A Component-based Architecture for Secure Data Publication*

Piero A. Bonatti[1] Ernesto Damiani[1] Sabrina De Capitani di Vimercati[2] Pierangela Samarati[1]

(1) Dip. di Tecnologie dell'Informazione
Università degli Studi di Milano
Via Bramante 65, 2601 Crema - Italy
{bonatti,damiani,samarati}@dti.unimi.it

(2) Dip. di Elettronica per l'Automazione
Università degli Studi di Brescia
Via Branze 38, 25123 Brescia - Italy
decapita@ing.unibs.it

Abstract

We present an approach for controlling access to data publishers in the framework of Web-based information services. The paper presents a model for enforcing access control regulations, an XML core schema and namespace for expressing such regulations, and illustrate the architecture of Access Control Unit *(ACU), an autonomous software component based on the proposed model. Besides "standard" authorizations, the ACU supports authorizations based on user profiles and* dynamic *conditions whose outcome is determined by user actions such as the acceptance of a written agreement and/or payment.*

1 Introduction

Many financial, industrial, and banking operations worldwide use the World Wide Web for distributing information in the form of structured or semi-structured data, available for download and/or for remote analysis and graphical representation. Such Web-based interchange of information is regarded as a key activity by *data producers* in the private as well as in public sectors (e.g., government agencies and research institutions). In the past, each data producer made its own data available for external release. Today, innovative *Web-based information services* are being developed where producers exploit the mediation of information brokers, called *data publishers*, which use the Web to collect and distribute data from various data producers. For instance, a data producer could make available a survey under the condition that it must be accessible only for research purposes or by academic institutions. Additional protection requirements could also be specified by the data publisher (e.g., privacy or law regulations specific to the country where it operates). This layered scenario poses an entirely new set of challenges to access control systems. Some of the main requirements are discussed below.

Interchangeable policy format Data producers need to specify protection requirements on the data they make available using a format both human- and machine-readable, easy to inspect and interchange. This format should be simple to complement and check on the part of data publishers for being compliant with externally defined regulations; also, it should be simple enough to be readily understood by non-specialists.

Interactive enforcement The traditional access control process operates in two phases: an *evaluation* phase evaluates the access policy and makes a decision, and an *enforcement* phase applies the decision. In Web-based information services, enforcing often requires more than just granting or denying an action. Rather than providing a simple yes or no decision, enforcement should provide a way of interactively evaluate the satisfaction of access restrictions, possibly managing complex user interactions such as the acceptance of written agreements and/or on-line payment for each report. Web-based systems should then guide users in acquiring the permission to obtain the desired data/services. To this end, the system should support conditions—called *dynamic*—that can be made true at access control time (e.g., by filling in a form or by signing an agreement).

Metadata support Semi-structured metadata formats are increasingly important for Web-based services, and are at the basis of the ongoing Semantic Web initiative of the World Wide Web Consortium (W3C) (`http://www.w3.org/2001/sw`).

*The work reported in this paper was partially supported by the European Community within the Fifth (EC) Framework Programme under contract IST-1999-11791 – FASTER project.

While traditionally important for information discovery and retrieval in a networked environment, metadata can also be exploited to provide access control by selectively releasing data based on conditions on their metadata.

Scalability Finally, access control systems for Web-based services must be scalable, as the availability of on-line subscriptions means that user communities may grow without warning. To achieve scalability, the access control enforcement functionality must be cleanly separated by the storage and publishing systems and easily integrated in a variety of server environments. *Component-based* software architectures offer a well-understood solution to the integration problem.

This change in paradigms and requirements has also been noted by other researches and several approaches have been proposed presenting access control models and systems with enriched functionalities. Among this, [9] presenting a uniform framework for the specification of multiple access control policies, [8] focusing on the protection of semistructured data sources in the form of XML documents, [4, 15] proposing access control based on digital certificates, [3] proposing an algebra for defining access control policies from autonomous and independent components, [10] introducing provisional-based access control where granting of requests may depend on the execution (by the system or the user) or specific operations. All these proposals focus on specific aspects with which access control should be enriched and present general solutions. In this paper, we present a complete model, and related language, that attempts to cover the need for enriched functionalities within the context of Web-based data dissemination. The model has been conceived within the context of the EU funded FASTER project, whose aim is the development of a system for making information maintained at national archives selectively available to the external world. Gathered requirements and proposed solutions have been therefore guided by the actual needs of the project' partners, which as we will see do reflect the need of many similar institutions and scenarios. In particular, our model supports access rules that allow reference of properties of the requester or the data being accessed; this allows the convenient specification of regulations where the ability to access data depends on characteristics that cannot be modeled (as traditionally done) through user or data groups, possibly because of their dynamic nature. Also, our model supports dynamic conditions (such as agreement acceptance, or payment) that can be brought to satisfaction at run time by interacting with the user. The model is therefore not bound to producing a yes/no decisions to access requests but can handle more dynamic situations. Also, the model supports both necessary and sufficient conditions for access (recalling rules used in the paper world practice) in the form of authorization and restrictions in the language. Another noticeable aspect of our proposal, coupled with the expressiveness discussed, is the simplicity of the access control language. The language provides expressiveness and simplicity by adopting semistructured data formats for information representation and by using few reserved identifiers for referring to request parameters, without the need of introducing and managing variables in the language. Our system is based on a flexible model and language expressed via an XML Schema and namespace (`http://www.w3.org/XML/Schema`), in line with the ongoing standardization effort toward XML-based languages for access control.[1] A preliminary version of our model appeared in [2]. In this paper, we extend the model with the consideration of "dynamic conditions" and present an XML-based language for expressing protection requirements. We also describe a component-based approach toward the design and development of an *Access Control Unit* (ACU) based on the proposed model.

2 Elements of the access control model

The development of an access control model requires the characterization of entities to be protected (*authorization objects*), the entities against which access must be controlled (*authorization subjects*), and the operations that subjects can request on objects (*actions*).

2.1 Objects: Datasets and metadata

In our model, authorization objects are both datasets and metadata. Datasets can be any kind of data units (e.g., tagged documents, reports, or tables) collected from data publishers for distribution, and can be organized in abstractions defining groups of datasets that can be collectively referred to with a given name. Groups may reflect the file system organization in directories and/or orthogonal abstractions defined by grouping datasets with common characteristics. Datasets and groups thereof define a partial order that naturally introduces a hierarchy [9]. Figure 1 illustrates an example of dataset hierarchy DH, where for simplicity single datasets are omitted and only the groups of datasets are reported. The hierarchy, whose root `Data` groups all datasets in the system, distinguishes between datasets from national surveys and datasets from foreign surveys

[1] This effort recently brought to the announcement of a new Technical Committee in the framework of OASIS (`http://www.oasis.org`) aimed at the definition of a standard XACML (eXtensible Access Control Markup Language).

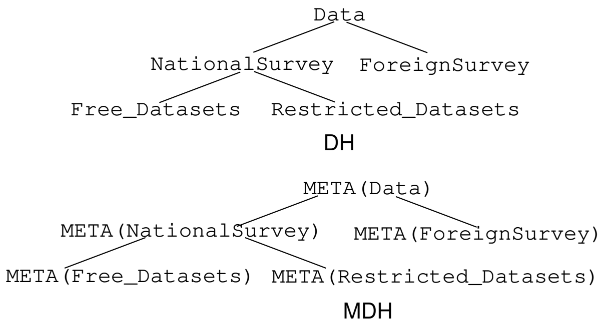


Figure 1. An example of dataset (DH) and metadata (MDH) hierarchies

and, within the latter, between datasets containing publically available information and datasets whose released should be regulated.

Besides actual datasets, each data publisher possibly maintains a collection of *metadata*. Metadata are usually not part of the dataset content; rather, they provide additional information on datasets that can be provided to users, and may be of help in searching for specific data. For instance, metadata may report the name of the statistical agency releasing a dataset and how and when a dataset was obtained. Several standards have been proposed for interoperable metadata interchange in the digital libraries domain (e.g., Z39.50, Dublin core, and RDF [6]). To make our approach generally applicable we do not make any assumption on the metadata format. In our model, metadata can be in the form of textual or semistructured documents (e.g., XML [1] or DDI [12]). No hierarchy is explicitly defined on metadata. However, the dataset hierarchy implicitly defines a metadata hierarchy MDH (see Figure 1). A bijective function META() makes the association between a dataset (or a group thereof) and its metadata (or the group of them). For instance, given dataset `d1` and group `NationalSurvey`, function META(`d1`) returns the metadata associated with `d1`; function META(`NationalSurvey`) returns the set of metadata documents associated with the datasets in `NationalSurvey`. A metadata document can then be referenced either through its identifier or, via function META, through 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. In this way, we can specify, for example, a protection requirement

```
<?xml version='1.0'?>
  <codeBook>
    <docDscr DataID="DATASET01" link="http://.../">
      <citation>
        <titlStmt>
          <title>The EU Referendum Study</title>
        </titlStmt>
        <prodStmt>
          <producer>NSSDS</producer>
          <prodDate>10-06-2000</prodDate>
          <prodDate>05-09-2001</prodDate>
          <prodPlac>NSD Bergen</prodPlac>
        </prodStmt>
      </citation>
      <docSrc DocSrcID="DOC003" link="http://.../">
        <citation>
          <titlStmt>
            <title>ISS-Rapport nr. 45,...</title>
          </titlStmt>
          <distStmt>
            <distrbtr abbr="NSSDS">
              Norwegian Social Science Data Service
            </distrbtr>
            <depositr affiliation="ISS">
              Department of Sociology and Political Science
            </depositr>
          </distStmt>
        </citation>
      </docSrc>
      ......
    </docDscr>
  </codeBook>
```

Figure 2. An example of a portion of an XML metadata document

stating that a subject can access all datasets produced in the current year, where the production year is a property specified in the metadata associated with datasets. Properties (elements and attributes, in the XML terminology) are referenced by means of *path expressions* written, for example, with the XPath language [16]. Since semistructured documents are usually modeled by means of tree structures [1], a path expression $l_1/l_2/\ldots/l_n$ on a document tree can be seen as a sequence of node labels l_i representing a path in the tree. To illustrate, consider the portion of XML metadata document associated with dataset `DATASET01` illustrated in Figure 2; it includes information about a comparative study of European referendums. Path expression META(`DATASET01`)`/codebook/docDscr/citation/titlStmt/title` identifies the title ("The EU Referendum Study") of the study with which the considered metadata document is associated.

2.2 Actions

Both datasets and their metadata can be accessed by remote subjects via different actions. Flexibly expressing and enforcing authorizations on such actions constitute the main functional requirements for our system. Flexibility is particularly important as actions supported by a specific distribution service may of course vary, depending on the nature of the service. Actions can be grouped into classes. For instance, possible classes are the following:

- *Browse* metadata associated with datasets. With the browse facility, users can navigate through the metadata to choose the actual dataset they are interested in.

- *Analyze-on-line* datasets. On-line analysis includes a set of pre-defined data analysis operations. Available operations may include graphic representation and in general will vary depending on the kind of dataset under consideration.

- *Download* data from the server. Downloading allows users to save whole datasets on their local machine in order to use off-line tools for data analysis.

Further abstractions can also be defined on actions, specializing actions or grouping them in sets. For instance, the three classes above can all be grouped in a set called `Access` and thus referred to as one.

2.3 Subjects

Traditional access control models characterize subjects by means of their *identities*, which are used as a basis for specifying access restrictions. However, often the decision of whether some data may or may not be released does not depend only on the requester but also on what she intends to do with the data being requested. *Use-based* access restrictions are seldom supported in current access control systems; still, they appear to be one of the requirements that should be addressed in data dissemination [13]. By analyzing current practices at the data publishers consulted, we have identified two different ways in which the use can be defined: *purpose* and *project*. A purpose represents the reason for which data are being requested and will be used; `statistical`, `educational`, and `scientific` are examples of purposes. A project is a named activity registered at the server that may have one or more purposes and for which different users can be subscribed. For instance, a financial agency involved in a consultancy project for a customer can register the project to the archive so that all its employees working on it can enjoy their employer's privileges for accessing data.

Consequently, in our model, each subject submitting access requests to the data publisher is characterized by a triple of the form ⟨`userid`,`purpose`,`project`⟩, where `userid` is the login with which the user has connected to the system, `purpose` and/or `project` are the reasons for which the specific access is requested, that is, the intended use of the data. Note that one or more elements within the subject triple may remain unspecified, denoted "_". This may happen, for example, in the case of requests from anonymous users or for which the user does not specify the intended use (in terms of project or purpose) of the data requested.

Access requests are then characterized by a subject triple, the action requested, and the object on which the action is requested. For instance, ⟨`john.doe`,`commercial`,`TokyoStockExchange`⟩, `download`,`dataset1` defines a request by user `john.doe` to `download` `dataset1` for `commercial` purposes within the `TokyoStockExchange` project.

To ease the specification and management of access restrictions, our model supports the definition of abstractions within the domains of users, projects, as well as purposes. Intuitively, abstractions allow the grouping of entities with common characteristics (e.g., all the projects registered by a given organization or all the projects with commercial goals) and the reference to the whole group with a name. With reference to the user domain, abstractions allow the definition of groups, representing named sets of users, as usually supported in current access control systems [5, 9]. At a very high level, groups can distinguish the different communities of users who may need access to a data archive, such as: academic community, policy making community, mass media community, and commercial community. Specializing these communities, we can obtain finer grained or orthogonal classifications of the users. Users together with their groups, projects together with their categories, and purposes with their abstractions define a partial order on the three domains (users, projects, and purposes). Such a partial order introduces a hierarchy [9], which is essentially an acyclic graph, on each domain. Figure 3 illustrates an example of user-group (UGH), project (PRH), and purpose (PUH) hierarchies, respectively. Again, for the sake of simplicity, leaf nodes (corresponding to individual users, projects, and purposes) are omitted. As we shall see in the sequel, authorization subjects are triples on the Cartesian product of the sets in the three hierarchies. This permits, for example, to state that a group of users can access a dataset for a specific purpose, or that a single user can access a dataset for a set of purposes.

In our system, a server recognizes only locally registered users and projects. Each user and project is assigned an unique 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, and occupation; a project may have properties such as title, abstract, and sponsor. To capture these properties

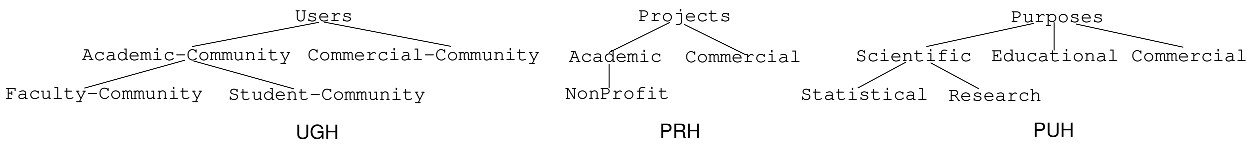


Figure 3. An example of user-group (UGH), project (PRH), and purpose (PUH) hierarchies

we assume each user and project is associated with a *profile* [2]. Intuitively, profiles are to users and projects what metadata are to datasets. Profiles can be modeled as semi-structured documents (XML or RDF like [1]). Profiles results very convenient for the support of access restrictions based on properties the requester enjoys which cannot be represented through user groups because of manageability, or because of their dynamic nature.

3 Access control language

We now illustrate the access control rules by which data publishers can specify access regulations to be enforced on the data. We start by introducing the components of the rules. We then give their syntax, in XML format, and their semantics.

3.1 Components of the access control rules

Access control rules specify the permissions/restrictions to be enforced. Rules, whose format and semantics will be discussed in the next section, are specified by defining the following components:

- *subject expression* defines the set of subjects to which the rule applies (i.e., whose requests are regulated by the rule);
- *object expression* defines the set of objects to which the rule applies (i.e., requests to access which are regulated by the rule)
- *action* defines the actions to which the rule applies (with the other two components it completely characterizes the requests to which the rule applies)
- *condition* defines conditions imposed by the rule for the access (and whose satisfaction - or lack of - can imply granting or denying of the requests, depending on the kind of rules)

The action field in a rule simply specifies the name of an operation or of an abstraction thereof (where rules specified on an abstraction apply to all actions in it).

Subjects and objects can also be specified simply by stating an identifier, specifying a given elementary entity in the corresponding domain, or a named abstraction of entities. Moreover, to provide expressiveness and flexibility, our language also allows the specification of subjects and objects through expressions, where each expression identifies a set of subjects (objects, resp.) that satisfy specific properties. Expressions can make use of the reserved identifiers **user**, **project**, **purpose**, **dataset**, and **metadata** to refer to the user, project, purpose, dataset, and metadata, respectively. The use of reserved identifiers gives our language the expressiveness we look for without the need of introducing variables in the authorization language [9]. More precisely, the appearance on one of such identifiers (e.g., **purpose**) in an expression is bounded with the actual parameters of the request (e.g., the purpose specified by the user requesting access) in the evaluation at access control time. Object and subject expressions are characterized as illustrated in the following.

3.1.1 Object expressions

An object expression identifies the set of objects to which the rule applies and has the form:

```
<objexpr>
  <objid id="object-id"/>
  <WITH> <condition> cond-expr </condition> </WITH>
</objexpr>
```

where *object-id* is either the identifier of a dataset (or group of datasets) or the identifier of a metadata document (or group of them) with possibly associated an XPath expression identifying portions of the document; *cond-expr* is a boolean formula of conditions that can evaluate membership of the object in groups or values of properties on metadata. Conditional expressions on metadata make it possible to define access rules applicable only to datasets whose metadata satisfy some conditions. Element `WITH` is optional and can be omitted. As an example, consider the following object expression.

```
<objexpr>
  <objid id="NationalSurvey"/>
  <WITH>
    <condition>
      META(dataset)/producer=StatisticalNationalAgency
    </condition>
  </WITH>
</objexpr>
```

It denotes all datasets in the `NationalSurvey` group that are produced by "Statistical National Agency" (`producer` is a meta-property). Given the richness of the metadata usually supported, this can provides a powerful and convenient way to specify access rules [7]. For instance, it allows the enforcement of embargo restrictions, where only datasets collected before a given year can be released (a restriction that can be imposed via a simple condition on the metadata).

3.1.2 Subject expressions

A subject expression identifies a set of subjects depending on whether they satisfy or not certain conditions. It has the form:

```
<sbjexpr>
  <userid id="user-id"/>
  <OF_PROJECTS id="project-id"/>
  <FOR_PURPOSES id="purpose-id"/>
  <WITH> <condition> cond-expr </condition> </WITH>
</sbjexpr>
```

where *user-id*, *project-id*, and *purpose-id* are the user, project, and purpose identifiers, respectively, and *cond-expr* is a boolean formula of terms that can evaluate the user's profile, or the project's profile. Conditional expressions can make reference to user, project, and purpose involved in the current request by means of the reserved identifiers **user**, **project**, and **purpose**, respectively. Elements `OF_PROJECTS`, `FOR_PURPOSES`, and `WITH` are optional and can be omitted. For instance, subject expression

```
<sbjexpr>
  <userid id="Academic-Community"/>
  <FOR_PURPOSES id="Scientific"/>
  <WITH>
    <condition> project/sponsor=EC </condition>
  </WITH>
</sbjexpr>
```

characterizes users belonging to group `Academic-Community` that intend to use the data for scientific purposes within an EC funded project.

3.1.3 Conditions

Besides subjects, objects, and actions, access control rules can specify *conditions* defining constraints that the rule requires be satisfied for the request to be granted. A condition is simply specified as an element `condition` whose textual content can include two kinds of conditions: *static* and *dynamic*. Static conditions evaluate membership of subjects and objects into classes or properties in their profiles and associated metadata. These are conditions similar to those appearing in subject and object conditional expressions, but which may need to be stated separately (as it will be clear in the next subsection). Dynamic conditions are constraints that can be brought to satisfactions at run-time processing of the request. Dynamic conditions require attachment to procedural calls that execute the necessary actions to bring the condition to satisfaction. Those conditions are usually related to agreement acceptance (which can be as simple as clicking an 'ok' button on a pop up window), payment fulfillment, registration, or form filling and are represented by non-arithmetic predicates, called *dynamic predicates*. Examples of dynamic predicates are illustrated in Figure 4.

3.2 Semantics and format of the rules

Having illustrated the different components of the rules, we now illustate the rule syntax and semantics. Our model supports two different types of access control rules: *authorizations* and *restrictions*. An authorization specifies a permission for the access and is of the form:

```
<authorization>
  <sbjexpr> ...... </sbjexpr>
  <CAN/>
  <action type="action"/>
  <objexpr> ...... </objexpr>
  <IF> <condition> ...... </condition> </IF>
</authorization>
```

where elements `sbjexpr`, `action`, and `objexpr` identify the requests to which the authorization applies, and `IF` is an optional element that includes a boolean expression of conditions (element `condition`) whose satisfaction authorizes the access. Usually, the conditional part of authorizations contains only dynamic conditions (static conditions can be included in the expressions specifying the *subjects* and *object* for the rule). An access request is considered to be authorized if at least one of the authorizations that applies to the request is satisfied.

A restriction specifies requirements that *must* be satisfied for an access to be granted and is of the form:

```
<restriction>
  <sbjexpr> ...... </sbjexpr>
  <CAN/>
  <action type="action"/>
  <objexpr> ...... </objexpr>
  <ONLY_IF> <condition>...</condition> </ONLY_IF>
</restriction>
```

where elements `sbjexpr`, `action`, and `objexpr` identify the requests to which the restriction applies, and `ONLY_IF` is an element that includes a boolean expression of conditions (element `condition`) that every request to which the restriction applies must satisfy. Lack by failure implies that the request will be denied. Element `condition` can contain both static and dynamic conditions. Unlike for authorizations, in fact static conditions cannot be all incorporated in the subject and object expressions of the rules as this would change the semantics of the restrictions. While static conditions appearing in the `condition` element impose constraints that if not satisfied imply that the access

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.
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.
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.

Figure 4. Predicates for dynamic conditions

should be denied, static conditions in the `WITH` element child of element `sbjexpr` (`objexpr`, resp.) simply limit the requests to which the restriction is applicable. As an example, notice the difference between statements like "Users can access data1 *only if* they are non-commercial and sign an agreement" and "Users who are non-commercial can access data1 *only if* they sign an agreement". While the first rule prohibits access to commercial users, the second rule does not.

Intuitively, authorizations correspond to traditional (positive) rules usually enforced in access control systems [11]. If multiple authorizations are applicable to a given access request, the request can be granted only if the conditions in at least one authorization are satisfied. Therefore, lack by failure simply makes the authorization ineffective; but it does not imply that the access will be denied. Intuitively, this means that different authorizations are considered as combined in OR.
The only support of authorizations (traditional open policy) would yield limited results. 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 dataset1 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 are 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. The combined use of authorizations and restrictions easily support both requirements: restrictions specify requirements of the exclusive *only if* form, while authorizations specify requirements in the traditional positive *if* form. Intuitively, restrictions play the same role as negative authorizations (denials) supported by recent access control systems (a restriction is equivalent to a negative authorization where the condition is negated). However, we decided to introduce restrictions as their format appears to be closer to the intuitive formulation of protection requirements in the policies examined. Restrictions are also easier to understand because of the clear separation between subjects to which a restriction applies on the one side and necessary conditions that these subjects must satify on the other side (which, in traditional approaches, would be collapsed into a single field [9]). Figure 5 illustrates some examples of protection requirements and corresponding ACU rules.[2] The first rule states that all users with an `Academic` project can `access Restricted_Datasets` if they have paid for the access. The second rule states that `Academic` users can `access Restricted_Datasets` if they have accepted the Standard Conditions Document. Finally, the third rule states that all European users can `download NationalSurvey` *only if* the surveys are marked as "downloadable".

4 Access control

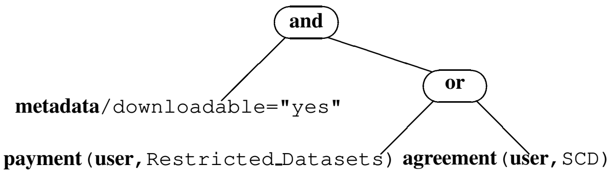
The discussion in the previous section already makes clear how access control works. Given an access request, all authorizations and restrictions applicable to it (i.e., for which the subject, object, and action of the request satisfy the corresponding expressions in the rules) are evaluated and the request is granted if it satisfies the conditions in all such restrictions and the conditions in at least one authorization. Particular care must be reserved in evaluating dynamic conditions: the system should not fail if some (dynamic) conditions are not satisfied but rather prompt them to the user to see whether they can be brought to satisfaction. In this section, we illustrate how dynamic conditions are dealt with. To fix ideas and make the discussion clear, consider the access rules in Figure 5. Consider an access request submitted by a European user, member of the `Academic-Community` group, and with project `Academic`. Given this request,

[2] Namespace prefix declarations have been omitted for clarity.

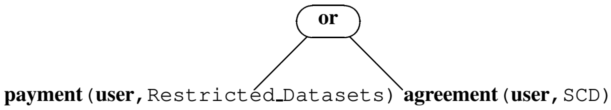
```
<?xml version='1.0'?>
<rules>
  <!-- The user can access any restricted dataset if project is Academic and
    she has paid for the access -->
  <authorization>
    <sbjexpr>
      <userid id="Users"/><OF_PROJECTS id="Academic"/>
    </sbjexpr>
    <CAN/>
    <action type="Access"/>
    <objexpr><objid id="Restricted_Datasets"/></objexpr>
    <IF> <condition> payment(user,Restricted_Datasets)
    </condition> </IF>
  </authorization>
  <!-- Academic users can access datasets if it has been shown the
    Standard Conditions Document -->
  <authorization>
    <sbjexpr><userid id="Academic-Community"/></sbjexpr>
    <CAN/>
    <action type="Access"/>
    <objexpr> <objectid id="Data"/> </objexpr>
    <IF>
      <condition> agreement(user,SCD)</condition>
    </IF>
  </authorization>
  <!-- European citizens can download national surveys only if they are
    marked as "downloadable" -->
  <restriction>
    <sbjexpr>
      <userid id="Users"/>
      <WITH>
        <condition> user/citizenship="EU" </condition>
      </WITH>
    </sbjexpr>
    <CAN/>
    <action type="download"/>
    <objexpr> <objid id="NationalSurvey"/> </objexpr>
    <ONLY_IF>
      <condition>metadata/downloadable="yes"</condition>
    </ONLY_IF>
  </restriction>
</rules>
```

Figure 5. An example of ACU rules

the system collects all the applicable rules, which are all the rules in the figure, and combines their conditions into one global condition C. More precisely, C is the conjunction of all the "only if" conditions (from restrictions) ANDed with the disjunction of all the "if" conditions (from authorizations). In this example C is:



The ACU simplifies C by evaluating its leaves to true or false when possible; dynamic conditions may evaluate to null. Then C is simplified using the usual boolean laws for true and false. In this example, if "**metadata**/downloadable = yes" were false, then C would be simplified to false, and access would be denied; if "**metadata**/downloadable = yes" were true and one of the dynamic predicates were true then C would be simplified to true and the requested file would be immediately downloaded. Next suppose that "**metadata**/downloadable = yes" is *true* while "**payment**(**user**,Restricted_Datasets)" and "**agreement**(**user**,SCD)" are *null*, meaning that the user has not sign the Standard Conditions Document and has not yet paid for restricted surveys. Then C is simplified to:



This condition—called the *residual condition*—cannot be further simplified and cannot be evaluated to true. Therefore, the requested surveys cannot be immediately downloaded. Intuitively, the user may perform the requested operation if the residual condition is satisfied. To guide the user in this process, the access control unit passes the residual condition over to an *Advisor module* (see Section 5) that prompts the user with a dynamic page obtained from the residual condition. For instance, with reference to our example the Advisor can return to the user a message of the form:

To obtain the requested service it is necessary to:

- SIGN AN AGREEMENT **OR**
- PAY for this access.

The underlined words are hyperlinks pointing to further dynamic pages for online agreement and payment. Now the user may choose to give up or follow one or more of the above links to complete the request.

In general, each dynamic predicate is associated with a set of actions (triggered through the advisor as explained above) that may update the profile databases (e.g., by recording that a certain agreement has been signed, so that the same agreement needs not be presented again and again to the user) and/or notify the user or other persons via e-mail messages. The latter are useful for starting manual procedures, when required. After action execution, dynamic predicate evaluation may return true (say, if the payment procedure concludes successfully) or false (e.g., if a manual payment procedure is needed; in this case the payment action can start the payment process by prompting the user with all the instructions and notifying the administration via e-mail).

5 Specifications and Architectural Design

The model illustrated in this paper has been implemented as the ACU component in the framework of a complete Web-based three-tier application for controlled dissemination of information as a result of the EU funded FASTER project (`www.faster-data.org`).

The sequence diagram in Figure 6 illustrates the basic software modules that provide a coarse-grained internal decomposition of our ACU. The ACU wraps up

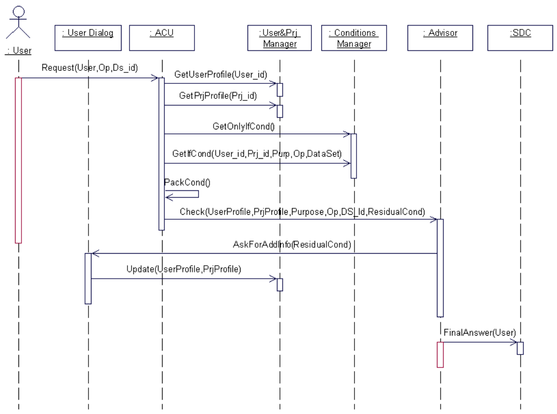


Figure 6. Internal operations of the ACU

entirely the computation of access permissions to individual datasets, returning, for each request, the decision of whether the access should be granted or denied. Internally, the main steps are the following.

Request For convenience, we model the front-end reaction to a dataset request as the creation of a transient ACU Main object. ACU Main is nothing but a dispatcher that handles all communications required to check permissions.

User Profile and Access Rule Identification The ACU Main module interacts with the *User & Project Manager* and *Condition Manager*, both of which are *data mediators*, encapsulating the information needed to identify the applicable access rules (i.e., the user/groups and conditions/policies databases, respectively). A fast access read-only memory structure linking subjects to authorizations is used for speed.

Final Decision Computation and Notification After getting from the mediators the available data about the access rules that apply to the requester, the ACU computes the final decision granting or negating access. 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. Otherwise, delegation to the *Advisor* module controlling dialog with the user is triggered.

User Dialogue The dialogue is controlled by the Advisor module according to the access policy in force. If some of the specified conditions can be fulfilled, for example, by signing an agreement or by authenticating the user, then the Advisor uses the *User Dialog* module to prompt the requester with the actions that would result in the required access. The User Dialog module encapsulates all the user interface resources (bitmaps, buttons, windows, and so on) required during user interaction. Once the needed information has been collected, the user profile is updated and a final access decision is determined. If access is to be granted (the user has executed all the required actions) control is passed to the external *Statistical Disclosure Control* (SDC), which will return to the requester the properly sanitized view of the data.[3]

6 Conclusions

We have presented the architecture of the *Access Control Unit* (ACU), an autonomous software component offering interfaces for controlling access to data archives in the framework of Web-based information services. The software design of our component makes it suitable for server-side integration in a variety of Web-based architectures for information dissemination. A prototype implementation based on a Java servlet is currently available. Our prototype runs under Windows NT/IIS4.0 with the servlet engine `JRUN`. Future work includes extending the approach to the consideration of digital certificates.

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Academic Press/Morgan Kaufmann, 1999.

[2] P. Bonatti, E. Damiani, S. De Capitani di Vimercati, and P. Samarati. An access control system for data archives. In *Proc. of the 16th International Conference on Information Security*, Paris, France, June 2001.

[3] P. Bonatti, S. De Capitani di Vimercati, and P. Samarati. A modular approach to composing access control policies. In *Proc. of Seventh ACM Computer and Communication Security*, pages 164–173, Athens, Greece, November 2000.

[4] P. Bonatti and P. Samarati. Regulating service access and information release on the web. In

[3]This model is needed for statistical datasets representing surveys containing possible sensitive information that must therefore be sanitized to prevent inference and record linkage attacks [14].

Proc. of the Seventh ACM Conference on Computer and Communications Security, pages 134–143, Athens, Greece, 2000.

[5] S. Castano, M.G. Fugini, G. Martella, and P. Samarati. *Database Security*. Addison-Wesley, 1995.

[6] *Communications of the ACM*, volume 41, April 1998.

[7] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. *Computer Networks*, 33(1–6):59–75, June 2000.

[8] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security*, 2002. To appear.

[9] S. Jajodia, P. Samarati, M.L. Sapino, and V.S. Subrahmanian. Flexible supporting for multiple access control policies. *ACM Transactions on Database Systems*, 2000. To appear.

[10] M. Kudo and S. Hada. XML Document Security based on Provisional Authorization. In *Proc. of the Seventh ACM Conference on Computer and Communication Security*, pages 87–96, November 2000.

[11] P. Samarati and S. Jajodia. Data security. In J.G. Webster, editor, *Wiley Encyclopedia of Electrical and Electronics Engineering*. John Wiley & Sons, February 1999.

[12] The data documentation initiative codebook DTD - version 1.0, March 2000. http://www.icpsr.umich.edu/DDI/CODEBOOK.TXT.

[13] B. Thuraisingham, S. Jajodia, P. Samarati, J. Dobson, and M. Olivier. Privacy issues in www and data mining: Panel discussion. In S. Jajodia, editor, *Database Security XII - Status and Prospects*. Kluwer, 1999.

[14] L. Willenborg and T. de Waal. *Statistical Disclosure Control Practice*. Springer Verlag, 1996.

[15] M. Winslett, N. Ching, V. Jones, and I. Slepchin. Using digital credentials on the World-Wide Web. *Journal of Computer Security*, 5(3):255–267, 1997.

[16] World Wide Web Consortium (W3C). *XML Path Language (XPath) Version 1.0*, November 1999. http://www.w3.org/TR/xpath.