Model Cover Page for Deliverables

Project Number: *IST 1999-11791*
Project Title: *FASTER – Flexible Access to Statistics, Tables and Electronic Resources*
Deliverable Type: (PU/LI/RP)* *PUBLIC*

Deliverable Number: *D 9.1.1*
Contractual Date of Delivery: *29th September 00*
Actual Date of Delivery: *6th October 00*
Title of Deliverable: *System Access Control Analysis and Design*
Work-Package contributing to the Deliverable: *WP 9*
Nature of the Deliverable: (PR/RE/SP/TO/OT)** *RE*
Author(s): *UNIMI TEAM*

Abstract

This document describes the analysis and design of the FASTER access control module. It illustrates the protection requirements gathered from the partners and through user analysis and the access control policies that will be adopted. It proposes a language for expressing access control rules and related access control policies. The document also contains some examples of translation of protection requirements received from the partners into rules of the access control language. The document will be circulated among the partners and the users, who will be requested to translate their organization policies into rules of the language, and provide feedbacks. The document also discusses architectural choices for the implementation of the access control module and its interface with the other modules operating at the server side.

Keywords: Access control policy, access control language, authorizations, restrictions, dataset, metadata.

*Type: PU-public, LI-limited, RP-restricted

**Nature: PR-Prototype, RE-Report, SP-Specification, TO-Tool, OT-Other

1 Introduction

The overall goal of the Faster project is the development of a flexible and open system for the controlled dissemination of statistical information. The scenario can be briefly summarized as follows. The Faster server manages access to data that an organization, called *data publisher*, makes available for external release. Such data are usually passed to the publisher by other organizations, called *data producers* that collect, process, and make data ready for publication. Although the publisher and producer might be the same (as in the case of statistical agencies distributing their data), generally they correspond to different organizations. For instance, statistical agencies (producers) can collect data through surveys, process them and produce datasets summarizing the results of the study, which are then stored at data archives (publishers) for public or semipublic distribution. Datasets can be in the form of *microdata*, reporting information of individual respondents, or *macrodata*, e.g., aggregates combining data of different respondents. Although both micro and macro data usually undergo a de-identification or sanitization process intended to remove explicit identities of the data respondents, or the possibility of inferring them (see Faster Workpackage 8 – Statistical Disclosure Control), data are not just released unconditionally to anybody. Rather, certain data may be released only to specific requesters or under specific conditions. As an example, there are data that can be released only to research or academic institutions, there are data whose release requires payment of the data production fee, and so on.

The development of an access control unit (ACU) for the definition and enforcement of such security requirements is the major task of the UNIMI partner within the Faster project. As it will be discussed more in details in Section 7, the ACU unit will be integrated with other units operating at the server side and will intercept every request submitted to the Faster server to determine whether, and possibly at what conditions, every single request can be granted.

Goal of this document is the proposal of an access control policy for the ACU and of a related language for the specification of security requirements that data publishers may wish to enforce on the data they are making available. The document also discusses architectural design, reporting specifications and discussing technology choices on which the ACU implementation will be based.

To avoid ambiguity in the presentation, Figure 1 reports a simple glossary of terms that are assumed known in this document. This simple glossary is useful since, as we noticed by looking at partners' security policies, often different publishers use different terms to denote a same concept and use the same term to denote different concepts.

The remainder of this document is organized as follows. Section 2 characterizes the form of the requests that are presented at the Faster server, and on which the ACU will need to take the access decision. Section 3 describes the organization at the server of the information managed by the ACU. Section 4 describes the language for expressing security requirements together with examples of rules translating security policies received by the partners. Sections 5 and 6 discusses administration of the ACU and its relationship with the statistical disclosure control module. Finally, Section 7 discusses architectural design.

2 Access requests

We start by characterizing the requests, submitted to the Faster server, to which the ACU will have to provide response. Intuitively, an access request is characterized by three elements:

- the *subject* that makes the request
- the *action* that is being requested

Term	Definition	Similar Terms
Dataset	Object, usually a table, access to which must be controlled. The table can contain either micro or macro data	Dataset (DDA), Data (NSD), Dataset (UK)
Group	Named set of users and (nested) groups. E.g., Students, EC-citizens	Position/Occupation (DDA), Position given by affiliation and function (NSD), Kind of User (UK)
Publisher	Organization that receives data from producers and makes them available for distribution	Distributor (??)
Macrodata	Tabular data computed from the specific micro data collected by aggregating data of different respondents	Aggregate Data (CBS)
Metadata	Information associated with a dataset, describing property of the dataset or its content. It is not contained in the dataset)	Metadata (DDA), Metadata (NSD), Metadata (UK)
Macrodata	Data referred to individual respondents	Macrodata (CBS), Macrodata (UK)
Project	Named activity grouping users working together for a given goal	Project (NSD)
Purpose	Reason for which the data can (or will be) used. It can be associated with users or projects to restricts the use that they can make of the data	use (DDA), intended use (UK)
Producer	An organization that produces data to be made available for controlled release	Donor (DDA), Data Owner (NSD), information supplier (SSB), collectors (UK)
User	Human entity that presents requests to the system	

Table 1: Glossary of terms

- the *object* on which the subject wishes to perform the action

We examine each of them next.

2.1 Subjects

Subjects are entities requesting access to data. The basic concept for the characterization of a subject is the person presenting the request, to which we refer to as user.

User Users are human entities that can connect to the system and make requests. Each user has associated an identifier (usually the user's login), with which the user is referred to in the system.

Although requests are actually typed in by a human user, in the Faster scenario the decision of whether some data may or may not be released may not (only) depend on the user identity but also on the use the user intends to do with the data being requested, possibly declared by the user at the time of the request. As a matter of fact, from the analysis of the partners practices, it appears clear that the use for which the data are being requested plays an important role in the decision of whether the data can or cannot be released. We identify two ways in which the use can be characterized: *purpose* and *project*.

Purpose is the reason for which data are being requested and will be used. Examples of purposes are:

- Pure research
- Commercial
- Teaching
- Personal interest
- Research consultancy

Project is a named activity registered at the server, for which different users can be subscribed, and which may have one or more purposes. Examples of projects can be `Nesstar` and `Faster`.

Accordingly, we characterize each subject making a request to the Faster server with a triple

$$\langle user,project,purpose \rangle$$

meaning *user* is making the access request for a given *project* and/or a given *purpose*.

Note that some elements within the triple may remain unspecified with respect to a given request. This is, for instance, the case of requests made by users who do not belong to any project or who do not declare the purpose for which the data are being requested. Identity also can remain unspecified, as in the case of anonymous requests.

2.2 Objects

Objects are the resources on which users can make requests. We distinguish two objects on which requests can be made: *datasets* and *metadata*.

Datasets are the objects containing statistical information, typically stored in tabular form as micro or macrodata.

Metadata represent information associated with datasets. Metadata are not part of the dataset content. They provide additional contextual information explaining, for example, to which study a dataset is referred, how it has been obtained, by whom, and so on.

We will elaborate more on datasets and metadata in Section 3.2.

2.3 Actions

Actions characterize the kind of access (operation) that the requesting subject wishes to perform on the object. Although specific actions may vary depending on functionalities provided on specific kinds of datasets, we can distinguish three main kinds of actions:

Browse: to visualize and query metadata associated with datasets. With the browse facility, users can walk through the metadata in order to choose the actual dataset they are interested in.

Analyse-on-line: to query datasets. On-Line analysis includes a set of pre-defined operations that perform on-line calculations on selected data. Available operations may vary depending on kind of dataset under consideration.

Download: to download data from the server. It allows users to save whole datasets on their local machine to perform off-line analysis.

2.4 Access requests form

In summary, access requests have the general form

$$\langle \textit{user,project,purpose} \rangle,\ \textit{action},\ \textit{object}$$

where the meaning of the different components is as discussed above, and one or more components of the subject triple may remain unspecified.

The following are examples of requests

- ⟨`tom.smith,Faster,research`⟩, `download`, `dataset1`

  user `tom.smith` requires to `download` `dataset1` for `research` purposes within the `Faster` project.

- ⟨`john.doe,Nesstar,_`⟩, `download`, `dataset1`

  user `john.doe` requires to `download` `dataset1` for use within the `Nesstar` project.

- ⟨`_,_,_`⟩, `browse`, `meta_dataset5`

  an anonymous user with undeclared project and purposes requires to `browse` metadata `meta_dataset5`

- ⟨`_,_,research`⟩, `browse`, `meta_dataset5`

  an anonymous user requires to `browse` metadata `meta_dataset5` for `research` purpose

In the remainder of the document we discuss subject and object organization at the server, the access control policies that we propose to use, and the format of the access control rules whose evaluation determines whether a request can or cannot be granted.

3 Subjects and objects organization at the server

We first define the assumptions on subjects and organization of subject-related information at the server. We next describe organization and management of data and metadata information.

3.1 Subjects

3.1.1 Users and projects profiles

The Faster server recognizes only users and projects registered at the server. Each user and project is assigned an identifier that allows the server to refer to the user (project, resp.). Besides their identifiers, users and projects registered at the server usually have other properties associated with them. For instance, a user may have properties such as name, address, occupation; a project may have properties such as title, abstract, or sponsor. To capture and reason about these properties we assume each user and project is associated with a *profile*. Intuitively, profiles are at users and projects what metadata are at datasets. To be as general as possible, we view profiles as semi-structured documents (XML or RDF like). The profile associated with a user (project, resp.) defines the name and value of some properties that characterize the user (project, resp.). Figure 1 illustrates an example of profile for users and projects.

User profile
Name
Login
Title/Position/Job
Address
Email
Telephone
Validation information
Groups
Purposes
Agreement
On line agreement
Registration Date

Project profile
ID
Title
Abstract
Objectives
Period
Purposes
Sponsor
Leaders
Participants
Responsible institution

Figure 1: An example of user and project profiles

3.1.2 User groups and profile/purpose abstractions

Abstractions can be defined within the domains of users, projects, as well as purposes. Intuitively, abstractions allow to group together users (projects, and purposes, resp.) with common characteristics and to refer to the whole group with a name. For instance, with respect to projects, abstractions can group together all the projects registered by a given organization, all the projects sponsored by a national institution, or all the projects with commercial goals. With respect to purposes, abstractions can correspond to generalization/specialization relationship. For instance, "pure research" and "applied research" can be seen as a specialization of "research". With reference to the user domain, abstractions allow the definition of groups, representing named sets of users. Groups can be nested (i.e., groups can be defined as members of other groups) and need not be disjoint (i.e., a user can belong to more that one group). At a very high level, groups can distinguish the different categories of users highlighted in the Faster functional specifications document, which are:

- academic community
- policy making community
- mass media community
- commercial community

Specializing these communities, we can obtain finer grained or orthogonal classifications of the users. For instance, going at a finer grain we can distinguish, within the academic community, groups `private_schools` vs `state_schools`, or `Faculty` and `Students`. At an orthogonal level we could also classify users based on other aspects such as their citizenship.

Users together with their groups, projects together with their categorizations, and purposes with their abstractions define a partial order on three domains (users, projects, and purposes). Such a partial order introduces a hierarchy on each domain, that we depict as a directed acyclic graph whose nodes are the elements of the domain and the defined abstractions and an arc from node n_1 to node n_2 indicates a *direct* (i.e., explicitly defined) membership of n_1 in n_2. Figure 2 illustrates an example of users, projects, and purpose hierarchies. We assume each hierarchy to be rooted, meaning there is one element to which all

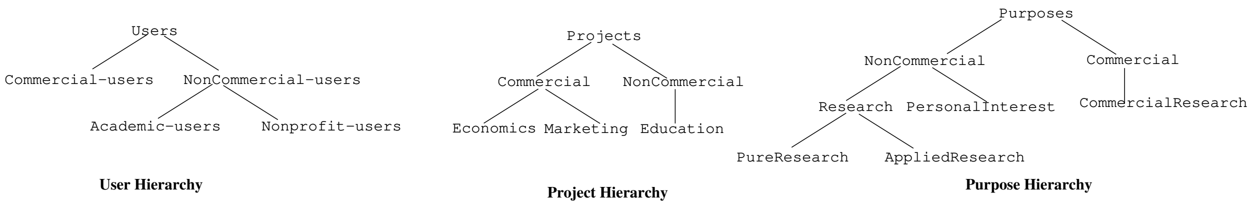

Figure 2: An example of user, project, and purpose hierarchies

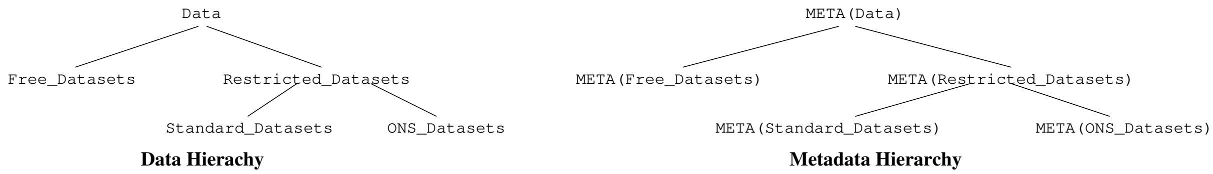

Figure 3: An example of data hierarchy

elements in the hierarchy belongs. This assumption is not limiting (a dummy group to which all elements in a domain belong can be assumed) and is common in many system, where, for instance, a group Public is usually considered to which all the users belong. In the remainder of this paper we assume group `Users` to be the root of the user-group hierarchy.

3.2 Objects

Objects are the entities to which accesses can be requested. As anticipated, the Faster server distinguishes two kinds of objects: *datasets* and *metadata*.

3.2.1 Datasets

Datasets are tabular data containing statistical information access to which is being regulated and can be in the form of microdata or macrodata. For the time being, we consider access to whole datasets only (i.e., an access to a dataset is either allowed or denied). Finer granularity could be considered once the assumption underlying the format of datasets will be final in the Faster project.

Datasets also can be organized in a hierarchical structure, defining sets of datasets that can be collectively referred together with a given name. Like for users, dataset groups need not be disjoint and can be nested. The definition of groups of datasets introduces a hierarchy of datasets and groups thereof. Figure 3 illustrates an example of such a hierarchy. Again, we assume the hierarchy to be rooted. In the remainder of this document, `data` will be assumed to be the root of the dataset hierarchy.

3.2.2 Metadata

Metadata represent data about data. They can be in the form of textual or semistructured documents (XML or DDI). Whatever their form metadata are, at a practical level, files that can be associated with each dataset. Note that metadata are associated only with specific datasets, not with abstractions on them, and no hierarchy is explicitly defined on metadata (see Figure 3). However, the abstraction hierarchy defined on the data reflects in an abstraction hierarchy on the corresponding metadata. We assume a bijective function META() that makes the association between a dataset (or classes thereof) and its metadata (or classes thereof). For instance, given a dataset `dataset1`, function META(`dataset1`) refers to the metadata associated with it. Given a dataset group `Free_Datasets`, function META(`Free_Datasets`) denotes the set of all metadata of the datasets in the group. A metadata document can then be referred to either by its identifier or, via function META, by the identifier of the dataset with which it is associated.

For metadata browsing as well as for the evaluation of conditions that may determine whether or not a given access to datasets can be allowed, it is useful to evaluate the content of metadata. While for textual metadata, we limit the granularity to the whole document, for semistructured documents, we allow reference to finer grained content at the level of properties. Properties (elements and attributes, in the XML terminology) within a metadata document are referred by means of *path expressions*, expressed, for instance, with the XPath language. Basically, a path expression is a sequence of element names or predefined functions separated by character `/` (slash): $l_1/l_2/\ldots/l_n$. Path expressions may terminate with an attribute name as the last term of the sequence. Attribute names are syntactically distinguished by preceding them with special character `@`. Intuitively, semistructured documents can be seen as trees, where each node represents an element or attribute of the considered document and an edge between two nodes represents a containment relationship between them. A path expression $l_1/l_2/\ldots/l_n$ on a document tree then represents all the attributes or elements named l_n that can be reached by descending the document tree along the sequence of nodes named $l_1, l_2, \ldots, l_{n-1}$. For instance, path expression META(`data1`)`/codeBook/stdyDscr/stdyInfo/subject` corresponds to the `subject` elements describing the data collection's intellectual content that are children of `stdyInfo` elements, that are children of the `stdyDscr` elements, and so on, in document META(`data1`).

Path expressions may also include conditions associated with the nodes of a path; in this case the path expression identifies the set of nodes that satisfy all the conditions. Conditions are distinguished from navigation specifications by enclosing them within square brackets. Given a path expression $l_1/\ldots/l_n$ on a document tree, a condition may be defined on any label l_i, enclosing in square brackets a separate evaluation context containing a predicate that compares the result of the evaluation of the relative path expression with a constant or another expression. For instance, expression `/codeBook//sumDscr/[./collDate = "2000-07-05"]` identifies the `sumDscr` elements of all the studies (datasets) whose date of collection (represented by the `collDate` element) is July 5, 2000.

3.3 Actions

As anticipated in Section 2.3, actions to be considered may vary depending on the functionalities provided by the producers on specific datasets. Generally the kinds actions supported include: browse, analyse-on-line, download. Abstractions can also be defined on actions, specializing actions or grouping them in sets. For instance, the three kinds of actions mentioned above can all be grouped in a set called `access` and thus referred to as one.

4 ACU language

In the previous sections, we have discussed the form of access requests and the organization of data and subject-related information at the server. We now introduce a language for the specification of access restrictions on the data.

4.1 Desiderata

Before introducing the specifics of the language it is important to highlight some desiderata that the language should satisfy, and that guided our decisions.

- The language should express access restrictions based on the typical abstractions used by data providers and distributors, such as user categorizations, projects and purpose thereof, operation (or access) type, and data objects categorizations. The aforementioned categories will be user-definable, and hierarchical structures will be supported (see previous section).
- The language should support restrictions based on conditions on meta-data or on local user profile conditions.
- The language should support restrictions related to signed agreements, and other fulfillments to be accomplished via manual procedures.
- The language should have a declarative form. The use of a declarative language makes it easier the task of specifying and especially keeping under control access restrictions.
- The language should be simple and expressive. It should be simple to make easy the management task of specifying and maintaining the security specifications. It should be expressive to make it possible to specify in a flexible way different protection requirements that may need to be imposed on different objects.
- Last but not least, the language should be easy to use to nonspecialists in the field. We could imagine that often, the people specifying the security policies will be employees unfamiliar with procedural or logic based languages. The language should therefore be based on a high level formulation of the access control rules, possibly close to natural language formulation.

4.2 Access control rules

We start introducing the general form of the rules.

4.2.1 Semantics and format of the rules

Declarative access control specifications are generally based on rules, usually in the form of triples called *authorizations* stating which *subjects* can exercise which *action* on which *object*. Different authorizations can be specified for the same subject, object, and/or action. In such a scenario, a request is granted if there is at least an authorization that allows the request. With respect to the specified authorizations, intuitively, this implies that different authorizations are considered as combined in OR. This traditional approach results limiting in the context of the Faster project. As a matter of fact, by looking at the specifications of several partners we noticed that often access restrictions are stated in a restrictive form rather than in the inclusive positive form just mentioned. By restrictive form we mean rules that state conditions that *must* be satisfied for an access to be granted and such that, if at least one condition is not satisfied, the access should not be granted. For instance, a rule can state that "access to dataset-1 can be allowed *only* to citizens". It is easy to see that such a restriction cannot be simply represented as an authorization stating that citizens can be authorized. In fact, while the single authorization brings the desidered behavior, its combination with other authorizations may not, leading the *only* constraint to be not satisfied anymore.

From what said it is clear that the traditional approach of specifying authorizations as positive permissions for the access is not sufficient. Consequently, the ACU language supports two kinds of rules: *restrictions* and *authorizations*. Intuitively, restrictions are useful to specify requirements of the exclusive *only if* form stated above; while authorizations specify requirements in the traditional positive *if* form.

Restrictions specify requirements that must all be satisfied for an access to be granted. Lack to satisfy any of the requirements that apply to a given request implies the request will be denied.

Syntactically, restrictions have the form

$$\langle \textit{subjects} \rangle \ \ \textsc{can} \ \langle \textit{actions} \rangle \ \langle \textit{objects} \rangle \ \ \textsc{only if} \ \langle \textit{conditions} \rangle$$

where:

- *subjects* identifies the set of subjects to which the restriction refers
- *actions* is the action (or class of actions) to which the restriction refers
- *objects* identifies the set of objects (either datasets or metadata) to which the restriction refers
- *conditions* is a boolean expression of conditions that every request to which the restriction applies must satisfy.

Authorizations specify permissions for the access. An access is granted if there is satisfaction of at least one of the permissions that apply to the given request and no restriction is violated.

Syntactically, authorizations have the form

$$\langle \textit{subjects} \rangle \ \ \textsc{can} \ \langle \textit{actions} \rangle \ \langle \textit{objects} \rangle \ [\ \textsc{if} \ \langle \textit{conditions} \rangle]$$

where *subjects*, *actions*, and *objects* have the same syntax and semantics as in restrictions, and $\langle$*conditions*$\rangle$ is a boolean expression of conditions whose satisfaction authorizes the access.

Unlike for restrictions, lack of satisfaction of a condition in an authorization simply makes the authorization inapplicable but it does not imply the access will be denied. In particular, access can be authorized if there is at least one authorization that applies to it for which the conditions are satisfied.

The actions field in the access control rules is simply the identifier of an action or group thereof. Authorizations/restrictions referred to groups of actions are considered applicable to all actions in the set. We now look at the different components in the rule.

4.2.2 Subjects

The field *subjects* is a subject expression that allows the reference to a set of subjects depending on whether they satisfy or not certain conditions, where conditions can evaluate the user's profile or its membership in groups, the project's profile or its membership in a category, and the purpose value or its inclusion in an abstraction.

To make it possible to refer to the user, project, or purpose of the request being evaluated without need of introducing variables in the language, we introduce the following keywords, whose appearance in a conditional expression is intended to be substituted with the actual parameters of the request in the evaluation at access control time:

user indicates the identifier of the person making the request

purpose indicates the purpose declared by the user for the request

project indicates the project declared by the user for the request

The value is “undefined” in the case no actual value has been declared.

We assume profiles to be referenced with the identity of the corresponding users and projects. Single properties within users and projects profiles are referenced with path expressions denoting the path from the root to the property. For instance, `Faster/sponsor/address` indicates the address of the sponsor of the Faster project. Here, `Faster` is the identity of the project (and therefore the identifier for the corresponding profile), and `sponsor/address` the path name of the address property. As another example, **user**`/affiliation` indicates the property `affiliation` within the profile of the user whose request is being processed.

The subject expression is a boolean formula of terms that evaluate conditions on the user, project, and purpose of the request. Conditions can evaluate the value of the elements, their membership in defined groups/categorizations, or properties in their profiles.

The following are examples of subject expressions:

- **user**`/citizenship=EC` AND (**project**`/sponsor=EC` OR **purpose** IN `research`)

  denoting requests made by users who are European citizens and intend to use the data for research purposes or within an EC funded project

- **user** IN `NonCommercial-users` AND **purpose** IN `research`

  denoting requests made by users belonging to group `NonCommercial-users` that intend to use the data for research purposes

- **user** IN `NonCommercial-users` AND **purpose** IN `research` AND **project**`/sponsor=EC`

  denoting requests made by users belonging to group `NonCommercial-users` that intend to use the data for research purposes within an EC funded project

In the case of authorizations applicable to all users within a given group or that request access for a given project or purpose (or category thereof), the group, project, and/or purpose element can be explicitly factorized out of the subject expression and isolated. The explicit reference to users/groups, projects, and purposes allows the indexing of the ACU rules and consequently improves performances in the access control.

Intuitively, a subject expression of the form

user IN user-id AND **project** IN project-id AND **purpose** IN purpose-id AND subject-expression

can be turned into an indexable expression of the form

user-id OF project-id PROJECTS FOR purpose-id PURPOSES WITH subject expression

where the clauses “ OF project-id PROJECTS”, “ FOR purpose-id PURPOSES”, and “WITH subject expression” are optional and can be omitted.

4.2.3 Objects

The field *objects* is an expression that identifies the set of objects to which the authorization/restriction applies. Like for subjects, to make it possible to refer to the dataset (or associated metadata) to which the request being processed refers, without need of introducing variables in the language, we introduce the following keywords, whose appearance in a conditional expression is intended to be substituted with the actual parameters of the request in the evaluation at access control time:

dataset indicates the identifier of the dataset to which access is requested

metadata indicates the identifier of the metadata document to which access is requested or associated with the dataset to which access is being requested.

Element *objects* has the form

object-id [WITH *conditional-object-expression*]

where:

- *object-id* is
  - the identifier of a dataset or group of datasets
  - the identifier of a metadata document, or group of them, with possibly associated an Xpath expression identifying portions of the document.
- *conditional-object-expression* is a boolean formula of conditions that can evaluate membership of the object in categories, values of properties on metadata and so on.

Some examples of objects are as follows:

- `Free_Datasets` WITH META(**dataset**)`/producer=ACME`

  denoting all datasets in the `Free_Datasets` class produced by ACME (where the produced in specified as property in the associated metadata)

- META(`Restricted_Datasets`)`//question_text`

  denoting the text of the question_text element of the metadata documents associated with datasets in the `Restricted_Datasets` set.

4.2.4 Conditions

Element *conditions* within access control rules defines conditions that must be satisfied for the request not to be rejected (as in the case of restrictions) or for the access to be granted (as in the case of authorizations).

We distinguish two kinds of conditions: *static* and *dynamic* conditions.

Static conditions evaluate membership of subjects and objects into classes or properties in their profiles and associated metadata. These kind of conditions are included in the *conditions* element of the rules for restrictions; while for authorizations all static conditions should be included in the subject and object conditional expressions.

Dynamic conditions are conditions that can be brought to satisfactions at run-time processing of the request. They can require interfacing with a database maintained at the server, and are usually related to agreement acceptance, payment fulfillment, or registration. Dynamic conditions require attachment to procedural calls that execute the necessary actions to bring the condition to satisfaction. We have identified the following kinds of dynamic conditions:

- *Agreement acceptance* The request can be granted only if the requester subscribes conditions imposed on the access. Agreement can be as simple as clicking an "ok" button on a pop up window
- *Payment* The request can be granted only if the requester pays (or has paid) a fee imposed on the release

Keywords	CAN , WITH , IF , ONLY IF , IN , AND , OR , NOT , IF , FOR , META, PURPOSES, PROJECTS	
Predicates	**agreement**(*id*,*a*)	checks if user *id* has accepted agreement *a*, and if not presents the user with the agreement. It returns true if the agreement has been accepted.
	register_user(*id*)	check if project *id* is registered, and if not starts the registration procedure for the user. It returns true if the user was registered or the registration has been successfully completed.
	register_project(*id*)	check if project *id* is registered, and if not starts the registration procedure for the project. It returns true if the project was registered or the registration has been successfully completed.
	fill_in_form(*id*,*form*)	checks if user *id* has filled in form *form*, and if not presents it to the user. It returns true if the user has filled the form.
	payment(id_1,id_2)	checks if user id_1 has paid to access object id_2, and if not starts the payment procedure for the user. It returns true if the user had paid or the payment procedure completes successfully.
Reserved identifiers	**user**	bounded to the identity (if defined) of the user making a request
	project	bounded to the project (if defined) specified by the user making a request
	purpose	bounded to the purpose (if defined) specified by the user making a request
	dataset	bounded to the identifier of the dataset to which access is requested
	metadata	bounded to the identifier of the metadata document to which access is requested or associated with the dataset to which access is being requested.

Figure 4: List of keywords, predicates, and reserved identifiers of the ACU language

- *Registration* The request can be granted only if the requester completes a registration procedure.
- *Form filling* The request can be granted only if the requester fills in a form.

For each of them we assume the existence of a procedure that perform the control and possibly trigger the necessary actions. The procedure returns either a true or a false value depending on whether or not the implemented condition was or has been brought to satisfaction.

Figure 4 summarizes the list of keywords, predicates and reserved identifiers of the ACU language. Figure 4.2.4 reports the BNF syntax of the language.

4.3 Examples of ACU rules for partners practices

We give some example of ACU rules corresponding to security practices followed by partners of the Faster project.

DDA: DDA data can be made available only to registered users

`Users` CAN `access data` WITH **dataset**`/producer=‘‘DDA’’` ONLY IF *register*(**user**)

DDA: Data in Category2 can be accessed only to users who have a project registered

`Users` CAN `access Category2` ONLY IF **user**/project $\neq$ nil

or, in alternative

`Users` CAN `access Category2` ONLY IF *register*(**project**)

```
restriction          ::=  subject-expression CAN action object-expression ONLY IF condition
authorisation        ::=  subject-expression CAN action object-expression [ IF complex-condition]
subject-expression   ::=  user-id [ OF project-id PROJECTS] [ FOR purpose-id PURPOSES] [ WITH condition]
action               ::=  identifier
object-expression    :=   object-id [ Xpath-expression ] [ WITH condition]
user-id              ::=  identifier
object-id            ::=  identifier | META(identifier)
purpose-id           ::=  identifier
project-id           ::=  identifier
complex-condition    ::=  simple-condition | complex-condition AND complex-condition |
                          complex-condition OR complex-condition | NOT complex-condition
path                 ::=  entity | entity/property
simple-condition     ::=  path math-op value | path math-op path | predicate
math-op              ::=  = | < | > | <= | >= | IN
entity               ::=  user | project | metadata | data | purpose
predicate            ::=  identifier (list)
list                 ::=  identifier | list [, identifier]
property             ::=  identifier | identifier/property
value                ::=  string
```

Figure 5: BNF syntax of the access control rules

DDA: No restriction for Category4 but any publication require a written permission from producer

`Users` CAN `access Category4` ONLYIF *agreement*(**subject**, Permission.Id)

DDA: Any access for Category 5 require a written permission from the data producer

`Users` CAN `access Category5` ONLY IF **metadata**/producer IN **user**/permissions/Category5

DDA: Free access to Category1 (except for commercial use) to registered users

`Users` WITH **purpose** $\neq$ "Commercial" CAN `access Category1` IF *register*(**user**)

DDA: Everybody can freely browse metadata

`Users` CAN `browse` META(`data`)

NSD: Users can access ADSN (anonymised data of Statistics Norway) only if they are registered and they have a registered project

`Users` CAN `download ADSN_data` ONLY IF *register*(**user**) AND *register*(**project**)

NSD: Question texts of market research should not be revealed to competitors

`Users` CAN `browse` META(`market-research`)//questions ONLY IF NOT **user** IN `Competitors`

UKDA: Everybody can access `Free_Datasets`

`Users` CAN `access Free_Datasets`

UKDA: Access to datasets not in Free_Datasets allowed only to registered users

`Users` CAN `access data` WITH NOT **dataset** IN `Free_Datasets` ONLY IF *register*(**user**)

UKDA: The user can execute on-line-analysis on Standard_Datasets if the user is NonCommercial and has been shown the Standard Conditions Document

`NonCommercial-users` CAN `analyse-on-line` `Standard_Datasets` IF *agreement*(**user**,`SCD.Id`)

UKDA: The user can download Standard_Datasets if project is NonCommercial

`Users` OF NonCommercial PROJECTS CAN `download` `Standard_Datasets`

UKDA: The user can download Data ONS_Dataset if project is NonCommercial or (project is a Consultancy Project and has been shown Consultancy Condition)

`Users` OF `NonCommercial` PROJECTS CAN `download` `ONS_Datasets`

`Users` OF `Consultancy` PROJECTS CAN `download` `ONS_Datasets` IF *agreement*(**user**,`CC.Id`)

5 ACU administration

As illustrated in the previous sections, the access control system requires the storage and management of information on users and projects as well as of the access control rules. Insertion, deletion, or modification of such information is performed through administrative operations reserved to one or more privileged users.

Administrative actions include:

- Project registration and administration
- User registration and administration (including group management)
- Access control rule administration (granting and revoking)

For the time being, we assume all such operations to be reserved to a privileged user, which we refer to as System Security Officer (SSO). We note that our proposal can be enhanced in future by allowing decentralization of responsibility, where different users can be in charge of different administrative actions or of managing authorizations on different parts of the system.

6 Relationships with SDC

Besides the security restrictions implemented by the ACU, the Faster server will need to obey statistical disclosure control restrictions (as imposed by the SDC module). Intuitively, SDC restrictions sanitize datasets by removing explicit identifiers and generalizing or suppressing quasi-identifiers (i.e., data that can bring to linkage). The enforcement by the ACU determines whether or not a request on certain data should be granted or denied, the specific view to be returned in response to the request is then determined by the SDC unit. In Faster it is assumed that SDC operates "off-line" producing, for each dataset, one or more views that can be released. To make it possible to retrieve the view applicable to the specific requests, upon completion of the access control procedure, the control is passed to the SDC module together with the specification of the requestor (meaning the triple $\langle$user,purpose,project$\rangle$). The SDC module can thus return the proper view of the data applicable to the request.

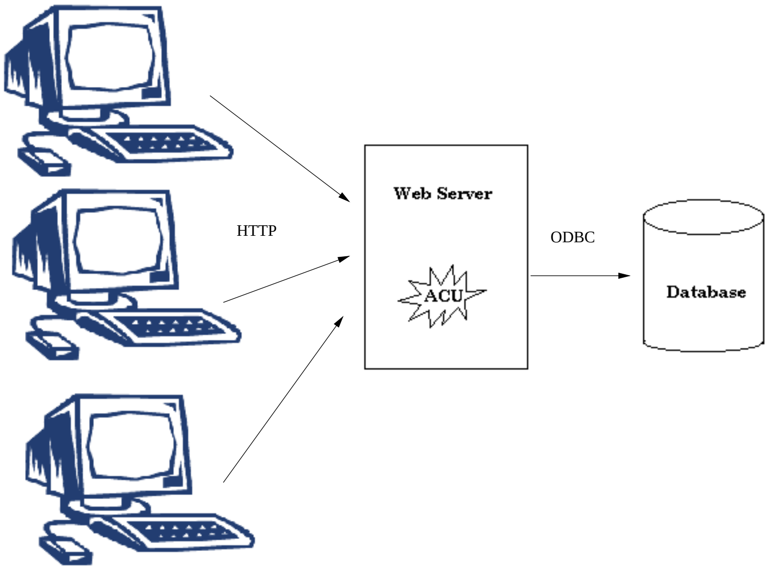

Figure 6: A Three-tier Client Server System

7 Specifications and Architectural Design

We are now ready to provide a set of *specifications* that complement the requirements and features of the *Access Control Unit* (ACU) described in the previous sections. Such specifications include a number of technology-related choices and constraints, and lead to the architectural design that will be described in the second part of this section. The philosopy underlying the entire set of ACU specifications is to ensure a clean integration of ACU as a *component* in the framework of a complete Web-based three-tier application for controlled dissemination of statistical information, involving a *Front-end* (FE), a *Back-End* (BE) and an *Intermediate Layer* (IL) of application logic composed by a set of interoperable components (see Figure 6). The ACU is one of such components; in the sequel, we shall refer to the rest of the IL as the *host application*.

We report here three main specifications that the overall host application must satisfy.

- *Internet/WWW FE* Users must be able to access datasets using the Internet/WWW; besides reaching a wide audience, WWW access brings many well-known advantages in terms of information management.
- *High Availability IL* The number of simultaneous access to datasets must be limited only by hardware constraints.
- *Flexible Data Access BE* A variety of dataset format and storage techniques must be supported by the Back End

7.1 ACU Specifications

The specifications that are specific to the ACU component design can be summarized as follows.

Declarative protections Datasets protection must not be hard-wired in the ACU; rather, administrators will use a simple language, supported by software tools, to specify the policies the system will follow.

Connection to Front-End The ACU does not communicate explicitly with the FE. The Web server FE must be able to activate the ACU component via a standard technique. ACU transparently supports internal multithreading to allow for dealing with simultaneous requests.

Connection to Back-end ACU explicitly communicates with the Back End to access information about users and current access control policy. Standard interfaces must be provided to allow the ACU to access the local databases containing user profiles, agreement forms and/or records, and manual procedure status. The access control policy must be pre-compiled in a suitable form, usable by the generic access control engine that will be part of the access control unit. The compilation phase will check consistency between the specification and the adopted categorizations

7.2 Static Interface of the ACU Component

The interface offered by the ACU is represented by a set of classes modeling all the entities and concepts introduced by the access control model. The formalization of the dynamic behavior is available separately as a *Universal Modeling Language* (UML) Class diagram.

7.3 Dynamic Behavior of the ACU Component

Concerning the ACU component's dynamic behavior, each access request will always result in a *final decision* value being returned to the rest of the system, notifying either that the requested access is granted, or that it is denied. Thus, the ACU component wraps up entirely the computation of access permissions to individual datasets. The formalization of the dynamic behavior is available separately as a *Universal Modeling Language* (UML) Class diagram. The steps are the following:

- *Connection* For conveniency, we model the front-end reaction to a connection request as the creation of a transient Connection Handler object. The actual connection technique depends on the host application.
- *ACU Instantiation/Activation.* As a result of the request, an ACU object is istantiated by the Connection Handler (or an additional thread is activated).
- *Authorizations Identification* The ACU object creates a Subjects object which fully encapsulates the information needed to identify the authorizations that apply to that specific user. A fast access read-only memory structure linking subjects to authorizations is used for speed.
- *Final Decision Computation and Notification* After getting the available data about the authorizations that apply the requestor, ACU computes the final decision granting or negating access. During this computation, dialog with the user may or may not be triggered.
- *User Dialogue* The dialogue is driven by the declarative policy specification. If conditions for the required access are already true, access is permitted immediately, and no dialogue is triggered. If none of the specified conditions that might grant the requested access can be fulfilled, then no dialogue is triggered, and access is denied. If some of the specified conditions might be fulfilled, e.g., by signing an agreement or by authenticating the user, then the user is prompted with the possible actions that would result in the required access. If the user immediately executes the suggested actions and fulfills the conditions, access is granted. Otherwise access is denied (until the user fulfills the specified conditions).

7.4 Architectural Design

From the architectural point of view, our design is fully *server side*: all the message exchanges except the connection itself take place on the server. ACU design relies on the concept of *software patterns*. The goal of patterns within the software community is to create a body of literature to help software developers resolve recurring problems encountered throughout all of software development. An *architectural pattern* expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them. In the ACU design, we follow the standard *Processor* and *Reactor* design patterns, allowing for different deployment solutions according to implementation platforms' performance profiles.

- The *Processor* pattern-based design envisions ACU as a blocking, monolithic object to be synchronously invoked by the application each time a user requires a dataset. It is responsibility of the application to provide ACU with the user and dataset identification via suitable IDs. The request cannot proceed until the invocation provides its return value, i.e. final decision has been taken by the ACU.
- The *Reactor/Proactor* pattern demultiplexes and dispatches requests that are delivered concurrently to an application by one or more clients. The resulting design envisions ACU as a non-blocking device, including the Processor and an asynchronous interface which is notified by the host application of incoming requests. The host application can then go on to process other requests and is asynchronously notified by the ACU interface about the outcome.

7.5 Technologies Considered

Currently, the main technologies for n-tier web based development are the *Distributed Component Object Model* (DCOM) and the *Java+ Common Object Request Broker Architecture* (CORBA) combination. DCOM is a proprietary technology from Microsoft, while Java has become a significant force in application development especially in distributed applications that can leverage its "built for network" design. It is well-known that Java has its own distributed computing architecture called *Remote Method Invocation* (RMI). However, RMI has received little support in the industry, given that CORBA is a non-proprietary, relatively well understood and perhaps more reliable technology. We believe that DCOM and Java+ CORBA represent the two most viable technologies for ACU implementations. The specific technologies used/under consideration for this project are discussed further below:

- *Java Development Kit* (various versions): A series of utilities and a virtual machine for developing Java programs.
- *CORBA broker and middleware* (version 3.0): Enables a Java program to communicate with a middleware application or another Java program (such as a server) using IIOP. IIOP is the method for communication in CORBA.

Java and CORBA do not restrict themselves to a specific machine architecture. Java is architecturally neutral and therefore complements CORBA's infrastructure. CORBA provides well tested distributed services; it extends Java with a distributed object infrastructure that includes a rich set of distributed object services (transactions, cyptographic security, etc.) Finally, Java code can be deployed and managed from a central server.

7.6 Implementation Guidelines

Although implementation is still at a preliminary stage, some basic guidelines can be given. Namely, for the Processor (which implements the entire ACU in the simpler design) a pure-Java implementation can be made. The ACU component is implemented as a complex Java object inheriting from Java Servlet class. Servlets are Web server extensions written in Java and associated with particular URLs. When a request for the URL of a servlet is received from a Web browser, the Web server invokes the servlet to process the request, providing it with all the available information about the request. Servlets can be pre-loaded by the server for efficiency or loaded on-the-fly, as they are needed. Our pre-loaded servlet wraps up a complete Java package, including classes for objects and subjects' representation. On the other hand, the Reactor/Proactor based design involves the same Java Processor, completed by an interface communicating with the rest of the application via CORBA support for asynchronous events notification.

8 Bibliography

The following is a list of related papers, acknowledging the support of the EC community under the Faster project.

- P. Bonatti, P. Samarati "Regulating Service Access and Information Release on the Web," in *Proc. of the Seventh ACM Conference on Computer and Communications Security*, Athens, Greece, November 1-4, 2000.

- P. Bonatti, S. De Capitani di Vimercati, P. Samarati "A Modular Approach to Composing Access Control Policies," in *Proc. of the Seventh ACM Conference on Computer and Communications Security*, Athens, Greece, November 1-4, 2000.

- E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, "Design and Implementation of an Access Control Processor for XML Documents" in *Computer Networks*, Elsevier, vol. 33, no. 1–6, June 2000, pp. 59-75; and *Proc. of the Ninth International World Wide Web Conference (WWW9)*, Amsterdam, NL, May 15-19, 2000.

- E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati "Regulating Access to Semistructured Information on the Web", in *Proc. of the IFIP-TC11 International Conference on Information Security*, Beijing, China, August 21-25, 2000.

- E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati "XML Access Control Systems: A Component-Based Approach" in *Proc. of the IFIP WG11.3 Working Conference on Database Security*, Schoorl, The Netherlands, August 21-23, 2000.

- E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, "Securing XML Documents," in *Proc. of the 2000 International Conference on Extending Database Technology (EDBT 2000)*, Konstanz, Germany, March 27-31, 2000.