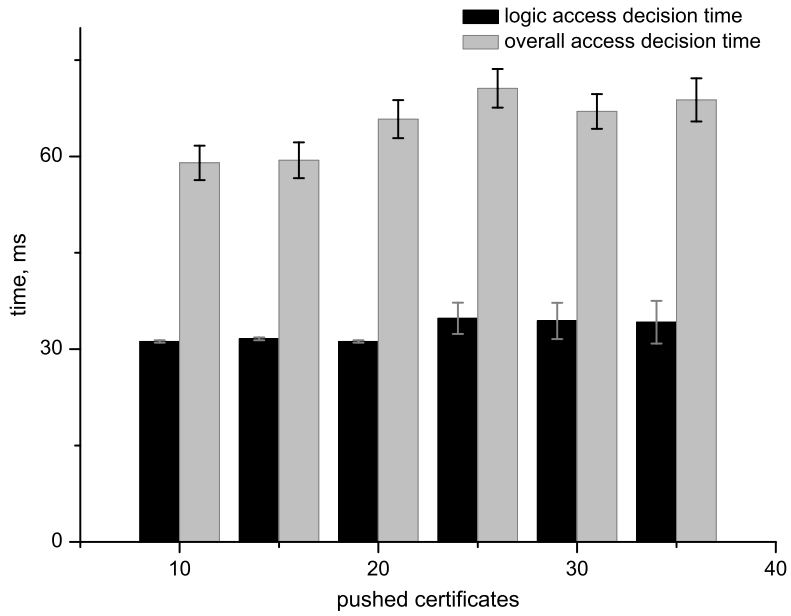

Figure 6.3: Authorization Service Performance with Increased Number of Pushed Credentials

certificates along with other artificially generated certificates along with the initial service request. For this purpose we generated additional X.509 certificates that in combination with the two from the scenario formed the different tests. Figure 6.3 shows the access decision performance measured in milliseconds.

The server invoked the logic engine only once on each trial and replied with grant decision. The average time for the logic access decision almost remained the same while the number of pushed certificates was increased from 10 to 35. The average time was calculated based on 10 repeating measurements on each trial. The measurement error is presented by a cursor arrow on the top of the columns.

The conclusion from the experimentation was that the logic access decision took approximately half of the overall time and did not change during the trials. While, the certificate validation and network delay time (included in the overall decision time) increased during the trials and equally influenced to the overall performance.

Second set of experiments was focused on the pull model where the server determines what credentials are required to establish trust. Here, the disclosure policy determines what credentials are disclosable (available) so that abduction reasoning finds those that are necessary to grant a service request.

We used the scenario of Figure 4.5(b) but this time we modified the disclosure policy (by adding new logic rules) on the server side so that the set of disclosable credentials feed to the abduction reasoning increased on each trial.

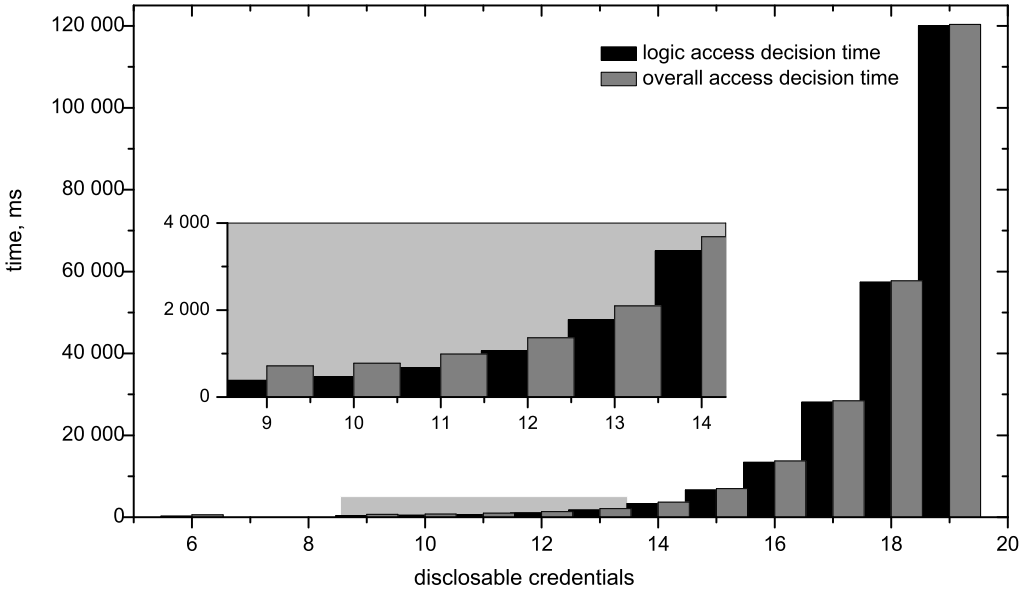

Figure 6.4: Authorization Service Performance Dependence on Disclosure Policy

Figure 6.4 shows the set of measurements performed. On each trial the server computed *visaCard* and *ssn* credentials as missing credentials returned to the client. The access decision time on every trial was very sensitive and varied considerably to the number of disclosable credentials (i.e., a number of logic rules in the server's disclosure policy). The performed tests measured the logic access decision time versus overall decision time.

Abduction reasoning time grew exponentially with increasing disclosable credentials from 9 to 19. With 19 hypotheses (disclosable credentials) the logic engine took approximately 2 minutes (99.7% of the overall time) to compute the missing credentials. Here, the impact of network delay and certificate validation onto the overall time became completely negligible. We expect at around 12 disclosable credentials as a reasonable threshold (approximately a second) for a decision. However, such a limitation also depends on possible negotiation strategies and negotiation session validity. We refer to [35] for in deep analysis and results on abduction problems and complexity.

Next and last set of experiments we performed was on the time performance of the negotiation protocol with respect to possible negotiation strategies in terms of number of client-server interactions.

The sequence of credential exchange during a negotiation process is controlled by negotiation strategies [117, 118]. One can adopt variety of negotiation strategies and privacy settings [122, 108, 19] depending on credential sensitivity, on familiarity

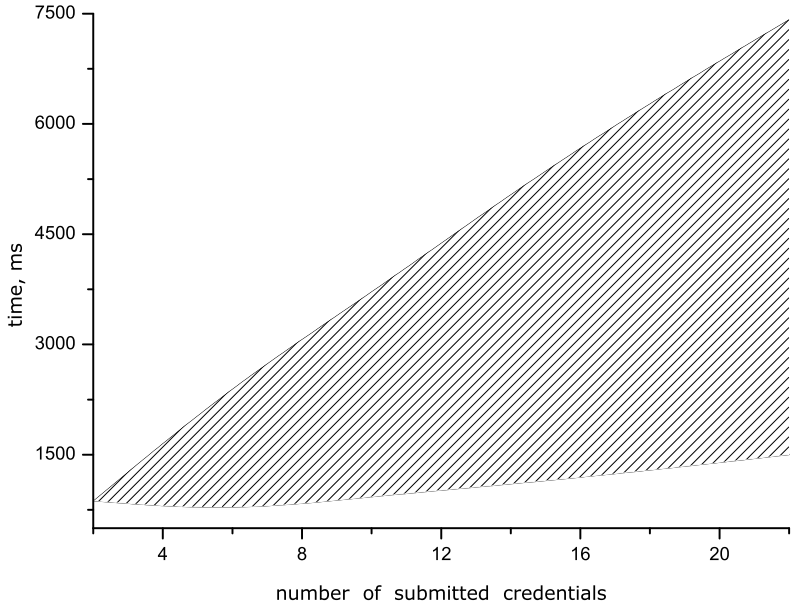

Figure 6.5: Time Bounds for Trust Negotiation

with the opponent or domain/environment, type of resources being negotiated upon etc.

The proposed negotiation protocol serves as a policy enforcement engine over the access and disclosure policies of an opponent, i.e. the protocol is data-driven by the deduction and abduction reasoning. In that sense, on top of the missing credentials computed one can additionally impose a strategy controlling the disclosure of these credentials. Here we note that such a strategy could be encoded directly in the disclosure policy (in [61] the authors define a stepwise reasoning on the disclosure policy structure) so that the protocol enforces the strategy directly. We also note that multiple disclosure policies can be defined for a given access policy thus encoding different possible strategies.

Since strategies depend on multiple factors and may take different sequences of credential exchange, so the goal we approach is to analyze the time boundaries the negotiation protocol performs by abstracting from a specific strategy. Essentially, we wanted to examine the protocol behavior on two extreme negotiation modes with the possibility of concurrent credential negotiations.

Figure 6.5 shows the time area resulted from our experimentations where all negotiation strategies, our prototype can perform, fall into.

The upper time bound, named as *mutual suspicious mode*, is implied by a stepwise disclosure of credentials where each entity discloses missing credentials consecutively one after the other. Thus, one credential by the server side and one by the

client side in response, where the next credential is requested (negotiated) if the previous negotiation has been completed. The negotiation process is run in a single system thread on a client side and on a server side without any concurrent requests.

We assume the following negotiation scenario of this mode. A client and a server have the same number of 11 credentials protecting sensitive resources. We chose 11 credentials to obtain a reasonable abduction computation close to the estimated threshold noted above. The client requests for a resource with no input of credentials. The server computes number N of missing credentials ($N \leq 11$) and negotiates on them in suspicious mode. For each of the missing credential the client has a counter request with a credential to the server side. The server grants the requested credential and the client, on its turn, grants the respective credential to the server.

In this way, for each of the server's credential we model a negotiation round where the client runs abduction reasoning to find a missing credential and successfully negotiates on it. The negotiation scenario has $N + 1$ invocation of abduction reasoning over fixed number of 11 hypotheses in any test.

The opposite to suspicious mode, is the *mutual greedy mode* of negotiations. This mode results in the bottom time bound in the figure. The minimum time for negotiation is determined by the strategy that implies complete disclosure of missing credentials in one round by client and by server side. Greedy mode utilizes multi-threaded concurrent exchange of missing credentials on each side. To obtain the minimum negotiation time, we modify the above scenario with the assumption that the client and the server have recently been in contact for that resource and, being in greedy mode, the client along with the request for the resource runs N requests for the credentials necessary to grant server's N missing credentials. On receiving request for a resource, the server runs N threads requesting the missing credentials. In this scenario, we avoid client running abduction reasoning but only deduction to check if server's requests are granted by the already run client's requests. In the greedy mode, the client and server run $2N$ system threads, while in the suspicious mode they run only 2 system threads. In greedy mode we loose in memory but profit in execution time.

Figure 6.5 illustrates how with increasing $1 \leq N \leq 11$ the two extreme modes perform in time. All possible negotiations lie in the hatched area. On x-axis we show the total number of credentials necessary to complete a negotiation process, where half of the credentials are requested by the server and the other half by the client side. With total of 22 credentials (the server and client ask each other for 11 credentials) any negotiation strategy would remain within 1.5 to 7.5 seconds. With 8 credentials the overall negotiation time is bound by 0.9 and 3.7 seconds.

In the following we briefly relate our results with those reported for the Traust service scenario [71, pp 14–16]. In the Traust scenario there are two subsequent negotiation sessions: one to disclose the request to access an information portal and one to access the portal as a rescue dog handler. The authors report an average execution time of the whole scenario to 4.04 seconds with total of nine disclosed

credentials. In our case, the scenario in Figure 4.5(a) approximates the interactions of the first negotiation of the Traust scenario, while the mutual suspicious mode with total of 8 credentials approximates (upper bound) the message exchanges of the second negotiation phase. Our average time (sum of the two negotiations) to simulate the scenario would be an average of 4 seconds. Although, we experimented with slightly faster CPUs the obtained results shown that our system performs comparably to that of Traust.

6.1.3 Integration of Usage Control in Globus

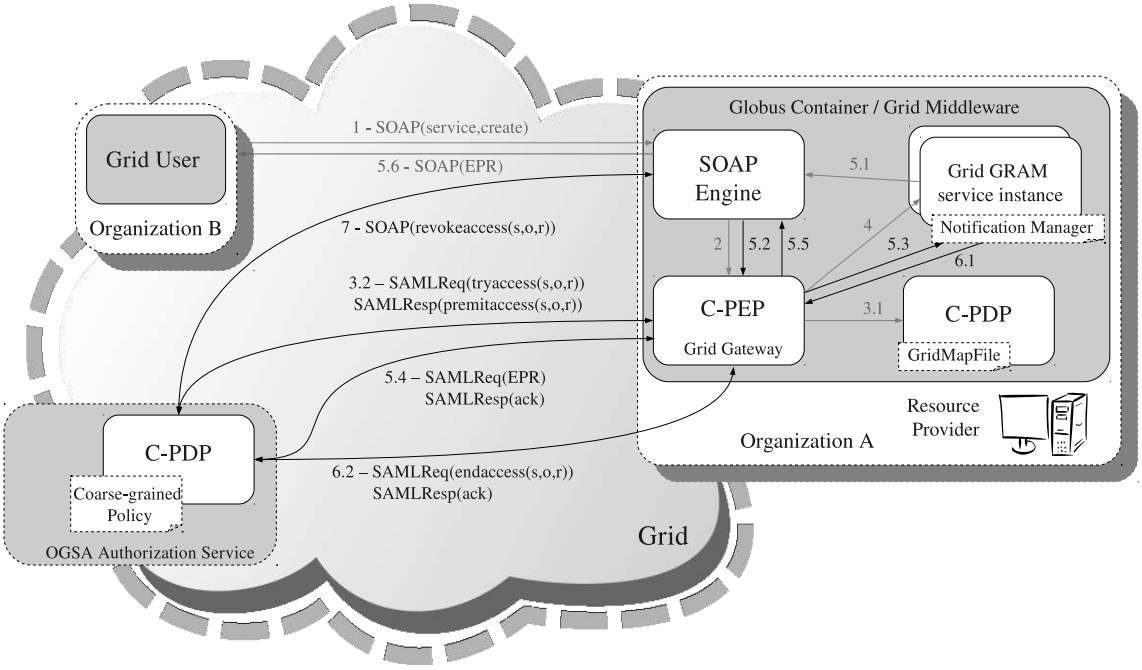

Figure 6.6: Usage Control in Globus Toolkit

UCON novelties require the implementation of the state-full coarse-grained reference monitor what is not supported by Globus Toolkit. Standard authorization services for Globus are just limited to grant access to services without taking into account the specific instance of them. Our approach proposes the state-full coarse-grained reference monitor which is able to manage the interactions also in terms of access to specific service instances, monitor continuously the execution of the methods after the access has been granted, and destroy running service instances which violate prescribed security policies.

Our solution entails some modifications in Globus Container and assumes many-rounds interactions between C-PEP and C-PDP during the security policy enforcement. C-PDP is modeled as remote authorization service through SAMLAuthzCallout. Modified Globus Container processes our access and usage control scenarios as

follows (see Figure 6.6):

- *Steps 1-4*: Correspond accordingly to the enforcement of default access control scenarios by Globus Container (rf. section 6.1.1 and Figure 6.1). The only difference is the semantics and processing of the SAML request and responde received and sent by C-PDP to C-PEP in the step 3.2. When C-PDP receives a new SAML request, it behaves as described in section 5.2. But instead of creating the complete security automaton that models the security policy, C-PDP builds only part of it. This part models access control scenarios, and last action accepted by the security automaton is the $permitaccess(s, o, r)$. When all policy statements corresponding to access control are enforced, C-PDP replies to C-PEP with the appropriate SAML response message. Assume, that access decision to create a new Grid service instance was positive, and C-PEP created the new instance;

- *Step 5*: SOAP Engine handles the end point reference (EPR) of the new instance (step 5.1). Originally, EPR is immediately sent back to the Grid user. Instead, we changed this mechanism to make additional procedures.

  In step 5.2, SOAP Engine sends EPR to C-PEP.

  In step 5.3, C-PEP contacts Notification Manager and subscribes to the notification mechanisms. Notification Manager is the core functionality of Grid services realized in Globus. It implements the WS-Notification specification and operates by monitoring continuously the service instance life-cycle. Any changes occurred with the service instance are automatically forwarded by Notification Manager to its subscribers. Thus, C-PEP becomes aware about the behavior of the service instance.

  In step 5.4, C-PEP creates a new SAML request message addressed to C-PDP. This request contains EPR and specifies that *continuous usage control* for the created service instance is required. The initial access request is also attached to this message. C-PDP receives this SAML request, and creates the last part of the security automaton that models the security policy. This part corresponds to pure usage control scenarios and starts with the $permitaccess(s, o, r)$. Then, C-PDP starts the enforcement of the security policy and replies to C-PEP with the acknowledge SAML response message.

  In step 5.5, C-PEP replies to SOAP engine that all steps regarding usage control are accomplished and EPR can be sent back to the Grid user (step 5.6). The phase of continuous control of the Grid service instance starts. From then, the security policy enforcement depends on what happens first: (step 6) the Grid service instance completes its tasks or the Grid user decides to stop the service; (step 7) C-PDP detects the security policy violation;

- *Step 6*: If the Grid service instance terminates normally, Notification Manager informs C-PEP about this event (step 6.1). In step 6.2, C-PEP creates

a new SAML request message addressed to C-PDP. This request contains EPR and specifies that the service instance with this EPR does not exist anymore. C-PDP receives this SAML request, and processes the enforcement of the $endaccess(s, o, r)$ action. When the enforcement of the security policy is completed, C-PDP replies to C-PEP with the acknowledge SAML response message. C-PDP and C-PEP delete all data relating to the enforced policy and this step ends the usage control enforcement;

- *Step 7*: Instead, if C-PDP detects the security policy violation, it sends to Globus Container the SOAP request to destroy the service instance with the specified EPR. Globus Container handles this request as usual, and forwards to C-PEP for the access decision evaluation. C-PEP recognizes that the massage came from C-PDP, allows its execution, destroys the services instance. C-PDP and C-PEP delete all data relating to the enforced policy and this step ends the usage control enforcement.

If C-PDP is the remote authorization service, i.e. it is hosted by different resource provider somewhere in Grid, the security overhead posed by integration of the state-full coarse-grained reference monitor is negligible. The system performance is slowed down only by many-rounds interactions between C-PEP and C-PDP. All computations regarding the evaluation of coarse-grained authorization and condition predicates, obligation and update actions execution are placed outside the machine which hosts requested Grid service instances.

6.2 Implementation of Fine-grained Reference Monitor

This section presents the implementation of the fine-grained reference monitor presented in Figure 5.3. We integrated some components of our model within the Globus GRAM service.

6.2.1 Monitoring of Java Jobs

First, we extended the standard Globus job description schema to define a new job type, the `java` one. In this way, to execute a Java application the Grid user specifies `java` as a job type in its job request. Second, we defined an alternative scheduling system that in case of a Java application executes it on our customized Java Virtual Machine (JVM), and we configured the Managed Job Service (MJS) component of the Globus GRAM service to invoke this scheduler instead of the standard one. The standard error stream is used in order to return to Grid users error messages when an application has been stopped because of a fine-grained security policy violation. The standard Globus mechanism to transfer files, i.e. GridFTP, is exploited to send to the remote Grid user the log file with the error description.

F-PDP interacts with the GRAM service in order to get the job request submitted by the remote Grid user. This interaction is simply implemented through an XML file that is generated by the GRAM service before invoking the security enhanced JVM. The file name is passed to JVM which serves as F-PEP as an input parameter and JVM, in turn, passes it to F-PDP. F-PDP reads from this file the resource requirements in the job request which will be enforced during execution. F-PDP also imports from Globus the user DN extracted from the proxy certificate that has been submitted by the Grid user.

F-PDP is activated by JVM every time a Java application wants to perform an interaction with the underlying resource. This has been implemented by modifying the implementation of the Java core classes, i.e. the classes that manage the interactions with the underlying resource. In particular, these classes have been modified by substituting the invocations to the system libraries functions which perform the security relevant system calls with invocation to a proper set of wrapper functions we defined. The execution of external code through the Java Native Interface (JNI) [75] has been disabled, since it allows interactions with underlying resource through external libraries (e.g., non java-based ones). A wrapper function first activates F-PDP to perform the security checks before the execution of the security relevant system call and suspends itself (i.e. suspends the JVM). Then, after F-PDP reactivates the wrapper function, the last enforces F-PDP access decision by either actually invoking the system library function or by interrupting the application execution. When the system library function has been executed the wrapper function activates F-PDP again to perform security checks after the system call execution. The interactions between F-PDP and JVM are implemented by using semaphores and shared variables.

F-PDP invokes Authorization Manager sometimes during the job execution to evaluate attributes of the Grid user submitted during the coarse-grained control. Authorization Manager checks these attributes using Property Policy (rf. Figure 5.1) and we name such functionality as Property PDP.

F-PDP calls Property PDP as an external library function. The interactions between the two components have a critical impact on the monitoring performance. Property PDP has been implemented in Java but F-PDP - in C, thus we invoke Property PDP without loading JVM every time. We use the Java Native Interface (JNI) library that allows us to invoke Java methods from inside a C code. As a part of the functionality of this library, we have a special command that explicitly loads JVM when needed for a Java class execution, and also commands to obtain links to a Java class and its methods (in our case the main class which implement Property PDP). In this way, at initialization time, F-PDP loads JVM, creates links to Property PDP methods, and keeps them in memory for all subsequent invocations. When the Java application being monitored terminates, F-PDP releases JVM and then terminates too.

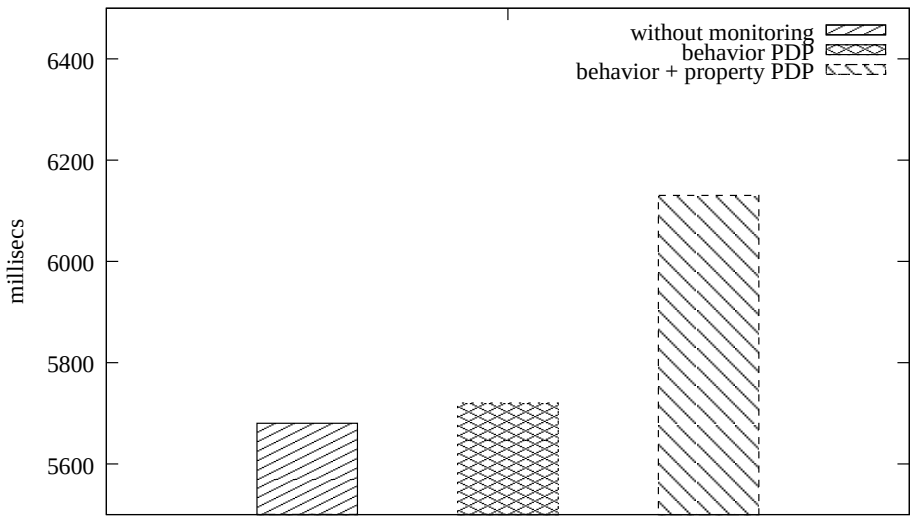

Figure 6.7: Fine-grained Monitoring Performance

6.2.2 Performance Evaluation

The security checks performed by the fine-grained monitoring introduce an overhead in the application execution time. The main factor concerning the execution overhead is if a java application performs more computational operations with only few system calls – case where the overhead of security monitoring is negligible with respect to the overall execution time, or if an application mainly performs system calls – case where security checks impacts on the overall time.

The test presented here evaluates the execution time of a Java application by adopting a standard JVM and a security enhanced JVM with Property PDP and F-PDP. In particular, we adopted the Jikes RVM Java Virtual Machine developed by the IBM T. J. Watson Research Center [13] run on Linux operating system. This JVM has been modified to embed our security support.

The test scenario enforces the fine-grained security policy given in subsection 3.4.4, where the Java application submitted by the Grid user exploits an utility library. In this case, the utility library consists of a collection of benchmarks that belong to the Ashes Suite Collection benchmarks[3]. In particular, user's remote application uses a library to convert an mp3 file to a wav one. Both, the Grid user's application and the utility library, were monitored by our system. System calls defined are: read the library, read an input file (in mp3 format), and write a result file (in wav format). This application is well-suited for our test because it performs a large number of security relevant system calls during its execution: around 1500 calls for about 5 seconds.

The fine-grained system controls if a user submitted the application has a proper

[3]http://www.sable.mcgill.ca/ashes

set of credentials to open the requested utility library, and if access to the library and to the data files is according to the admitted behavior.

Figure 6.7 shows the results of the evaluation as an average of ten trials. The average execution time of the library without any security control is 5680 milliseconds, while the execution time with the enforcement of the fine-grained policy only is 5720 milliseconds, and the execution time with the enforcement of both the fine-grained and the user property policies is 6130 milliseconds. F-PDP has been invoked 1489 times, while the Property PDP has been invoked only 1 time - when the application opened the utility library file. The overhead introduced by F-PDP is about 1%, while the overhead by both Property PDP and F-PDP is about 8% of the application execution time. The results shown the practical aspects of F-PDP, while Property PDP shown that the system overhead is sensitive against logic access decisions at the fine-grained level of control.

On the fine-grained level of control, all security threats (e.g. to subvert system functionality and block up computational resources) come from Java applications executed by the Globus GRAM computational service. An application executed on the platform of the Grid resource provider can not perform dangerous or forbidden operations on platform's resources, since every action performed by the application is intercepted and checked before the actual execution. Moreover, the application cannot bypass the monitoring mechanism, because the fine-grained reference monitor was integrated inside JVM, and the execution of the application is completely mediated by JVM. Additionally, we disabled the Java Native Interface support for executing arbitrary code from a Java application. We also emphasize on the importance of trust between policy decision and enforcement components on the fine-grained level.

Chapter 7

Concluding Remarks

7.1 Summary

The advance of Grid computing technology promises to have far-reaching effects on resource-intensive engineering and scientific applications.

This thesis discusses access and usage control framework to protect computational Grids on three levels of abstraction: policy, enforcement and implementation. The proposed framework is based on UCON model which main novelties are continuity of control and attributes mutability [70, 32].

The policy level [64, 63, 58, 65, 36, 33] outlines a basic system abstraction for coarse-grained and fine-grained control.

Coarse-grained objects are Grid services instances, and Grid service workflows. Coarse-grained subjects are Grid users attempting to execute jobs in Grid by means of its computational services. Access rights on the coarse-grained level of control are portTypes of Grid services. A Grid service instance is a long-lived object and access to this object is continuously evaluated. If in meanwhile attributes change values in a such way that the security policy is not satisfied, the access is revoked and the Grid service instance is destroyed.

Fine-grained security policy controls behavior of submitted jobs. Objects here are low-level computational resources like files, sockets, CPU, and access rights specify operations over these resources. Fine-grained level of control monitors jobs execution and terminates those execution which violates security policies.

Coarse- and fine-grained levels of control share attributes used to make access decision an revocation of access on any levels leads to the revocation of the last level.

We classify attributes on immutable, attributes with enforceable and observable mutuality. Also, attributes may be remote and local regarding their provenance. Access and usage control scenarios enforcement varies according to attributes used to construct security policies. U-XACML policy specification language is introduces to express basic access and usage scenarios. It enhances XACML policy languages with UCON novelties - attribute mutability and continuity of control. POLPA language

based on process algebra is exploited to express more complex usage scenarios, e.g. to model allowed Grid service workflows on coarse-grained level and to express allowed execution traces of submitted jobs on fine-grained level of control.

Further, the policy level of our framework introduce some enhancements, trust negotiations and risk-aware access and usage control model.

Trust negotiations are applied on coarse-grained level of control and capture access control scenarios based on remote attributes. Additional credential and disclosure policies are introduced to negotiate on the attributes needed to evaluate authorization predicates.

A risk-aware usage control model in presented on coarse-grained level of control and capture access and usage control scenarios based on remote attributes with observable mutability. Attributes observed by the reference monitor might not correspond to real attributes values due to several reasons. Thus, the accesses decision is produced taking into account some uncertainties associated with attributes. These uncertainties can be intentional and unintentional. To produce access decision, the reference monitor computes these uncertainties and weights risks to grant or deny access. Based on the cost matrix, the reference monitor takes the less risky decision. In usage control scenarios, the uncertainties grow with the time passed since the last attribute value was received and the security policy evaluated. When these uncertainties are too big and the policy enforcement becomes to risky, the reference monitor pulls new attribute values. The risk-aware access and usage control model is used to make the efficient scheduler for attributes acquisition in usage control scenarios.

The enforcement level [30, 31, 33, 58] outlines the architecture of the reference monitor on coarse- and fine-grained levels of control. The interactions between main components of the reference monitor enforcing access and usage control scenarios are state-full.

The implementation level [30, 31, 58] proposes the integrations of our access and usage control model in Globus Toolkit. Trust negotiations are implemented on the coarse-grained level and show reasonable security overhead. Coarse-grained level of control is realized to protect any WSRF-compliant Grid service. Fine-grained level of control is realized for the GRAM computational service and protects resources allocated for the execution of Java jobs submitted by Grid users. The integration of our security support entails small modification in the Globus container and the GRAM service.

7.2 Future Work

A future work concerns the improvements on every level of access and usage control model. The policy level should be enhanced with the unified policy language capable to capture different access and usage control scenarios. The risk-aware usage control model should research on mining initial probabilities and cost matrix values. The

enforcement level should introduce a reference monitor with distributed components, e.g. a single PDP might control only a part of Grid services and the security policy over all Grid services should be enforced by providing a model of collaborations among several PDPs which enforce only the part of the policy. The implementation level should address the implementation of obligations and conditions managers and attribute updates in the presence on concurrent usage sessions.

Besides, as a future work we explore other areas of UCON model applicability, e.g. in networking and in cloud computing.

Bibliography

[1] Akenti. http://dsd.lbl.gov/security/Akenti/.

[2] The globus alliance. http://www.globus.org.

[3] Gridshib project. http://grid.ncsa.uiuc.edu/GridShib.

[4] Permis. http://sec.cs.kent.ac.uk/permis/index.shtml.

[5] Shibboleth project. http://shibboleth.internet2.edu/.

[6] Datagrid security design. Deliverable 7.6 DataGrid Project, 2003.

[7] M. Abadi. Logic in access control. In *LICS '03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 228, Washington, DC, USA, 2003. IEEE Computer Society.

[8] M. Alam, J.-P. Seifert, Q. Li, and X. Zhang. Usage control platformization via trustworthy SELinux. In *ASIACCS '08: Proceedings of ACM symposium on Information, computer and communications security*, pages 245–248, New York, NY, USA, 2008. ACM.

[9] M. Alam, X. Zhang, M. Nauman, T. Ali, and J.-P. Seifert. Model-based behavioral attestation. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 175–184, New York, NY, USA, 2008. ACM.

[10] C. J. Alberts and A. J. Dorofee. Octave criteria, version 2.0. Technical report, CMU/SEI-2001-TR-016, December 2001.

[11] R. Aleliunas, R. M. Karp, R. J. Lipton, L. Lovasz, and C. Rackoff. Random walks, universal traversal sequences, and the complexity of maze problems. In *SFCS '79: Proceedings of the 20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 218–223, Washington, DC, USA, 1979. IEEE Computer Society.

[12] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS: An authorisation system for virtual organizations. In *Proceedings of 1st European Across Grid Conference*, 2003.

[13] B. Alpern, C. Attanasio, J. Barton, et al. The jalapeño virtual machine. *IBM System Journal*, 39(1):211–221, 2000.

[14] K. Apt. Logic programming. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*. Elsevier, 1990.

[15] B. Aziz, S. N. Foley, J. Herbert, and G. Swart. Reconfiguring role based access control policies using risk semantics. *J. High Speed Networks*, 15(3):261–273, 2006.

[16] F. Baiardi, F. Martinelli, P. Mori, and A. Vaccarelli. Improving grid services security with fine grain policies. In *OTM Workshops*, Lecture Notes in Computer Science, pages 123–134. Springer, 2004.

[17] M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. *International Journal of Software: Practice and Experience*, 32(15):1437–1466, 2002.

[18] M. Bartoletti. *Usage Automata*, pages 52–69. Springer-Verlag, Berlin, Heidelberg, 2009.

[19] S. Baselice, P. A. Bonatti, and M. Faella. On interoperable trust negotiation strategies. In *Proceedings of IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, pages 39–50. IEEE Computer Society, June 2007.

[20] M. Y. Becker and S. Nanz. The role of abduction in declarative authorization policies. In *Proceedings of the 10th International Symposium on Practical Aspects of Declarative Languages (PADL'08)*, LNCS. Springer, 2008.

[21] A. Berthold, M. Alam, R. Breu, M. Hafner, A. Pretschner, J.-P. Seifert, and X. Zhang. A technical architecture for enforcing usage control requirements in service-oriented architectures. In *SWS '07: Proceedings of ACM workshop on Secure web services*, pages 18–25, New York, NY, USA, 2007. ACM.

[22] E. Bertino, P. Bonatti, and E. Ferrari. TrBAC: A temporal role-based access control model. *ACM Trans. Inf. Syst. Secur.*, 4(3):191–233, 2001.

[23] C. Bettini, S. Jajodia, X. Wang, and D. Wijesekera. Obligation monitoring in policy management. In *POLICY '02: Proceedings of the 3rd International Workshop on Policies for Distributed Systems and Networks*, Washington, DC, USA, 2002. IEEE Computer Society.

[24] M. Bouzeghoub and V. Peralta. A framework for analysis of data freshness. In *Proceedings of the International Workshop on Information Quality in Information Systems*, pages 59–67, 2004.

[25] J. Bryans. Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35, New York, NY, USA, 2005. ACM.

[26] D. Chadwick, A. Novikov, and A. Otenko. Gridshib and permis integration. http://www.terena.org/events/tnc2006/programme/presentations/show.php?pres_id=200.

[27] D. Chadwick and A. Otenko. The PERMIS x.509 role based privilege management infrastructure. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 135–140, New York, NY, USA, 2002. ACM Press.

[28] D. Chadwick, G. Zhao, S. Otenko, R. Laborde, L. Su, and T. A. Nguyen. PERMIS: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*, 20(11):1341–1357, August 2008. Online ISSN: 1532-0634.

[29] S. J. Chapin, D. Katramatos, J. Karpovich, and A. Grimshaw. Resource management in Legion. *Future Generation Computer Systems*, 15(5–6):583–594, 1999.

[30] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. Controlling the usage of grid services. *International Journal of Computational Science*, 3(4):373–387, 2009.

[31] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. On usage control for grid services. In *proceedings of International Joint Conference on Computational Sciences and Optimization: CSO'09*, pages 47–51, Washington, DC, USA, 2009. IEEE Computer Society.

[32] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. *Handbook of Information and Communication Security*, chapter Access and Usage Control in Grid Systems, pages 293–308. Springer, 2010.

[33] M. Colombo, A. Lazouski, F. Martinelli, and P. Mori. A proposal on enhancing XACML with continuous usage control features. In *proceedings of CoreGRID ERCIM Working Group Workshop on Grids, P2P and Services Computing*, pages 133–146. Springer US, 2010.

[34] N. N. Diep, L. X. Hung, Y. Zhung, S. Lee, Y.-K. Lee, and H. Lee. Enforcing access control using risk assessment. *European Conference on Universal Multiservice Networks*, 0:419–424, 2007.

[35] T. Eiter, G. Gottlob, and N. Leone. Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, 189(1–2):129–177, 1997.

[36] D. Fais, M. Colombo, and A. Lazouski. An implementation of role-base trust management extended with weights on mobile devices. *Electron. Notes Theor. Comput. Sci.*, 244:53–65, August 2009.

[37] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, MasanobuTakagi, R. Bing, A. Amir, R. Murakawa, O. Hernandez, J. Magowan, and N. Bieberstein. *Introduction to grid computing with globus.* IBM Corp., Riverton, NJ, USA, 2003.

[38] I. Foster. Globus toolkit version 4: Software for service-oriented systems. In *Proceedings of IFIP International Conference on Network and Parallel Computing.* Springer-Verlag, LNCS, 2005.

[39] I. Foster and C. Kesselman. Computational grids. In *In VECPAR*, pages 3–37. Morgan Kaufmann, 1986.

[40] I. Foster and C. Kesselman. The globus project: A status report. In *Proceedings of IPPS/SPDP '98 Heterogeneous Computing Workshop*, pages 4–18, 1998.

[41] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The physiology of the grid: An open grid service architecture for distributed system integration. Globus Project, 2002. http://www.globus.org/research/papers/ogsa.pdf.

[42] I. Foster, C. Kesselman, L. Pearlman, S. Tuecke, and V. Welch. A community authorization service for group collaboration. In *Proceedings of the3rd IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY'02)*, pages 50–59, 2002.

[43] P. Gama and P. Ferreira. Obligation policies: An enforcement platform. In *POLICY '05: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 203–212, Washington, DC, USA, 2005. IEEE Computer Society.

[44] M. Hafner, M. Memon, and M. Alam. Modeling and enforcing advanced access control policies in healthcare systems with sectet. In *Models in Software Engineering*, pages 132–144. Springer Berlin/Heidelberg, 2008.

[45] Y. Han, Y. Hori, and K. Sakurai. Security policy pre-evaluation towards risk analysis. In *ISA '08: Proceedings of the 2008 International Conference on Information Security and Assurance (isa 2008)*, pages 415–420, Washington, DC, USA, 2008. IEEE Computer Society.

[46] S. O. Hanson. Decision theory: A brief introduction, August 1994.

[47] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Commun. ACM*, 19(8):461–471, 1976.

[48] M. Hilty, D. Basin, and A. Pretschner. On obligations. In *Proceedings of ESORICS 2005*, pages 98–117, 2005.

[49] M. Hilty, A. Pretschner, C. Schaefer, and T. Walter. Usage control requirements in mobile and ubiquitous computing applications. In *ICSNC '06: Proceedings of the International Conference on Systems and Networks Communication*, Washington, DC, USA, 2006. IEEE Computer Society.

[50] O. C. Ibe. *Fundamentals of Applied Probability and Random Processes*. Elsevier Academic Press, 2005.

[51] K. Irwin, T. Yu, and W. H. Winsborough. On the modeling and analysis of obligations. In *CCS 06: Proceedings of the 13th ACM conference on Computer and communications security*, pages 134–143. ACM Press, 2006.

[52] R. Jagadeesan, W. Marrero, C. Pitcher, and V. Saraswat. Timed constraint programming: A declarative approach to usage control. In *PPDP '05: Proceedings of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 164–175, New York, NY, USA, 2005. ACM.

[53] H. Janicke, A. Cau, and H. Zedan. A notes on the formalisation of UCON. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 163–168, New York, NY, USA, 2007. ACM.

[54] A. Kapadia, G. Sampemane, and R. H. Campbell. KNOW why your access was denied: regulating feedback for usable security. In *Proceedings of the 11th ACM conference on Computer and Communications Security*, pages 52–61, New York, NY, USA, 2004. ACM Press.

[55] B. Katt, X. Zhang, R. Breu, M. Hafner, and J.-P. Seifert. A general obligation model and continuity: Enhanced policy enforcement engine for usage control. In *SACMAT '08: Proceedings of the 13th ACM symposium on Access control models and technologies*, pages 123–132, New York, NY, USA, 2008. ACM.

[56] K. Keahey and V. Welch. Fine-grain authorization for resource management in the grid environment. In *GRID '02: Proc. of the Third International Workshop on Grid Computing - LNCS*, volume 2536, pages 199–206, 2002.

[57] V. Kolovski, J. Hendler, and B. Parsia. Analyzing web access control policies. pages 677–686, 2007.

[58] H. Koshutanski, A. Lazouski, F. Martinelli, and P. Mori. Enhancing grid security by fine-grained behavioral control and negotiation-based authorization. *International Journal of Information Security*, 8(4):291–314, 2009.

[59] H. Koshutanski, F. Martinelli, P. Mori, and A. Vaccarelli. Fine-grained and history-based access control with trust management for autonomic grid services. *Proc. of International Conference on Autonomic and Autonomous Systems*, 2006.

[60] H. Koshutanski and F. Massacci. Interactive access control for Web Services. In *Proceedings of the 19th IFIP Information Security Conference (SEC 2004)*, pages 151–166, Toulouse, France, August 2004. Kluwer Press.

[61] H. Koshutanski and F. Massacci. A negotiation scheme for access rights establishment in autonomic communication. *Journal of Network and System Management (JNSM)*, 15(1), 2007.

[62] H. Koshutanski and F. Massacci. Interactive access control for autonomic systems: from theory to implementation. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 2009.

[63] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin. Influence of attribute freshness on decision making in usage control. To appear in proceedings of 6th International Workshop on Security and Trust Management: STM'10, 2010.

[64] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin. Risk-aware usage decision making in highly dynamic systems. *International Conference on Internet Monitoring and Protection: ICIMP'10*, pages 29–34, 2010.

[65] L. Krautsevich, A. Lazouski, F. Martinelli, and A. Yautsiukhin. Risk-based usage control for service oriented architecture. In *proccedings of 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing: PDP'10*, pages 641–648. IEEE Computer Society, 2010.

[66] R. Krishnan, R. Sandhu, and K. Ranganathan. Pei models towards scalable, usable and high-assurance information sharing. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 145–150, New York, NY, USA, 2007. ACM.

[67] W. Ku and C.-H. Chi. Survey on the technological aspects of digital rights management. In *Information Security*, pages 391–403, 2004.

[68] D. Kyle and J. C. Brustoloni. Uclinux: A linux security module for trusted-computing-based usage controls enforcement. In *STC '07: Proceedings of ACM workshop on Scalable trusted computing*, pages 63–70, New York, NY, USA, 2007. ACM.

[69] L. Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[70] A. Lazouski, F. Martinelli, and P. Mori. Usage control in computer security: A survey. *Computer Science Review*, 4(2):81–99, 2010.

[71] A. J. Lee, M. Winslett, J. Basney, and V. Welch. The traust authorization service. *ACM Transactions on Information and System Security (TISSEC)*, 11(1):1–33, 2008.

[72] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, and F. Scarcello. The dlv system for knowledge representation and reasoning. *ACM Transactions on Computational Logic*, 2006. Available via url http://www.arxiv.org/ps/cs.AI/0211004.

[73] R. Lepro. Cardea: Dynamic access control in distributed systems. Technical Report NAS Technical Report NAS-03-020, NASA Advanced Supercomputing (NAS) Division, 2003.

[74] Y. Li, H. Sun, Z. Chen, J. Ren, and H. Luo. Using trust and risk in access control for grid environment. *International Conference on Security Technology*, 0:13–16, 2008.

[75] S. Liang. *Java(TM) Native Interface: Programmer's Guide and Specification*. Addison-Wesley, 1999.

[76] J. Ligatti, L. Bauer, and D. Walker. Run-time enforcement of nonsafety policies. *ACM Trans. Inf. Syst. Secur.*, 12:19:1–19:41, January 2009.

[77] S. Lili and L. Yan. XML schema in XML documents with usage control. *International Journal of Computer Science and Network Security*, 2007.

[78] M. Lorch, D. B. Adams, D. Kafura, M. S. R. Koneni, A. Rathi, and S. Shah. The prima system for privilege management, authorization and enforcement in grid environments. In *GRID '03: Proceedings of the 4th International Workshop on Grid Computing*, page 109, Washington, DC, USA, 2003. IEEE Computer Society.

[79] L. C. Lung, M. S. Higashiyama, R. R. Obelheiro, and J. da Silva Fraga. Adapting the UCON Usage Control Policies on CORBASec Infrastructure. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 483–488, Washington, DC, USA, 2007. IEEE Computer Society.

[80] F. Martinell and I. Matteucci. Through modeling to synthesis of security automata. *Electron. Notes Theor. Comput. Sci.*, 179:31–46, July 2007.

[81] F. Martinelli and P. Mori. A Model for Usage Control in GRID systems. In *Proceedings of GRID-STP*. IEEE Press, 2007.

[82] F. Martinelli, P. Mori, and A. Vaccarelli. Towards continuous usage control on grid computational services. In *ICAS-ICNS '05: Proceedings of the Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services*, Washington, DC, USA, 2005. IEEE Computer Society.

[83] P. Massonet, A. Arenas, F. Martinelli, P. Mori, and B. Crispo. Usage control for trust and security in next generation grids. In *At Your Service: Service Engineering in the Information Society Technologies Program*. MIT Press, 2008.

[84] Y. Mei, X. Dong, W. Wu, S. Guan, and J. Xu. UCGS: A Usage Control Approach for Grid Services. In *CISW '07: Proceedings of International Conference on Computational Intelligence and Security Workshops*, pages 486–489, Washington, DC, USA, 2007. IEEE Computer Society.

[85] N. H. Minsky and A. D. Lockman. Ensuring integrity by adding obligations to privileges. In *ICSE '85: Proceedings of the 8th international conference on Software engineering*, pages 92–102, Los Alamitos, CA, USA, 1985. IEEE Computer Society Press.

[86] R. Motwani and P. Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996.

[87] M. Nauman, M. Alam, X. Zhang, and T. Ali. Remote attestation of attribute updates and information flows in a ucon system. In *Proceedings of the Second International Conference on Trust Computing.*, volume 5471 of *Lecture Notes in Computer Science*, pages 63–80. Springer-Verlag, 2009.

[88] V. Nefedova, R. Jacob, I. Foster, Z. Liu, Y. Liu, E. Deelman, G. Mehta, M.-H. Su, and K. Vahi. Automating climate science: Large ensemble simulations on the teragrid with the griphyn virtual data system. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 32, Washington, DC, USA, 2006. IEEE Computer Society.

[89] S. Osborn, R. Sandhu, and Q. Munawer. Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Trans. Inf. Syst. Secur.*, 3(2):85–106, 2000.

[90] J. Park. *Usage control: A unified framework for next generation access control.* PhD thesis, George Mason University, Fairfax, VA, USA, 2003.

[91] J. Park and R. Sandhu. Towards usage control models: Beyond traditional access control. In *SACMAT '02: Proceedings of the seventh ACM symposium on Access control models and technologies*, pages 57–64, New York, NY, USA, 2002. ACM.

[92] J. Park and R. Sandhu. The UCON ABC usage control model. *ACM Trans. Inf. Syst. Secur.*, 7(1):128–174, 2004.

[93] J. Park, X. Zhang, and R. S. Sandhu. Attribute mutability in usage control. In *DBSec*, pages 15–29. Kluwer, 2004.

[94] L. Pearlman, C. Kesselman, V. Welch, I. Foster, and S. Tuecke. The community authorization service: Status and future. *Proceedings of Computing in High Energy and Nuclear Physics (CHEP03): ECONF*, C0303241:TUBT003, 2003.

[95] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Commun. ACM*, 49(9):39–44, 2006.

[96] A. Pretschner, M. Hilty, D. Basin, C. Schaefer, and T. Walter. Mechanisms for usage control. In *ASIACCS '08: Proceedings of ACM symposium on Information, computer and communications security*, pages 240–244, New York, NY, USA, 2008. ACM.

[97] A. Pretschner, F. Schuetz, C. Schaefer, and T. Walter. Policy evolution in distributed usage control. STM 08: Proceedings of 4th International Workshop on Security and Trust Management, 2008. to be published.

[98] A. Pretschner and T. Walter. Negotiation of usage control policies - simply the best? In *ARES '08: Proceedings of Third International Conference on Availability, Reliability and Security*, pages 1135–1136, Washington, DC, USA, 2008. IEEE Computer Society.

[99] J. H. Saltzer and M. D. Schroeder. The protection of information in computer systems, 1975.

[100] P. Samarati and S. D. C. di Vimercati. Access control: Policies, models, and mechanisms. In *FOSAD'00: International School on Foundations of Security Analysis and Design*, pages 137–196, London, UK, 2001. Springer-Verlag.

[101] R. S. Sandhu and J. Park. Usage control: A vision for next generation access control. In *MMM-ACNS*, volume 2776 of *Lecture Notess in Computer Science*, pages 17–31. Springer, 2003.

[102] M. Sastry, R. Krishnan, and R. Sandhu. A new modeling paradigm for dynamic authorization in multi-domain systems. In *Communications in Computer and Information Science*, volume 1, pages 153–158. Springer Berlin Heidelberg, 2007.

[103] F. B. Schneider. Enforceable security policies. *ACM Trans. Inf. Syst. Secur.*, 3:30–50, February 2000.

[104] F. B. Schneider, K. Walsh, and E. G. Sirer. Nexus authorization logic (nal): Design rationale and applications. Technical report, Cornell Computing and Information Science Technical Report, 2009.

[105] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proceedings of IJCAI'89*, pages 1055–1060. Morgan Kaufmann, 1989.

[106] C. Skalka, X. S. Wang, and P. Chapin. Risk management for distributed authorization. *J. Comput. Secur.*, 15(4):447–489, 2007.

[107] B. Spencer. Neesgrid: A distributed collaboratory for advanced earthquake engineering experiment and simulation. In *13th World Conf. on Earthquake Engineering*, 2004.

[108] A. Squicciarini, E. Bertino, E. Ferrari, F. Paci, and B. Thuraisingham. PP-trust-X: A system for privacy preserving trust negotiations. *ACM Trans. Inf. Syst. Secur.*, 10(3):12, 2007.

[109] F. Stagni, A. Arenas, and B. Aziz. On usage control in data grids, June 2009. to appear in the 3rd IFIP International Conference on Trust Management (TM'09).

[110] A. J. Stell, R. O. Sinnott, and J. P. Watt. Comparison of advanced authorisation infrastructures for grid computing. In *Proc. of High Performance Computing System and Applications 2005, HPCS*, pages 195–201, 2005.

[111] M. Thompson, A. Essiari, K. Keahey, V. Welch, S. Lang, and B. Liu. Fine-grained authorization for job and resource management using akenti and the globus toolkit. In *Proceedings of Computing in High Energy and Nuclear Physics (CHEP03)*, 2003.

[112] M. Thompson, A. Essiari, and S. Mudumbai. Certificate-based authorization policy in a pki environment. *ACM Transactions on Information and System Security, (TISSEC)*, 6(4):566–588, 2003.

[113] A. Vahdat, T. Anderson, M. Dahlin, E. Belani, D. Culler, P. Eastham, and C. Yoshikawa. WebOS: Operating system services for wide area applications. In *Proceedings of the Seventh Symp. on High Performance Distributed Computing*, 1998.

[114] J. Vollbrecht, P. Calhoun, S. Farrell, L. Gommans, G. Gross, B. d. Bruijn, C. d. Laat, M. Holdrege, and D. Spence. Aaa authorization framework, 2000.

[115] H. Wang, Y. Zhang, and J. Cao. Ubiquitous computing environments and its usage access control. In *InfoScale '06: Proceedings of the 1st international conference on Scalable information systems*, New York, NY, USA, 2006. ACM.

[116] V. Welch, T. Barton, and K. Keahey. Attributes, anonymity, and access: Shibboleth and globus integration to facilitate grid collaboration. In *Proceedings of the 4th Annual PKI R&D Workshop "Multiple Paths to Trust"*, 2005.

[117] W. Winsborough, K. Seamons, and V. Jones. Automated trust negotiation. In *Proceedings of DARPA Information Survivability Conference and Exposition (DISCEX)*, volume 1, pages 88–102. IEEE Press, 2000.

[118] M. Winslett. An introduction to trust negotiation. In *First International Conference on Trust Management (iTrust'03)*, volume 2692 of *LNCS*, pages 275–283. Springer, 2003.

[119] XACML. eXtensible Access Control Markup Language (XACML). www.oasis-open.org/committees/xacml.

[120] M. Xu, X. Jiang, R. Sandhu, and X. Zhang. Towards a VMM-based usage control framework for OS kernel integrity protection. In *SACMAT '07: Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 71–80, New York, NY, USA, 2007. ACM.

[121] W. Yao, K. Moody, and J. Bacon. A model of OASIS role-based access control and its support for active security. In *SACMAT '01: Proceedings of the sixth ACM symposium on Access control models and technologies*, pages 171–181, New York, NY, USA, 2001. ACM.

[122] T. Yu, M. Winslett, and K. E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Transactions on Information and System Security (TISSEC)*, 6(1):1–42, 2003.

[123] L. Zhang, A. Brodsky, and S. Jajodia. Toward information sharing: Benefit and risk access control (barac). In *POLICY '06: Proceedings of the Seventh IEEE International Workshop on Policies for Distributed Systems and Networks*, pages 45–53, Washington, DC, USA, 2006. IEEE Computer Society.

[124] X. Zhang. *Formal Model and Analysis of Usage Control*. PhD thesis, George Mason University, Fairfax, VA, USA, 2006.

[125] X. Zhang, M. Nakae, M. J. Covington, and R. Sandhu. A usage-based authorization framework for collaborative computing systems. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 180–189, New York, NY, USA, 2006. ACM.

[126] X. Zhang, M. Nakae, M. J. Covington, and R. Sandhu. Toward a usage-based security framework for collaborative computing systems. *ACM Trans. Inf. Syst. Secur.*, 2008.

[127] X. Zhang, F. Parisi-Presicce, R. Sandhu, and J. Park. Formal model and policy specification of usage control. *ACM Trans. Inf. Syst. Secur.*, 8(4):351–387, 2005.

[128] X. Zhang, J. Park, F. Parisi-Presicce, and R. Sandhu. A logical specification for usage control. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 1–10, New York, NY, USA, 2004. ACM.

[129] X. Zhang, R. Sandhu, and F. Parisi-Presicce. Safety analysis of usage control authorization models. In *ASIACCS '06: Proceedings of the ACM Symposium on Information, computer and communications security*, pages 243–254, New York, NY, USA, 2006. ACM.

[130] G. Zhao, D. Chadwick, and S. Otenko. Obligations for role based access control. In *AINAW '07: Proceedings of the 21st International Conference on Advanced Information Networking and Applications Workshops*, pages 424–431, Washington, DC, USA, 2007. IEEE Computer Society.