Authorization in XML Store

Morten Bartvig Peter Theill

January 12, 2006

Vejleder:
Fritz Henglein

Abstract

This thesis proposes a method for implementing authorization in XML Store. It is shown that access control can be implemented using XML Store's value-oriented model without affecting the XML Store desiderata.

Further extension of access control filters is supported.

Update operations have been shown to be problematic and are not implemented in the prototype.

Contents

1 Introduction

This section introduces a proposed approach towards applying authorization to XML Store.

1.1 XML Store

XML Store[6] is a peer-to-peer mobile persistence layer using a value-oriented approach for retrieving and storing semi-structured data. Value-oriented means that stored data is immutable, i.e. will never change once stored. Whenever changes are made, new nodes are constructed and stored, and the old nodes are left unchanged. The resulting new node is a combination of old nodes and new nodes. Instead of storing duplicates of the old nodes the new node contains references to these existing nodes. This means that nodes with the exact same content are shared. It also means that nodes can't have names in the same way as files in a filesystem. Instead a name server is used to maintain a list of names and references to nodes. When a user changes a node and wants this node to have the old name, the reference in the name server is changed so it points to the new node.

This thesis uses the Koala XML Store implementation by Thomas Ambus based on thesis written by Kasper Bøgebjerg Pedersen and Jesper Tejlgaard Pedersen[12] as well as the XPath for XML Store[13] implementation by same author.

The current implementation of Koala XML Store is available at:
`http://www.plan-x.org/xmlstore`

The current implementaion of XPath for XML Store is available at:
`http://www.plan-x.org/projects/xpath`

This thesis requires basic knowledge of XML and XPath.

1.2 Motivation

XML Store is meant to be used as a distributed multi-user system. This makes authorization necessary, as in any other multi-user system, where permissions have to be separated and private data has to be kept private.

There are many different ways of applying access control. Some methods only processes the queries to the data source, other methods iterate through the nodes, and yet other methods use a combination of methods. The access control can be designed in several ways. One way is to let nodes keep information about rules, other methods use access control lists in different ways, and other methods provide a generic and comprehensive language for defining such rules.

Authorization in XML Store is interesting because XML is used as data type, which can make authorization more fine-grained for an XML document as opposed to a file in a filesystem, where the file is either accessible or inaccessible.

1.3 Problem statement

The goal for this thesis is to find an answer to the question:

> *Is it possible to define authorization in XML Store using its own value-oriented programming model and implement it without affecting central properties[12, section 1.1.1, page 7]?*

Throughout analysis and design these eight properties are considered and commented in detail. For reference these are listed here:

1. Decentralized
2. Distributed persistence
3. Efficient and transparent sharing of XML documents
4. Convenient and adequate API
5. Hide location and distribution
6. Lazy loading
7. No parsing and unparsing of XML documents
8. Configurable

1.4 Limitations

Some aspects are beyond scope of this report and not considered.

Authentication No authentication will be considered in this report, hence all peers are implicitly assumed to be "trusted".

Efficiency of algorithms No analysis and efficiency of algorithms are considered.

1.5 Conclusion

This thesis proposes a method for implementing authorization for XML Store. The primary goal is on implementing a prototype which does not violate any of the desiderata set forth by XML Store while allowing access control for both load and save operations. A subgoal is to suggest a way to provide fine-grained access control on XML documents.

A prototype has been built which does not violate any of the desiderata. The prototype uses the approach to store all authorization data as nodes in XML Store. This prototype is capable of loading nodes based on subjects' privileges. Update of nodes has been found more difficult than expected and is not implemented, and further research and analysis on this topic has to be conducted.

Fine-grained access control has been made possible using a set of XPath expressions as rules either allowing or denying access to node-sets.

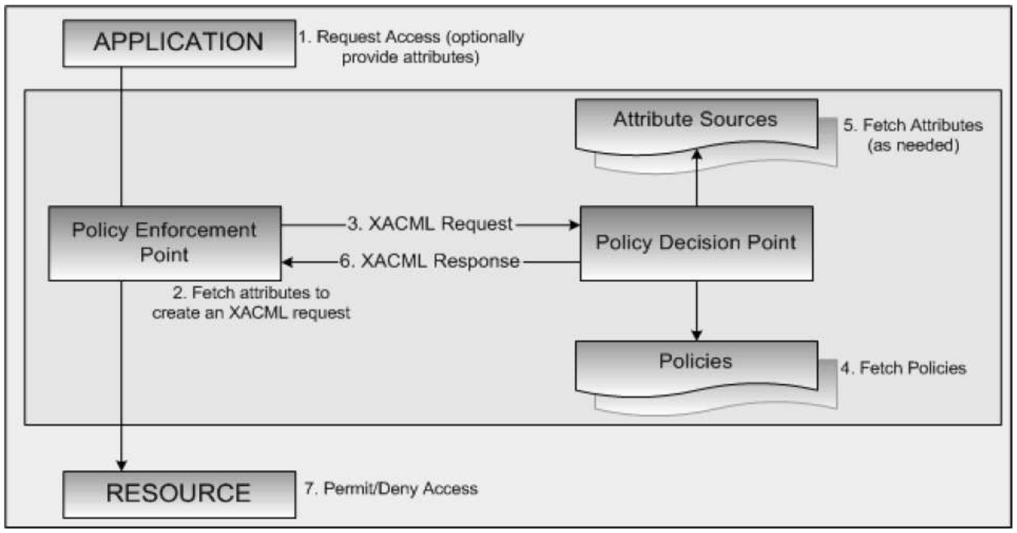

Figure 1: Overview of XACML[8, figure 1]

2 Analysis

The intention of this analysis is to consider some approaches for implementing authorization in XML Store and analyse whether or not they comply with central properties of XML Store. The goal is to find a method that exploits the existing functionality of XML Store without violating central properties.

Existing standards that could be a basis for authorization in XML Store will be evaluated.

2.1 Standards for authorization

There has been some attempts to make standards related to authorization, which are based on XML. This section analyses some of these standards.

2.1.1 XACML

XACML (eXtensible Access Control Markup Language) is intended to be used as a generic access control enforcer for distributed systems, which can be used to control access to many different kinds of data and to interoperate between different authorization components. “XACML is a standard, general purpose access control policy language defined using XML”[11].

An overview of XACML is shown in figure 1.

1. Requester requests access

2. PEP fetches attributes

3. PEP requests PDP (optionally using SAML)
4. PDP fetches attributes
5. PDP fetches policies
6. PDP sends response back to PEP
7. PEP responds with an answer ("Permit", "Deny")

Abbreviations used are explained in the following list. Several more terms and abbreviations are used in XACML, but this thesis does not go into a detailed discussion about XACML and therefore the description here is a simplification of XACML.

Attribute in XACML terms is a characteristic of a subject, resource, action or environment. A subject is the entity that makes the request, i.e. user or agent.

PEP Policy Enforcement Point, the point that communicates between the requester and PDP.

PDP Policy Decision Point, the point that decides whether or not to grant the requested access.

XACML either permits or denies a request. This limits the possibilities of what kind of requests can be made. In an environment with different authorization components across different domains this is quite acceptable, because a request could be "Open front door in building B" or "Show me document X", and XACML should only decide whether or not this request should be granted and not how to actually perform the requested action. A request such as "Show me all documents I can access" can not be answered with a simple yes or no.

XACML is designed to authorize access to different types of data. There is no restriction of what kind of data this is. This is partly what makes XACML so generic. XML Store is, however, based purely on XML and therefore does not need the extra capabilities for applying access control to different types of data.

XACML is designed as a generic language for defining access control policies and to interoperate between different authorization components and across domains. Domains in this context mean separate and autonomous administrative domains. XML Store is based on one specific data type and thus does not need different authorization components or a complicated generic language for defining access control rules.

On the other hand, the ability to interoperate between XML Store and other systems is desirable if there is a need to combine authorization for different authorization domains.

2.1.2 SAML

SAML (Security Assertion Markup Language)[2] is an XML standard for exchanging authentication and authorization data between security domains. SAML can be used with XACML to exchange data between a PDP and a PEP. SAML does not in itself provide a security mechanism but is used to facilitate communication, so it will not be considered an authorization solution.

2.2 Cryptography

Because XML Store is a decentralized and distributed peer-to-peer system, data can reside on any peer which is connected to the XML Store network. This means that when a user's data is stored on another peer there is no way of knowing whether the peer is controlled by a malicious user or not. Encrypting the private nodes is thus a necessity.

One method of cryptography is to encrypt the data channel between a user and a server. This will ensure that users that are able to listen to the data communication can't intercept information. This solution works very well if the XML Store is used on trusted peers but in an untrusted environment. This means that the peers are trusted but the data channel is not trusted.

When a user only stores data that are either public available, i.e. is not needed to be encrypted or private, i.e. only the owner of the document has access to it, only an owner needs to know the encryption keys. In such a setting the owner only needs to use one key for encrypting and decrypting, i.e. a symmetric key encryption algorithm.

If a user wants to store data that uses a more advanced and fine grained access control scheme, another approach is needed. One such approach is given in the following section.

Cryptography, not being a central point of focus for this thesis, will not be discussed in detail and different cryptographic algorithms will not be discussed.

2.2.1 Encryption of nodes

This section describes a cryptographic method which ensures encryption of nodes which will be accessible by several subjects[1]. The advantage of this method is that a node does not have to be encrypted n times, if n different subjects are going to have access to this node.

Figure 2 is an example of how to protect a document with different encryption keys. Node 1 states that the subject can gain access to `<hosp>`, if he has decryption key K_1. $(K_1 \wedge K_3) \vee K_4$ means that the subject can access the node `<nurse>`, if he has either both keys K_1 and K_3 or key K_4. A node with "true" instead of a key name means that the node is not encrypted.

The equivalent normalized tree protection for figure 2 is depicted in figure 3. Normalized means that all nodes are guarded by an atomic formula, i.e. either `true`, `false` or a decryption key K_i. As can be seen on the figure, a subject needs encryption key K_6 to gain access to node `<nurse>`. K_6 can be retrieved in nodes 11 and 12. Node 11 is protected

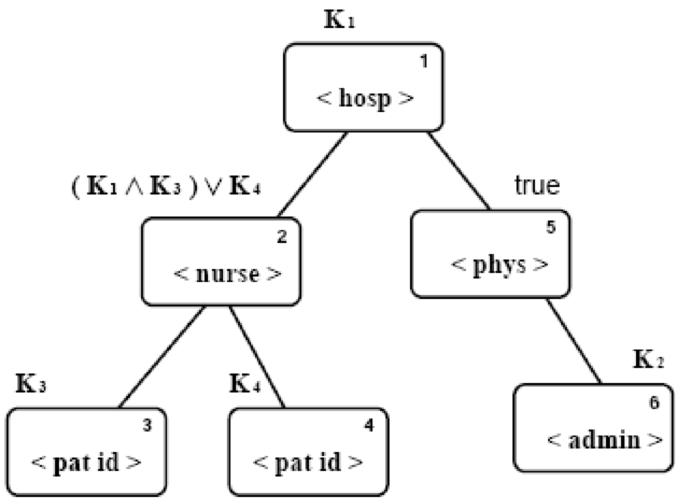

Figure 2: An example of an encrypted XML document[1, figure 1]

with K_5 which can be retrieved from nodes 9 and 10, which are protected with K_1 and K_3, etc. As an example a user with decryption keys K_1 and K_3 can get access to the "real" nodes 1, 2, 3, and 5.

Encrypting nodes in this way makes it possible to define different access to different users without duplicating the nodes and encrypt them with each user's public key. The owner of the document can make keys available to different users elsewhere, for instance some kind of container which is encrypted with the corresponding user's public key, so that only that specific user can access the keys in the container. See figure 4 for an example of how these containers could be constructed. User A can decrypt `Container A` with the private key to get keys K_1, K_2 and K_3. These keys can be used to decrypt nodes 1, 9, 10, 11[1], 2, 3 and 6 in figure 3.

When this kind of encryption is used, the XML Store property of sharing will be violated. If some subject encrypts a node, and this node exists somewhere else in the XML Store, the two nodes will be different, because one of the nodes is encrypted. It's possible but unlikely, though, that there will exist many nodes with the exact same content that is encrypted with different keys. On the other hand, the technique makes it possible for a subject to encrypt a node only once with one key and make it accessible to a number of subjects. All these subjects can then retrieve the correct decryption key.

Cryptography doesn't violate the other properties of XML Store as encryption and decryption is decentralized.

[1]Access to node 11 from key K_5 is derived from nodes 9 and 10

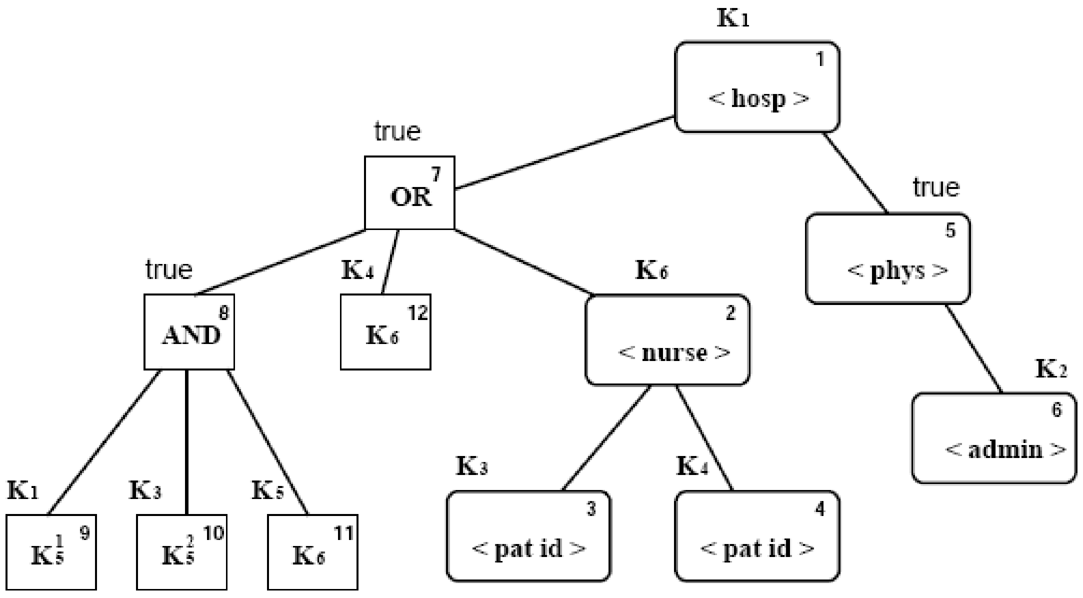

Figure 3: A normalization of the tree protection in figure 2[1, figure 1].

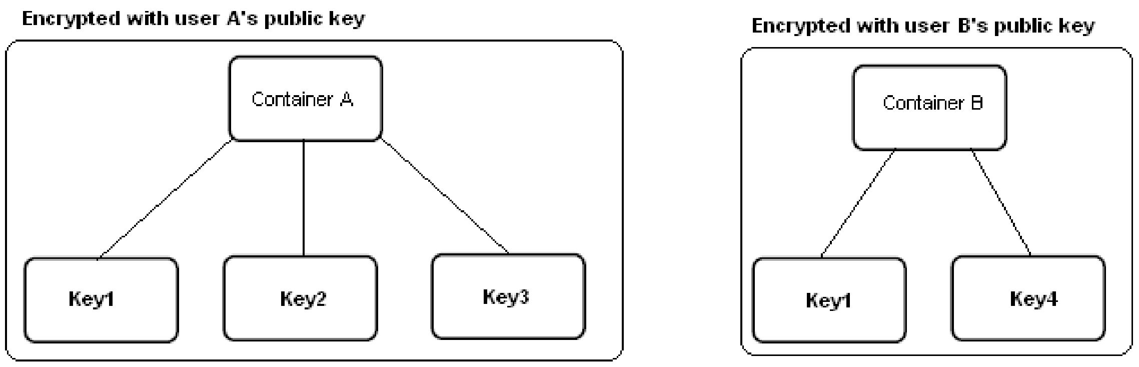
Encrypted with user A's public key
Container A
Key1
Key2
Key3
Encrypted with user B's public key
Container B
Key1
Key4

Figure 4: Example of encrypting encryption keys

2.3 Types of access control

Two types of access control schemas are considered in order to figure out which one best suits this thesis' needs. Dicretionary access control, referred to as DAC and Mandatory access control, referred to as MAC.

2.3.1 Discretionary access control

Discretionary access control is an access control policy defined by owners of a given resource. An owner of a resource is often the subject who created it. For each resource, owners decide which subjects are allowed to access that resource and what privileges they have, so two concepts are that it's "owner-based" and access is controlled by "individual rights".

Access control lists (ACL) and role-based access control (RBAC) are techniques used for applying DAC. ACLs name specific rights assigned to a subject for a given resource e.g. `read access to resource` M to user X. Role-based defines access by assigning functional roles to subjects greatly simplifying access rights management e.g. a role `Reader` having `read access to resource` M. Following users X, Y and Z may be assigned to role `Reader` now getting `read access to resource` M.

2.3.2 Mandatory access control

Mandatory access control is an access control policy defined by container system. For each resource a level of sensitivity is associated and by comparing resource levels with subject levels, access is controlled. So two concepts are "system-controlled" and access is controlled by "sensitivity levels".

Rule-based access control and Lattice-based access control are techniques used for applying MAC. Rule-based specifies rules such as "nodes with a `Name` subnode" which defines access not specifically related to a subject. Lattice-based specifies access by different levels such as "confidential" or "secret".

2.4 Existing authorization systems

Designing a useful authorization system requires a usable configuration of access control. Evaluating existing authorization systems helps in determining which method is best suited for this. The goal is to figure out how to define access control using XML structures.

2.4.1 Filesystems

Access control in a filesystem is primarily based on objects being either a file or a folder. Users might have either access or no access to an object. Based on a folder's tree-structured hierarchy, it's possible to define fine-grained control i.e. allowing access to file A in subfolder B for user or group Y. This fine-grained control is very much similar to an XML structure where a folder hierarchy might be represented as shown in 5. Allowing limited access to file `.profile` but full access to file `cv.html` is a common scenario.

```
<folders>
  <folder name="home">
    <folder name="pt">
      <file name=".profile" />
      <file name="cv.html" />
    </folder>
  </folder>
</folders>
```

Listing 5: A folder structure represented as XML

Configuring access control in filesystems is, however, different depending on which type of filesystem it is. Many filesystems use attributes on files to reflect access control. In *NIX an object has an owner and a group. An owner is a reference to a user and a group is a reference to a set of users. Basic attributes are allowed to be set for each object and are a combination of read, write and execute. In Windows a role-based approach is used for configuring access control. Each object isn't explicitly being tagged with attributes but are instead associated with a role having specific permissions to access that object. Configuring access control is then a matter of assigning proper roles to users.

Thus access control in XML requires a way to pinpoint a specific node e.g. `.profile` in above example. For this, a query language such as XPath or XQuery is needed.

2.4.2 Databases

Database management systems or DBMSs have been front-runners for RBAC so these systems usually use same type of access control. Access control configuration is different across DBMSs as is the case for filesystems but in this case it is only a matter of syntax and not access control type.

In databases an object is one of many things e.g. either a table, a view, a stored procedure or a user-defined function. To ease configuration a similar approach towards each object type is used, hence configuring a role to a table is typically identical to configuring a role to a view, etc. In relational databases a tree-structured hierarchy isn't easily put on top of these objects, thus relating access control in databases with access control applied to XML isn't a perfect match.

2.4.3 Web servers

Configuring access control in web servers is a matter of either allowing or denying a client to request a given resource. A resource in web server terminology may be many things but is often a web page, an image, or a file. In order to control access to these resources an access control list such as example 6 is configured. This denies access to files named `.profile` and allows all users to view resources in folder `/home/pt/public`.

```
<Files ~ "^\.profile">
  Order allow,deny
  Deny from all
</Files>
```

```
<Directory /home/pt/public>
  AllowOverride All
  Options ExecCGI
  Order allow,deny
  Allow from all
</Directory>
```

Listing 6: An example of an Apache .htaccess file

Caused by the nature of web servers, configuration of access control is request-based e.g. a request is initially determined and based on this its privileges are computed. For web servers, configuration of access control can be determined by e.g. folders, files and URIs with further ability to override defined access control in subfolders.

Representing access control from web servers in XML requires a way to override a “base” security setup, hence needing a way to represent a base access “template” with an ability to override its “scope”.

2.5 Query filtering

QFilter[9] is a method for filtering queries. The idea behind QFilter is to produce an XPath expression which is safe to evaluate given the defined access control rules, which are defined as XPath expressions. That the produced expression is safe to evaluate means that when evaluated using an XPath implementation the read nodes will be nodes the user has access to according to the rules. This also means that the produced expression can be evaluated against any XML data source which uses XPath.

There are defined two answer models, and these are “answer-as-nodes” and “answer-as-subtrees”. “Answer-as-nodes” means that the returned answer is only the projection nodes themselves. “Answer-as-subtrees” includes the descendants. This means that an answer in the “answer-as-nodes” model will not include any descendants, i.e. they will be pruned out. In the “answer-as-subtrees” model only the denied descendants will be pruned out.

Post-processing of the output nodes is necessary, if one wishes to use the “answer-as-subtree” model. This is described in section 2.5.3.

Write operations are not covered, so only read operations are considered in the analysis of QFilter.

There are two main approaches to QFilter, the primitive approach and the Nondeterministic Finite Automaton approach (aka. Nondeterministic Finite State Machine). A third approach is described by [9] but this approach being a filtering which iterates through the nodes of the document puts it in another category covered in section 2.6.

2.5.1 Primitive approach

Consider the following mathematical expression

$$Q \cap ((a_1 \cup a_2 \cup \cdots \cup a_n) \setminus (d_1 \cup d_2 \cup \cdots \cup d_m))$$

$a_1, \dots a_n$ are the sets of the document which the subject is allowed to access, $d_1, \dots d_m$ are the sets the subject is not allowed to access, and Q is the queried piece of the document. Simplifying with $A = a_1 \cup \dots \cup a_n$ and $D = d_1 \cup \dots \cup d_m$ and rewriting the expression[2]:

$$Q \cap (A \backslash D) = (Q \cap A) \backslash D$$

This expression takes the queried part of the document, joins it with the accessible part of the document and removes the inaccessible parts of the document. This is the basic idea of the primitive QFilter approach.

Instead of evaluating all these parts of the document and then combining them to produce a resulting set, the mathematical expression can be translated into an XPath expression where the evaluation of the XPath expression is executed in the end. That is, the XPath expressions are being combined to a single expression before the actual XPath evaluation. Translating the mathematical expression to an XPath expression, we get:

```
(Q INTERSECT A) DIFFERENCE D
```

`INTERSECT` and `DIFFERENCE` are not actual XPath functions in XPath version 1.0 and will later be rewritten to actual XPath expressions. `A` and `D` are constructed as joining respectively the allow expressions and the deny expressions: `A = a1 | a2 | ... | aN`, where `aN` is the last allow expression. `D` is constructed equivalently. What the XPath expression really means is "an XPath expression which, when evaluated, will produce a node-set of the requested document which is an intersection of the queried part and the allowed part, and lastly the denied parts removed".

Because the node-sets aren't sets mathematically speaking, it's important to distinguish between the "answer-as-nodes" and "answer-as-subtrees" models. See 2.5.3 for further elaboration on this.

The major advantage of the resulting XPath expression is that it's constructed from the access control rules and the query alone. This means that only the resulting allowed part of the document has to be loaded, i.e. the property of *lazy loading* of XML Store is achieved. And the resulting XPath expression can be evaluated against any XML data source that is capable of processing XPath expressions. Thus, little implementation is necessary to apply authorization.

In this section the primitive approach is a little simplified compared to [9], in which rules with incompatible projection nodes compared to the query are ignored if they are of local type, and appended to the query if they are of recursive type. A local type only specifies one level of nodes, ie. no descendants. A recursive type specifies all descendants too. It's safe enough to join a rule, which has incompatible projection nodes with the query, because M joined with $\emptyset$ (the empty set) is still M, so this simplification only has impact on the length of the resulting safe XPath expression.

[2]This rewrite has an impact on the length of the constructed XPath expression later.

```
/Contacts/Contact/Name//*
/Contacts/Contact/EmailAddress/Email
```

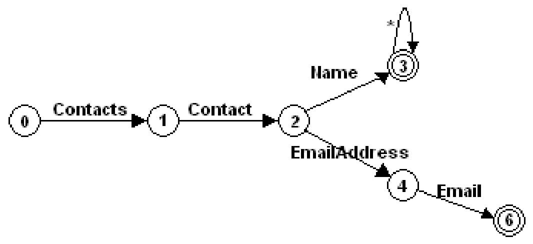

Figure 7: From ACR to NFA

2.5.2 NFA-based approach

This approach uses NFA (Nondeterministic Finite Automata) to rewrite the query expression. The NFA states are built upon the steps from the control access rules. Each transition from one NFA state to another are constructed from four basis steps. These are “`/x`”, “`/*`”, “`//x`”, and “`//*`”. The steps from the access control rules are converted to NFA transitions as shown in figure 7.

Detailed discussion of NFA is beyond scope of this thesis.

Deterministic transitions These are the `/x`-transitions, that is, with no wildcards in either the NFA or in the query.

Non-deterministic transitions These are when the query step is a `/x` but there are more than one outgoing transition or if there is a transition with a `//x` or `//*` step. All possible transitions from the current state are followed and if a path ends in an accepted state, the query is accepted.

Rewriting when query contains `*` When the query step is a `*`, all the transitions from the current state has to be considered. This means that a new query is constructed from all the outgoing transitions. If one of the transitions is a `*`, then the current query step is kept. Otherwise the accepted queries are joined.

Rewriting when query contains `//` “`//`” means this point and every ancestor below this point. So actually `//x` means that the rewritten query should contain every access control rule that has a state below this point and ends with a transition `x` to an accepting state. `//*` is the same, except it doesn’t matter what the name of the final transition is.

“`/*`” and “`//*`” are nondeterministic and can therefore give problems if the query has special properties. For instance “`/foo/bar/preceding-sibling::`” can potentially bypass access control rules if such XPath functions aren’t dealt with properly. For instance, to handle the XPath function `parent::` properly, the execution of the query can go back one step in the NFA. Functions such as `preceding-sibling::` or `child::` are somewhat more difficult to solve because they require knowledge of the document itself.

Either these problematic XPath functions should not be allowed, or the query should be modified according to the access control rules and the document, i.e. not only the rules. See figure 8 for an example of bypassing an access control rule. The rules are as follows: include: `Contact/Name/*`, exclude: `Contact/Name/Surname`, and the query is `Contact/Name/GivenName/next-sibling::.`. The returned node is bold in the figure.

```
<Contact Id="pt">
  <Name>
    <Title>Mr.</Title>
    <GivenName>Peter</GivenName>
    <SurName>Theill</SurName>
  </Name>
</Contact>
```

Listing 8: Bypassing of access control rules

Predicates are handled without being evaluated. A predicate is a filter in an XPath expression, e.g. in the XPath expression `/Contact[@Id="mb"]/EmailAddress` the predicate is `[@Id="mb"]`. The NFA can be extended to include predicates by constructing "pseudo"-states or extending the transitions to include predicates. Whenever a state (or transition) in the NFA contains a predicate, this predicate is inserted into the query string at the proper place. This way predicates don't need to be treated in a special way because the evaluation of the predicates themselves are postponed to the XPath implementation.

2.5.3 Pruning subnodes

When the safe XPath expression is used to fetch data, the data returned is only safe in regards to the projection nodes. Consider figure 9 with include rule `R1: /Contact/*`, exclude rule `R2: /Contact/Name/SurName` and query `Q: /Contact/Name`.

```
<Contact Id="mb">
  <Name>
    <GivenName>Morten</GivenName>
    <SurName>Bartvig</SurName>
  </Name>
  <EmailAddress>
    <Email>bartvig@gmail.com</Email>
  </EmailAddress>
</Contact>
```

Listing 9: Accepted node containing denied subnodes

`R1` states that nodes `Name` and `EmailAddress` are accessible, and `R2` states that node `SurName` is not accessible. This means that when the query `Q` is evaluated, `SurName` should not be returned to a requester. However, the result is that the node `Name` (including subnodes) will be returned. In [9]'s terms, a "bite" operator "Q bitten-by R2" is needed to prune out denied subnodes of `R2` from accepted nodes of `Q`. "Q bitten-by R2" means that the sub nodes of `Q` which `R2` denies, should be pruned out. According to [9, section 3.1] there is not yet such a method of "biting" descendants in the subtree.

There is, however, another method that can be used to make sure that an accepted subtree that contains denied nodes will not be returned. For every deny rule, there can be constructed a rule which says that if any part of the selected tree contains a descendant which is denied, this selected tree should be denied. In XPath format this could be an append of the deny rule with `/ancestor-or-self::*`. In the example in listing 9 the exclude expression would be changed to "Exclude: `/Contact/Name/SurName/ancestor-or-self::*`", and there will be no returned subtree. Alternatively all descendants of every node in the resulting nodeset can be pruned out. This will conform to the "answer-as-nodes" model.

Denying an entire subtree because a subnode is denied makes it impossible to get a list of allowed nodes. If such a list is needed, a specific list method can be implemented in XML Store. Another possibility is to implement a filtering method which filters out the denied nodes of a document. This topic will be dealt with in section 2.6.

2.5.4 QFilter in XML Store

QFilter modifies XPath queries and doesn't change the way XML Store saves and loads nodes. The downside of QFilter is that it can't prune out subnodes from an XML tree, so either one has to accept that the returned node can contain nodes, which should be inaccessible, implement a method to prune out these denied nodes, or a tree with illegal subnodes should not be returned. However, if the latter is chosen the structure of the document is not broken.

QFilter is a useful approach, since it conforms to all the desiderata of XML Store. It's also a fast approach[9, section 5] since only pieces of a document have to be processed after the safe query has been constructed. The NFA approach is even better than the simple one, because not all access control rules might apply to the query, and this makes the resulting constructed safe query shorter (and even the execution of the NFA faster, since not all transitions will be followed). If a query is either accepted as it is, or denied completely (this only applies to the NFA approach) the execution of the XPath becomes faster. In the case of an accepted query, the expression will be shorter than when it's rewritten, and consequently the XPath execution is faster. In the case of a denied query, there will not be executed any XPath expression because the query is denied as a whole.

2.6 Post filtering

Post filtering is a method that processes the document after a possible pre-processing of the query (such as QFilter). The downside of post filtering is that the access control rules have to be evaluated directly on the document, and hence the XML Store property of *lazy loading* can not be achieved.

A method of pruning out denied nodes of a document to compute a requester's view of a document is described in [4]. In short the method labels the nodes in the document, according to the access control rules, with either `deny` or `allow` or `not applicable` (ϵ), which can later be converted to `allow` or `deny` dependendent on the other rules. After the labeling, denied nodes are pruned out. The requester's XPath expression will be evaluated on this computed view of the original document. The focus in [4] is on web-based documents and the document viewed will be the entire computed view. Accordingly no XPath expression will be evaluated on the computed view, but it would be possible to do so. If one wishes to be able to evaluate queries according to the original document, the method can be extended to label the nodes according to the query and thereafter only label those nodes according to the rules and finally prune out denied nodes and nodes that aren't allowed, including nodes that aren't queried. Another option is to insert nodes to the computed view, so the original structure is kept, and then evaluate the XPath expression on this.

Write operations (`insert`, `delete` and `update`) are supported in [4]. The access control rules are defined in the same way as for read operations, and write operations are dealt with very similar to read operations. Insert operations are evaluated by executing the labeling process on the new document with the new node inserted. If the labeling process produces an `allow` label on the new node, the insert operation is accepted, otherwise it's rejected. Delete operations are accepted, if the labeling process of the document labels the chosen node as `allow`. Update operations are evaluated by labelling both the original document and the updated one. If the node being updated has been labeled `allow` in both the original and the updated document, the update operation is accepted. All write operations imply almost same amount of processing of the document, and thus *lazy loading* is not achieved in write operations.

Although *lazy loading* is not achieved, the other XML Store properties are kept. Contrary to QFilter this method computes the requester's view and write operations are dealt with. This can make it a better choice than QFilter, if these properties are more important than *lazy loading*.

2.7 Conclusion of analysis

The analysis' goal has been to evaluate some standards for authorization, evaluate properties of existing authorization implementations and to evaluate some specific methods used to control access to XML documents.

XACML and SAML are standards in the topics of authentication and authorization, and are thus obvious candidates for this thesis. SAML, being a standard for exchanging authentication and authorization data, offers no authorization and were evaluated because of its close relation and cooperation with XACML. XACML, being a framework for tying together large-scale authorization systems and being a general purpose policy system, offers a possible approach for authorization. However, the policy language is complex and advanced, because it's meant as a generic access control markup language, and XML Store would only need a subset of the language's features, and is therefore not going to be the central part of authorization. In the future, though, XML Store can still be extended to use XACML and SAML.

QFilter seems an obvious choice because it's a simple approach, it conforms to all the desiderata of XML Store, and it's faster than the post filtering approach[9, section 5], which also doesn't comform to the property of *lazy loading*.

Instead of limiting authorization to use only one authorization method, this thesis will make it possible to extend authorization to use other methods, being QFilter, post filtering, XACML, or a completely different method without having to know the inner workings of XML Store. One particular method will be used as a prototype. For futher information about the choice of prototype, see section 4.1.

Cryptography will not be used in this thesis, but section 6.1 offers a short description of how XML Store can be extended to include this.

3 Design

The primary goal for this thesis is to define authorization in XML Store without affecting its desideratas as described in [12]. To reach this goal each desideratum is examined and evaluated to find the most suitable solution for applying authorization.

3.1 Evaluating XML Store desiderata

The eight XML Store desiderata are considered and commented below. The desiderata are highly desirable properties for an implemented XML Store and violating just a single of these properties may prove the store invaluable.

Decentralized An authorized XML Store must remain decentralized meaning extra layers or components should be kept to a minimum, if at all. One way to support this would be to ensure that any new layer or component is indeed decentralized by itself. XML Store is separated into layers ranging from an application layer down to an operating system layer. Implementing authorization on as high a layer as possible eases this decentralized desideratum, i.e. if it's possible to load and save information of authorization in the application layer it would fulfil this desideratum.

Distributed persistence Persisting information about authorization such as an access control list, a list of roles or maybe a list of allowed or denied peers must be distributed, i.e. it must consider general requirements such as transparency, scalability, efficiency, replication, consistency, security and fault tolerance [3, p. 315-316]. In order to ensure this desideratum, authorization information must be kept distributed which must be explicitly kept in mind in case a new layer or component is added. This desideratum is fulfilled by design if information of authorization is applied at the application layer, but must explicitly be considered for layers below.

Efficient and transparent sharing of XML documents An XML Store implementation stores nodes using a value-oriented "share-create" approach i.e. only one copy of a given value is stored. Authorization must keep this in mind i.e. adding or modifying nodes or attributes on stored documents should be avoided in general unless these can be applied in such a way that sharing across documents with different authorization schemas is maintained. However this would violate desideratum "*No parsing and unparsing of XML documents*" described below. Separating authorization from "content" documents seems to be a more desirable approach which wouldn't interfere with this desideratum.

Convenient and adequate API Configuring access control must use existing APIs already available for accessing data in XML Stores, i.e. information of authorization needs to be represented as XML data and existing methods for loading and saving these XML nodes should be used to retrieve or persist authorization. Extensions to API may be considered from a convenience perspective but is not a requirement for successful configuration of authorization.

Hide location and distribution Application programmers should not treat authorization configuration differently with regards to their stored location such as in-memory or on-disk.

Lazy loading Authorization must not require XML documents to be fully loaded before access to a document can be determined. In order to keep this desideratum, an authorization implementation must consider several aspects. First, documents should not be loaded if no matching role or authorization schema was found. Secondly, a method which can be applied on an XML document without requiring full node retrieval must be implemented. A method capable of this is QFilter[3].

No parsing and unparsing of XML documents Any XML document stored in XML Store must be saved in its raw format without a need for parsing and unparsing any part of the document. An authorization system must in other words not "tag" documents or nodes stored in XML Store in order to apply authorization since it would require a need to modify the XML document.

Configurable XML Store is configurable and in order to fulfill this property that fact must be kept in mind. Thus authorization must be applied using a decorator pattern extending functionality of XML Store without limiting the possibility of it being further extended.

Two additional goals are set forth to improve adoption of a developed authorization system.

Decorating existing XML Stores Providing authorization as a decorator pattern extends the usability, because any XML Store conforming to the API can be used as a decorated pattern in authorization. This goal is in good line with XML Store properties concerning *No parsing and unparsing of XML documents* and *Configurable*.

[3]QFilter is analyzed in section 2.5

Backward compatibility No requirements must be set forth for XML Stores choosing to apply authorization i.e. any existing XML Stores such as LocalXMLStore, DistributedXMLStore or CompactXMLStore may be used directly. This goal is achived by storing authorization information when data is saved to XML Store if not already available and thus allow large existing stores to progressively incorporate authorization on documents.

3.2 Access control based on RBAC

Since MAC does not allow subjects to entirely determine its own access control for stored XML documents, it is restricting XML Store usage which is not a desired property because XML Store is a decentralized system. However, MAC gives an administrator the ability to set a "base" access control for all resources stored by a subject, and thus does not require a subject to explicitly define an access policy for each stored resource. This avoids subjects accidentally granting other subjects more privileges than they are supposed to have which is a desired property.

With DAC, document owners need to define access policies for stored resources but does not put any restrictions on XML Store usage. Furhermore DAC is a commonly used type of access control and simplifies access rights management, which is a desired property taking into account that it should be defined by document owners. Providing fine-grained control of semi-structured data does not further complicate this procedure.

Role-based access control seems to be a better choice for authorization than MAC since it allows XML documents to be stored in XML Store by subjects each controlling their own level of access to other subjects.

3.3 Security model

Selecting a proper security model for the purpose requires a knowledge of other existing security implementations.

Various security models have already been discussed and evaluated in 2.4 and based on these as well as ideas from Microsoft .NET My Services[10] a security model for XML Store is suggested below. Focus is based on a model for sharing, XML document owners must explicitly define

- Who to share information with
- What information to share
- How information is shared

A subject initially storing an instance of a document in XML Store, i.e. the user doing a `bind` operation on an authorization-enabled name server (from this point known as AuthNameServer) is referred to as the creator of the document.

The goal is to facilitate sharing of information that owners want to share with other subjects. This is not a simple goal, though, since a complex set of inputs will result in owners misconfiguring security and in effect shares more information than intended, shares the wrong information, etc. Many combinations resulting in incorrect sharing exist so a set of implementation level goals must further be considered:

- Sharing on a coarse grained level so subjects might share information of similar types with a higher degree of confidence based on an access template.
- Different views on documents based on what a given subject is allowed to see, so one subject might see it one way and another subject in a different way. An example could be sharing an entire `Contact` with subjects in a `Friends` group including `GivenName`, `SurName` and `Email` but only sharing `GivenName` and `SurName` with subjects of another (or no) group.

As stated previously it is necessary to know *who*, *what* and *how* to share information. A model for role-based authorization is used to ease security administration for document owners.

Who A subject is encapsulated in a `Role` and specifies an identification of a subject which might be any authenticated entity (e.g. an agent). A role is a list of subjects that have the same permissions. A role could be named “reader” or “editor”. A subject can be member of many roles but only one role per document instance i.e. a given subject may be allowed to access multiple documents stored in XML Store but each document maintains their own access control configuration. This configuration is maintained in a `RoleList` and `RoleMap` object and will be discussed later.

What The scope of the document, for which the `Role` applies, is encapsulated in a `Scope` object including or excluding node-sets based on XPath expressions.

How Types of data access such as “querying” or “inserting” nodes are encapsulated in a `RoleTemplateMethod` object and identify what operations are allowed. In a value-oriented programming model a need for “replace” and “delete” operations are superfluous albeit these may further ease access control configuration if owners need to express methods such as “delete only nodes from a document created by caller”. In this thesis, however, only “query” and “insert” are considered.

The terms `Scope`, `RoleMap`, `RoleList`, `t` and `nil` has been taken from .NET My Services. Several other ideas have not been used as they contradict with the desiderata of XML Store.

Authorization is based on different structures listed in details below. Each structure has a 1:1 mapping with a class. XML schemas exist for these structures and can be found in appendix B but are otherwise not used.

3.3.1 Scope

Scopes are used to define which node-sets are accessible for each subject. A scope may define a node-set that includes all public email addresses for a given document, information created by authorized subject, only surnames of contacts, etc. See listing 10 for scope definition examples where first scope allows everything (full set), second example allows only node sets created by owner, and third example allows all email address nodes marked as being "Public".

```
<scope base="t" name="all" />

<scope base="nil" name="creator">
  <shape type="include" select="//*[@Id='$callerId']" />
</scope>

<scope base="nil" name="public-emails">
  <shape type="include" select="//EmailAddress[Category='Public']" />
</scope>
```

Listing 10: Examples of scopes

Scopes contain shapes each defining a set based on a type and an XPath expression. Node-sets are computed by using scope base combined with list of shapes. A scope base is either `t` or `nil` indicating everything or nothing as recognized from e.g. LISP[7].

3.3.2 RoleTemplate

Role templates define how information is to be shared for a given document or instances of same document type. A role template may define type of access allowed for an authorized subject such as a `reader` indicating it's possible to read but not write information. Examples of role template definitions are shown in listing 11.

```
<roleTemplate name="owner">
  <roleTemplateMethod type="query" scopeRef="all" />
  <roleTemplateMethod type="insert" scopeRef="all" />
</roleTemplate>

<roleTemplate name="editor">
  <roleTemplateMethod type="query" scopeRef="all" />
  <roleTemplateMethod type="insert" scopeRef="creator" />
</roleTemplate>

<roleTemplate name="reader">
  <roleTemplateMethod type="query" scopeRef="all" />
</roleTemplate>
```

Listing 11: Examples of role templates

Each role template contains a name describing its purpose and a set of methods and scopes defining what and how information is allowed to be accessed. Only two methods exists for querying and inserting data, named "query" and "insert" respectively. The value-oriented nature of XML Store, which should be kept when maintaining security, eliminate a need for "replace" and "delete" methods as mentioned earlier.

3.3.3 Role

Roles are used to define which subjects have access rights to the document.

```
<role roleTemplateRef="owner" scopeRef="all">
  <subject userId="pt" />
</role>

<role roleTemplateRef="reader">
  <subject userId="mb" />
</role>
```

Listing 12: Examples of roles

A subject is identified by a user id and can be included in one role. A role contains a required `roleTemplateRef` attribute indicating its related `RoleTemplate` object, which determines base allowed access. An optional `scopeRef` attribute refers to a scope which may further restrict access using access defined in role template as a base.

3.3.4 RoleList

A role list defines which users have access rights to a document by maintaing a list of role elements as shown in listing 13. A full example of this document is available in A.2 and its associated XML Schema can be found in B.1.

The purpose of scope elements is to provide more fine-grained per-user data control over scopes stored on a role template. A given scope associated with a role is optional and will be combined with the scope assigned to the subject in the `RoleMap` (which is defined in the next section).

```
<roleList>

  <!-- list of scopes as described above -->
  <scope base="t" name="all" />

  <!-- list of roles as described above -->
  <role roleTemplateRef="owner" scopeRef="all">
    <subject userId="pt" />
  </role>

  <role roleTemplateRef="reader" scopeRef="all">
    <subject userId="mb" />
  </role>

</roleList>
```

Listing 13: Example of a role list

The role list is an important part of computing authorization. A role element must exist in this list that maps a calling user to a role template. If no matching role element is found, an authorization fault is generated. Once a role element is located, the referenced role template is located. A subject can only have one role associated.

3.3.5 RoleMap

A role map defines the allowable methods, and what scope of data is accessible while using this method, as shown in listing 14. Only one instance of a role map exists for each stored document and is authored by the owner of the document. A full example of this document is available in A.3 and its associated XML Schema can be found in B.2.

```
<roleMap>

  <!-- list of scopes as described above -->
  <scope base="t" name="all" />

  <scope base="nil" name="creator">
    <shape type="include" select="//*[@CreatedBy='$callerId']" />
  </scope>

  <!-- list of role templates as described above -->
  <roleTemplate name="owner">
    <roleTemplateMethod type="query" scopeRef="all" />
    <roleTemplateMethod type="insert" scopeRef="all" />
  </roleTemplate>

  <roleTemplate name="reader">
    <roleTemplateMethod type="query" scopeRef="all" />
  </roleTemplate>

</roleMap>
```

Listing 14: Example of a role map

The reason for defining a role map is to simplify access control configuration as it appears for users of documents. This is done by first specifying a fixed set of access patterns, named role templates, that occur in a given document type. Now document owners only need to decide which user maps to which role template.

Allowing distinction on “current user” is useful in a security environment. A variable is introduced called `$callerId` which is initialized to id of subject at runtime, meaning a role map author is able to express shapes such as “Any nodes owned by caller”.

3.4 Storing authorization data in XML Store

The proposed approach to store all access control details in XML Store requires a discussion of what, where and how to store it.

3.4.1 Saving data

Saving a node in XML Store returns a reference. This reference can be bound to a name via the name server using either `bind` (for a new document) or `rebind` (for an existing document). In the value-oriented model an existing document means, that the modified document is saved in the XML Store without deleting the old nodes.

Three authorization nodes, creator, role map, and role list, and one content node are stored when executing a `save` method and all these references are bound to names in the name server when executing `bind`.

One role map exists for a given stored document since it's stored on a per-user basis. It would be possible to combine a role list and a role map into one document and associate this with the actually stored document but it has been decided to separate this into two different documents. The reasoning behind this is based on the idea that a given stored document can have a specific type such as "Emails", "Contacts", "Notes", etc. If each type of document is used by several subjects it would be possible to define a common set of access control rules for a given type. As an example it would be possible to define a role allowing subjects to see all contacts having a specific company name. This can be expressed using an XPath expression such as `//Contact[CompanyName='IBM']`. Then, when an owner is going to define access control for a document of the same type, only defining who is a member of that role is needed.

3.5 Applying authorization to XML Store

The following sections describes how authorization can be applied to XML Store.

3.5.1 Loading data

A reference is needed to load data from XML Store. This reference is looked up by a name via a name server. Access control is performed during load operations only, hence no checks are performed when looking up a reference from the name server. Three references related to authorization and one reference related to content are looked up in the decorated name server.

Once a reference has been returned from a call to `lookup` it can be used in a call to `load`. The `load` method uses additional authorization references to load access control lists from XML Store if available. Since XML Store supports a single load method taking a reference and returning a node, this must be extended for fine-grained control. An overloaded `load` method taking not just a reference but also an XPath expression is added since access control otherwise would be limited to only selecting a root node.

Nodes stored in XML Store are not distinguished by type, e.g. it is not possible to ask XML Store to fetch all nodes of type `RoleMap`. A scheme for loading a specific type such as a RoleMap or a RoleList is however necessary, i.e. it must be possible to find an associated access control configuration for a given stored XML document. A way to ensure this is to use known bind names when binding or rebinding nodes which makes it possible to look up a user's role list by doing a lookup on bind name "`urn:authx:`α `RoleList`", where α is the actual bind name set by the subject.

4 Implementation

In this section the implemented authorization prototype is described and thoughts behind it are discussed.

Binaries and Java documentation may be found at the AuthX project web site:
`http://www.plan-x.org/projects/authx`

4.1 AuthXMLStore

Class `AuthXMLStore` is a decorated[5, p. 175] XML Store.

Implementing interface `XMLStore<R>` Any given XML Store must implement interface `XMLStore<R>` shown in listing 15 which is implemented by `AuthXMLStore` as well.

```
public interface XMLStore<R extends Reference> {
  public R save(Node node) throws IOException;
  public Node load(R ref) throws IOException,
    UnknownReferenceException;
  public NameServer<R> getNameServer();
  public void close() throws IOException;
}
```

Listing 15: Interface for implementing XML Stores

An overloaded `load` method is added to support explicit querying of sub nodes based on an XPath expression. See listing 16. This method is required to support a fine-grained access control scheme.

```
public Node[] load(AuthReference ref, String query) throws IOException,
  UnknownReferenceException;
```

Listing 16: Signature for additional `load` method

Having only a `load(R ref)` method as specified by `XMLStore<R>` interface would lead to a problem with a QFiltering approach considering its "answer-as-nodes" nature. With this additional method it is possible to evaluate an XPath expression on a node loaded from `AuthReference` and return a node-set after it has been processed by a filtering method.

Implementing interface Filter Filtering methods implements interface shown in listing 17.

```
public interface Filter {
  Node[] evaluate(
    Node contextNode,
    RoleTemplateMethodType methodType,
    String query,
    Role role
    ) throws IOException;
}
```

Listing 17: Interface for implementing filtering

The authorization prototype includes a filtering method based on QFilter which is implemented in `QFilter` class. This implementation uses the simple QFilter approach.

Access control rules are treated as recursive rules. E.g. `/Contacts/Contact` refers to the `Contact` nodes and all their descendants.

4.2 AuthReference

`AuthReference` is a container for references `Creator`, `RoleMap`, `RoleList`, and `Content` which are the references used in the decorated name server. The reason for using a single container reference is to maintain the XML Store interfaces, and that it's simpler for the application programmer only to consider one reference instead of four. Besides, an application programmer should not be able to construct an `AuthReference` from arbitrary references as this can tamper with authorization.

4.3 AuthNameServer

The method `getNameServer()` returns an instance of AuthNameServer which is a decorated name server implementing the `NameServer<R>` interface. This name server uses references of type `AuthReference`.

When performing a `lookup` or `bind` operation on this name server four operations on the decorated name server are actually being executed. Three operations are related to looking up or binding access control nodes and one for "content" node. Transactions are not implemented in the prototype implementation. This will result in an incorrect state in case some of the four operations fail. This is an issue which should be addressed in a final implementation either by extending core name server capabilities to support transactions or by handling this directly in `AuthNameServer`. When performing a `rebind` operation, only one operation on the decorated name server is executed, since only the content document has been changed.

Authorization is not applied in the name server component, thus any subject may look up an AuthReference from a bind name. Access control is performed during load.

An overloaded `rebind` method is added to support rebinding of a specific authorization document such as a role map or role list. See listing 18.

```
public void rebind(String name, AuthReference oldRef, AuthReference newRef,
  AuthReferenceDocumentType documentType) throws IOException, NameServerException;
```

Listing 18: Signature for overloaded `rebind` method

Bind names As described, each `bind` operation does three extra binds to XML Store. Names for these additional binds are prefixed with `urn:authx:` though no specific name conventions exist for XML Store bind names.

4.4 Container classes

The container classes together implement the `RoleMap` and `RoleList` documents. They all have a `toXml` method that serializes to XML string format.

4.5 Limitations

The following limitations apply to the implemented prototype.

Invalid access control rules The `AuthXMLStore` implementation does not take into account illegal or invalid authorization. That is, an invalid `RoleMap` or `RoleList` document will not be checked for errors. E.g. if a `RoleList` does not have any `Role` nodes, or if a `Subject` is contained in multiple `Role` nodes no proper error message will be generated.

Transactions There has not been implemented any form of transaction support. This can be a problem in cases, where a series of actions have to be either completed fully or not completed at all. For instance, when `AuthNameServer` executes four requests to the decorated name server, and some error prevents the third request from being completed, there will be inconsistency in the authorization data.

Unsecure `AuthReference` The `AuthReference` has not been implemented securely, as a trusted environment is assumed. `AuthReference` can potentially be constructed from arbitrary references. If document X is inaccessible for user A, and user A creates a `RoleMap` and `RoleList` with full access to A and constructs an `AuthReference` from these and the original `Content` and `Creator` for document X unauthorized access can be gained.

4.6 Corrections to "XPath for XML Store"

Two issues are discovered with the XPath implementation available at the time of writing. These issues have been addressed and are commented below.

Missing `count()` function An implementation of QFilter requires the `count` function[4] which returns the number of unique nodes in the argument node-set. This function is not implemented in the existing XPath implementation so a new class has been developed and can be found in appendix C.1.1.

Bug in `union` operator The `XNodeSet` class is implemented as a `java.util.List` in the existing XPath implementation. This has been done for performance reasons as well as for convenience[13, p. 31] but unfortunately its implementation contains a bug which allows `Node` objects, which are identical, to appear more than once. This causes a problem in the qfiltering method since its use of `count` needs to return a number of unique nodes. Corrected source code can be found in appendix C.2.1.

[4]See http://www.w3.org/TR/xpath#function-count

5 Evaluation

Authorization has been implemented using XML Store's value-oriented programming model and without affecting central properties.

Authorization data is fully stored in XML Store, which makes authorization distributed and decentralized. Thus, the *decentralized* property is kept in `AuthXMLStore`.

Update of a document is not supported acceptably. Any subject can rebind any name to a new document. This means that in order for authorization to be usable, a proper update mechanism has to be implemented in `AuthXMLStore`.

The problems with update operations are due to the fact, that access control for `insert` can not be evaluated during a rebind operation, as the name server can't load the authorization data from XML Store. It seems as if update operations have to be implemented through methods in `AuthXMLStore` (on the same level as `load` and `save`) instead of through `rebind` in `AuthNameServer`.

The prototype is designed to handle filter extensions using a filtering interface. It is thus easy to extend `AuthXMLStore` with further filtering functionality.

6 Future works

Some features are not fully implemented or could be extended in the future to provide better or new support for various properties. This section describes such future works.

6.1 Cryptography

A useful extension to AuthXMLStore is cryptography. This section offers a proposal of a possible cryptographic XML Store implementation based on section 2.2. It should not be considered a thoroughly analyzed and designed proposal but a possible approach to extending XML Store with cryptography.

A cryptography extension could be implemented with decorator patterns, so that the cryptographic store makes use of AuthXMLStore features.

A public key could be associated with an instance of the cryptographic store. This key would be used throughout XML Store to encrypt private data, sign encrypted data and decrypt further decryption keys for decryption of accessible nodes.

Whenever a node is encrypted with a key, there should be some form of identificaton of what decryption key is to be used, so that a user can get this decryption key from his container of decryption keys associated with the current document.

When a document is to be encrypted with different keys it should be as transparent as possible for the application programmer, how the nodes are to be encrypted. The cryptographic extension should do as much as possible, and the application programmer should only define as little as possible. AuthXMLStore already offers authorization of documents. These access control rules can be used to find out which roles are associated with what nodes, and hence which nodes should be encrypted with the same keys and which should be encrypted with other keys so a user can't gain access to more nodes than intended. This would indeed make encryption transparent and the cryptographic store could be used in exactly the same way as AuthXMLStore, except that the public and private keys have to be explicitly generated.

A cryptographic implementation should also consider signing nodes, especially the nodes `RoleMap`, `RoleList` so that it's clear which content document they are assigned to. This also prevents tampering of the `AuthReference`.

6.2 Filtering using XSLT

Filtering of a document with regards to the access control rules could be done with XSLT. If an XSLT processor is available for XML Store the filtering process currently implemented could be replaced with a proper XSL transformation.

6.3 Updating documents

Update operations aren't fully supported in `AuthXMLStore`. Some basic operations like `load` and `save` in `AuthXMLStore` could be implemented as `insert`, `delete` and `replace`. These extra methods would then make sure that a document's name would be rebound properly, if a subnode of a document was to be modified by a subject not owning the document.

6.4 Name server overhead

`AuthNameServer` executes four name server requests on the decorated name server for each `bind` and `lookup`. This is not very optimal. The nameserver could be extended to support transactions, so that the four requests could be executed as one, atomic operation.

Alternatively, `AuthXMLStore` could be modified so that only one nameserver request would be necessary. Instead of constructing four documents for each XML document to be stored in XML Store, the structure of the document could be changed to include authorization documents as shown in listing 19. This would, however, prevent the XML Store on disk to be used directly in another XML Store, e.g. LocalXMLStore or DistributedXMLStore, which is in contradiction with the goal "*backward compatibility*", see 3.1).

```
<authx:document>
  <authx:creator>
    <!-- creator document-->
  </authx:creator>
  <authx:roleMap>
    <!-- roleMap document -->
  </authx:roleMap>
  <authx:roleList>
    <!-- roleList document -->
  </authx:roleList>
  <authx:content>
    <!-- content document -->
  </authx:content>
</authx:document>
```

Listing 19: Examples of role map and role list included in a single document

7 References

[1] Martín Abadi and Bogdan Warinschi. Security analysis of cryptographically controlled access to XML documents. In *Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 108–117, 2005. ISBN 1-59593-062-0.

[2] Scott Cantor, John Kemp, Rob Philpott, and Eve Maler. Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0, March 2005. `http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf`.

[3] George Coulouris, Jens Dollimore, and Tim Kindberg. *Distributed Systems Concepts and Design*. Addison Wesley, 2001.

[4] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, and Pierangela Samarati. A fine-grained access control system for XML documents. *ACM Transactions on Information and System Security (TISSEC)*, Volume 5(Issue 2):169–202, May 2002.

[5] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns Elements of Resuable Object-Oriented Software*. Addison Wesley, 1999.

[6] Koala XML Store. `http://www.plan-x.org/xmlstore/`.

[7] ANSI and GNU Common Lisp Document. `http://www.cs.queensu.ca/software_docs/gnudev/gcl-ansi/gcl_toc.html`.

[8] Markus Lorch, Seth Proctor, Rebekah Lepro, Dennis Kafura, and Sumit Shah. First experiences using XACML for access control in distributed systems. In *Proceedings of the 2003 ACM workshop on XML security*, pages 25–37, 2003.

[9] Bo Luo, Dongwon Lee, Wang-Chien Lee, and Peng Liu. QFilter: Fine-Grained Run-Time XML Access Control via NFA-based Query Rewriting. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 543–552, 2004. ISBN 1-58113-874-1.

[10] Microsoft. *Microsoft .NET My Services Specification*. Microsoft Press, 2001.

[11] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0, February 2005. `http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf`.

[12] Kasper Bøgebjerg Pedersen and Jesper Tejlgaard Pedersen. Value-oriented XML Store. Master's thesis, ITU, DTU, November 2002. Supervisors: Michael R. Hansen, Fritz Henglein.

[13] XPath for XML Store. `http://www.plan-x.org/projects/xpath/`.

A Examples of documents

Appendix including examples of XML documents used and referred to in report.

A.1 contacts.xml

The listing below is a full example of “contacts” document used throughout this report.

```
<Contacts>
  <Contact Id="pt">
    <Name>
      <Title>Mr.</Title>
      <GivenName>Peter</GivenName>
      <SurName>Theill</SurName>
    </Name>
    <EmailAddress>
      <Category>Private</Category>
      <Email>peter@theill.com</Email>
      <Name>Personal E-mail</Name>
    </EmailAddress>
    <EmailAddress>
      <Category>Public</Category>
      <Email>pt@commanigy.com</Email>
      <Name>Business E-mail</Name>
    </EmailAddress>
    <WebSite>
      <Category>Personal</Category>
      <Url>http://www.theill.com/</Url>
    </WebSite>
    <TelephoneNumber>
      <Category>Mobile</Category>
      <Number>+45 61 71 50 96</Number>
    </TelephoneNumber>
    <TelephoneNumber>
      <Category>Home</Category>
      <Number>+45 59 44 50 96</Number>
    </TelephoneNumber>
    <TelephoneNumber>
      <Category>Business</Category>
      <Number>+45 59 44 50 96</Number>
    </TelephoneNumber>
  </Contact>
  <Contact Id="mb">
    <Name>
      <GivenName>Morten</GivenName>
      <SurName>Bartvig</SurName>
    </Name>
    <EmailAddress>
      <Email>bartvig@gmail.com</Email>
      <Name>Personal E-mail</Name>
    </EmailAddress>
    <WebSite />
    <TelephoneNumber>
      <Category>Mobile</Category>
      <Number>+45 28 49 33 90</Number>
    </TelephoneNumber>
    <TelephoneNumber>
      <Category>Home</Category>
      <Number>+45 43 40 57 52</Number>
    </TelephoneNumber>
    <TelephoneNumber>
      <Category>Business</Category>
      <Number>+45 35 87 88 19</Number>
    </TelephoneNumber>
  </Contact>
</Contacts>
```

A.2 roleList.xml

The listing below is a full example of "role list" document used throughout this report.

```
<roleList>

  <scope base="t" name="all" />

  <scope base="nil" name="creator">
    <shape type="include" select="//*[@Id='$callerId']" />
  </scope>

  <scope base="nil" name="public-emails">
    <shape type="include" select="//EmailAddress[Category='Public']" />
  </scope>

  <role roleTemplateRef="owner" scopeRef="all">
    <subject userId="pt" />
  </role>

  <role roleTemplateRef="reader" scopeRef="creator">
    <subject userId="mb" />
  </role>

  <role roleTemplateRef="anonymous">
    <subject userId="anonymous" />
  </role>

</roleList>
```

A.3 roleMap.xml

The listing below is a full example of "role map" document used throughout this report.

```
<roleMap>

  <scope base="t" name="all" />

  <scope base="nil" name="creator">
    <shape type="include" select="//*[@Id='$callerId']" />
  </scope>

  <scope base="nil" name="friends">
    <shape type="include" select="//Contact[Category='Public']" />
  </scope>

  <scope base="nil" name="coworkers">
    <shape type="include" select="//Contact[Category='Public']" />
  </scope>

  <roleTemplate name="owner">
    <roleTemplateMethod type="query" scopeRef="all" />
    <roleTemplateMethod type="insert" scopeRef="all" />
  </roleTemplate>
  <roleTemplate name="editor">
    <roleTemplateMethod type="query" scopeRef="all" />
    <roleTemplateMethod type="insert" scopeRef="creator" />
  </roleTemplate>

  <roleTemplate name="reader">
    <roleTemplateMethod type="query" scopeRef="all" />
  </roleTemplate>

</roleMap>
```

B Schemas for documents

Appendix including developed XML Schemas for authorization in XML Store.

B.1 roleList.xsd

The listing below shows XML Schema for “role list” document.

```
   <?xml version="1.0" encoding="utf-8"?>
<xsd:schema id="roleList" xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation>
      Schema for role list in AuthX project.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="roleList">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="scope" type="scopeType" />
        <xsd:element name="role" type="roleType" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="scopeType">
    <xsd:sequence>
      <xsd:element name="shape" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="type" type="xsd:string" use="required" />
          <xsd:attribute name="select" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="base" type="xsd:string" use="required" />
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>

  <xsd:complexType name="roleType">
    <xsd:sequence>
      <xsd:element name="subject" minOccurs="1" maxOccurs="1">
        <xsd:complexType>
          <xsd:attribute name="userId" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="roleTemplateRef" type="xsd:string" use="required" />
    <xsd:attribute name="scopeRef" type="xsd:string" />
  </xsd:complexType>

</xsd:schema>
```

B.2 roleMap.xsd

The listing below shows XML Schema for “role map” document.

```
   <?xml version="1.0" encoding="utf-8"?>
<xsd:schema id="roleMap"  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:documentation>
      Schema for role map in AuthX project.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="roleMap">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="scope" type="scopeType" />
        <xsd:element name="roleTemplate" type="roleTemplateType" />
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>

  <xsd:complexType name="scopeType">
    <xsd:sequence>
      <xsd:element name="shape" minOccurs="0" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="type" type="xsd:string" use="required" />
          <xsd:attribute name="select" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="base" type="xsd:string" use="required" />
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>

  <xsd:complexType name="roleTemplateType">
    <xsd:sequence>
      <xsd:element name="roleTemplateMethod" minOccurs="1" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:attribute name="type" type="xsd:string" use="required" />
          <xsd:attribute name="scopeRef" type="xsd:string" use="required" />
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="name" type="xsd:string" use="required" />
  </xsd:complexType>

</xsd:schema>
```

C XPath for XML Store corrected source code

A couple of issues were identified in the “XPath for XML Store” implementation. This appendix contains added and corrected source code classes.

C.1 org.planx.xpath.function

C.1.1 CountFunction.java

```java
package org.planx.xpath.function;

import org.planx.xpath.Context;
import org.planx.xpath.Environment;
import org.planx.xpath.Navigator;
import org.planx.xpath.object.*;

/**
 * Core count function.
 **/
public class CountFunction<N> implements Function<N> {
    public XObject<N> evaluate(XObject<N>[] args, Context<N> ctxt, Environment<N> env,
                                  Navigator<N> nav) throws FunctionException {
        if (args.length == 1) {
            return new XNumber<N>((double) ((XNodeSet)args[0]).size());
        } else {
            throw new FunctionException("Illegal number of arguments");
        }
    }
}
```

C.2 org.planx.xpath.expr.operator

C.2.1 UnionOperator.java

```java
package org.planx.xpath.expr.operator;

import org.planx.xpath.Navigator;
import org.planx.xpath.XPathException;
import org.planx.xpath.expr.Expression;
import org.planx.xpath.object.*;

/**
 * A union between two expressions returning node sets.
 **/
public class UnionOperator<N> extends Operator<N> {
    public UnionOperator(Expression<N> e1, Expression<N> e2) {
        super(e1, e2);
    }

    protected XObject<N> evaluate(XObject<N> o1, XObject<N> o2, Navigator<N> navigator)
                                                   throws XPathException {
        XNodeSet<N> set1 = null, set2 = null;
        try {
            set1 = (XNodeSet<N>) o1;
            set2 = (XNodeSet<N>) o2;
            //set1.addAll(set2);
            for (N a : set2) {
              if (!set1.contains(a)) {
                set1.add(a);
              }
            }
            return set1;

        } catch (ClassCastException e) {
            String err = "Expression did not evaluate to an XNodeSet: ";
            if (set1 == null) throw new XPathException(err+e1);
            if (set2 == null) throw new XPathException(err+e2);
            throw e; // should never happen
        }
    }

    protected String operatorName() {
        return "|";
    }
}
```

D Source code

Appendix including developed source code grouped into packages and sorted alphabetically by filename.

D.1 org.planx.authx

D.1.1 Role.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import org.planx.xmlstore.Node;

/**
 * A role contains information about which subject can access which data.
 *
 * @author pt
 */
public class Role {
    private RoleTemplate roleTemplate;

    private Scope scope;

    private Subject subject;

    /**
     * @return Returns the roleTemplate.
     */
    public RoleTemplate getRoleTemplate() {
        return roleTemplate;
    }

    /**
     * @param scope
     *            The scope to set.
     */
    public void setScope(Scope scope) {
        this.scope = scope;
    }

    /**
     * @return Returns the scope.
     */
    public Scope getScope() {
        return scope;
    }

    /**
```

```
     * @return Returns the subject.
     */
    public Subject getSubject() {
        return subject;
    }

    /**
     * @param roleTemplate
     * @param subject
     */
    public Role(RoleTemplate roleTemplate, Subject subject) {
        this.roleTemplate = roleTemplate;
        this.subject = subject;
    }

    /**
     * @return Object serialized as Xml.
     */
    public String toXml() {
        StringBuffer xml = new StringBuffer();
        xml.append(String.format("<role roleTemplateRef=\"%s\"",
                new Object[] { this.roleTemplate.getName() }));
        if (this.scope != null) {
            xml.append(String.format(" scopeRef=\"%s\"",
                    new Object[] { this.scope.getName() }));
        }
        xml.append(">");
        xml.append(this.subject.toXml());
        xml.append("</role>");
        return xml.toString();
    }

    /**
     * Deserialize object from specified node.
     *
     * @param n Node with Role object
     * @param roleMap RoleMap instance used by deserialization process.
     * @return Role object with deserialized object.
     */
    public static Role toObject(Node n, RoleMap roleMap) {
        Role a = new Role(roleMap.getRoleTemplateByRef(n
                .getAttribute("roleTemplateRef")), Subject.toObject(n
                .getChildren().get(0)));

        String scopeRef = n.getAttribute("scopeRef");
        if (scopeRef != null && !scopeRef.equals("")) {
            a.setScope(roleMap.getScopeByRef(scopeRef));
        }

        return a;
    }
}
```

D.1.2 RoleList.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.util.ArrayList;
import java.util.List;

```

```java
import org.planx.xmlstore.Node;

/**
 * A role list contains information about which roles are available for a
 * given stored document in XML Store.
 *
 * @author pt
 */
public class RoleList {
    private List<Scope> scopes;

    private List<Role> roles;

    /**
     * @param scopes
     *            The scopes to set.
     */
    public void setScopes(List<Scope> scopes) {
        this.scopes = scopes;
    }

    /**
     * @return Returns the scopes.
     */
    public List<Scope> getScopes() {
        return scopes;
    }

    /**
     * @param roles
     *            The roles to set.
     */
    public void setRoles(List<Role> roles) {
        this.roles = roles;
    }

    public Role getRoleByUserId(String userId) {
        if (roles == null || roles.size() == 0 || userId == null
                || userId.equals("")) {
            return null;
        }

        for (Role a : roles) {
            if (userId.equals(a.getSubject().getUserId())) {
                // only -one- Role should be available for each User Id
                return a;
            }
        }

        return null;
    }

    /**
     * @return Returns the roles.
     */
    public List<Role> getRoles() {
        return roles;
    }

    /**
     * Serializes object into Xml string.
     *
     * @return String with object serialized as Xml.
     */
    public String toXml() {
        StringBuffer xml = new StringBuffer();
        xml.append("<roleList>");

        if (scopes != null) {
            for (Scope s : scopes) {
                xml.append(s.toXml());
            }
        }

        if (roles != null) {
            for (Role r : roles) {
                xml.append(r.toXml());
            }
        }

        xml.append("</roleList>");
        return xml.toString();
    }

    /**
     * Deserializes object from node.
     *
     * @param roleListNode Node to use in deserialization process.
     * @return RoleList object deserialized from node.
     */
    public static RoleList toObject(Node roleListNode, RoleMap roleMap) {
        List<Scope> scopes = new ArrayList<Scope>();
        List<Role> roles = new ArrayList<Role>();

        for (Node a : roleListNode.getChildren()) {
            if ("scope".equals(a.getNodeValue())) {
                scopes.add(Scope.toObject(a));
            }
            else if ("role".equals(a.getNodeValue())) {
                roles.add(Role.toObject(a, roleMap));
            }
        }

        RoleList roleList = new RoleList();
        roleList.setScopes(scopes);
        roleList.setRoles(roles);

        return roleList;
    }

}
```

D.1.3 RoleMap.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.util.ArrayList;
import java.util.List;

import org.planx.xmlstore.Node;

/**
 * A role map contains information about scope of role templates for a given
 * stored document in an authorized XML Store.
 *
 * @author pt
 */
public class RoleMap {
    private List<Scope> scopes;

    private List<RoleTemplate> roleTemplates;

    /**
     * @param scopes
     *            The scopes to set.
     */
    public void setScopes(List<Scope> scopes) {
        this.scopes = scopes;
    }

    /**
     * @return Returns the scopes.
     */
    public List<Scope> getScopes() {
        return scopes;
    }

    /**
     * @param roleTemplates
     *            The roleTemplates to set.
     */
    public void setRoleTemplates(List<RoleTemplate> roleTemplates) {
        this.roleTemplates = roleTemplates;
    }

    /**
     * @return Returns the roleTemplates.
     */
    public List<RoleTemplate> getRoleTemplates() {
        return roleTemplates;
    }

    /**
     * Serializes object into Xml string.
     *
     * @return String with object serialized as Xml.
     */
    public String toXml() {
        StringBuffer xml = new StringBuffer();
        xml.append("<roleMap>");

        if (scopes != null) {
            for (Scope s : scopes) {
                xml.append(s.toXml());
            }
        }

        if (roleTemplates != null) {
            for (RoleTemplate r : roleTemplates) {
                xml.append(r.toXml());
            }
        }

        xml.append("</roleMap>");
        return xml.toString();
    }

    /**
     * Deserializes object from node.
     *
     * @param roleMapNode Node to use in deserialization process.
     * @return RoleMap object deserialized from node.
     */
    public static RoleMap toObject(Node roleMapNode) {
        RoleMap roleMap = new RoleMap();

        List<Scope> scopes = new ArrayList<Scope>();
        for (Node a : roleMapNode.getChildren()) {
            if ("scope".equals(a.getNodeValue())) {
                scopes.add(Scope.toObject(a));
            }
        }
        roleMap.setScopes(scopes);

        List<RoleTemplate> roleTemplates = new ArrayList<RoleTemplate>();
        for (Node a : roleMapNode.getChildren()) {
            if ("roleTemplate".equals(a.getNodeValue())) {
                roleTemplates.add(RoleTemplate.toObject(a, roleMap));
            }
        }
        roleMap.setRoleTemplates(roleTemplates);

        return roleMap;
    }

    /**
     * Finds role template based on specified name.
     *
     * @param roleTemplateRef String with name of role template to find.
     * @return RoleTemplate object or null if no role template exists with
     *  specified name.
     */
    public RoleTemplate getRoleTemplateByRef(String roleTemplateRef) {
        if (roleTemplates == null || roleTemplates.size() == 0
                || roleTemplateRef == null || roleTemplateRef.equals("")) {
            return null;
        }

        for (RoleTemplate a : roleTemplates) {
            if (roleTemplateRef.equals(a.getName())) {
                return a;
            }
        }

        return null;
    }

    /**
     * Finds scope based on specified name.
     *
     * @param scopeRef String with name of scope to find.
     * @return Scope object or null if no scope exists with specified name.
     */
    public Scope getScopeByRef(String scopeRef) {
        if (scopes == null || scopeRef == null || scopeRef.equals("")) {
            return null;
        }

        for (Scope a : scopes) {
            if (scopeRef.equals(a.getName())) {
                return a;
            }
        }

        return null;
    }
}
```

D.1.4 RoleTemplate.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.util.ArrayList;
import java.util.List;

import org.planx.xmlstore.Node;

/**
 * A role template contains information about allowed methods.
 *
 * @author pt
 */
public class RoleTemplate {
    private String name;

    private List<RoleTemplateMethod> methods;

    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }

    /**
     * @param methods
     *            The methods to set.
     */
    public void setMethods(List<RoleTemplateMethod> methods) {
        this.methods = methods;
    }

    /**
     * @return Returns the methods.
     */
    public List<RoleTemplateMethod> getMethods() {
        return methods;
    }

    public RoleTemplate(String name) {
        this.name = name;
    }

    /**
     * Serialized object into Xml string.
     *
     * @return Serialized object as string.
     */
    public String toXml() {
        StringBuffer xml = new StringBuffer(String.format(
                "<roleTemplate name=\"%s\"", new Object[] {
                        this.name }));

        if (this.methods != null && this.methods.size() > 0) {
            xml.append(">");
```

```java
            for (RoleTemplateMethod a : this.methods) {
                xml.append(a.toXml());
            }
            xml.append("</roleTemplate>");
        }
        else {
            xml.append(" />");
        }

        return xml.toString();
    }

    /**
     * Deserializes object from node.
     *
     * @param roleTemplateNode Node to use in deserialization process.
     * @param roleMap RoleMap object to use in deserialization process.
     * @return RoleTemplate object deserialized from node.
     */
    public static RoleTemplate toObject(Node roleTemplateNode, RoleMap roleMap) {
        RoleTemplate a = new RoleTemplate(roleTemplateNode.getAttribute("name"));

        List<Node> methodsNode = (List<Node>) roleTemplateNode.getChildren();
        if (methodsNode != null && methodsNode.size() > 0) {
            List<RoleTemplateMethod> methods = new ArrayList<RoleTemplateMethod>();
            for (Node methodNode : roleTemplateNode.getChildren()) {
                methods.add(RoleTemplateMethod.toObject(methodNode, roleMap));
            }
            a.setMethods(methods);
        }

        return a;
    }
}
```

D.1.5 RoleTemplateMethod.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import org.planx.xmlstore.Node;

/**
 * A role template method contains information about what scopes are
 * allowed on which methods e.g. is it possible to query all nodes
 * in a document having a subnode called 'Name'.
 *
 * @author pt
 */
public class RoleTemplateMethod {
    private RoleTemplateMethodType type;

    private Scope scope;

    /**
     * @return Returns the name.
     */
    public RoleTemplateMethodType getType() {
        return type;
    }

    /**
     * @return Returns the scope.
     */
    public Scope getScope() {
        return scope;
    }

    /**
     * Constructs new object instance.
     *
     * @param type
     * @param scope
     */
    public RoleTemplateMethod(RoleTemplateMethodType type, Scope scope) {
        if (type == null) {
            throw new IllegalArgumentException("Type must be specified");
        }

        if (scope == null) {
            throw new IllegalArgumentException("Scope must be specified");
        }

        this.type = type;
        this.scope = scope;
    }

    /**
     * Serializes object in Xml string.
     *
     * @return Returns serialized object into Xml.
     */
    public String toXml() {
        return String.format(
                "<roleTemplateMethod type=\"%s\" scopeRef=\"%s\" />",
                new Object[] { this.type, this.scope.getName() });
    }

    /**
     * Deserializes object from Node.
     *
     * @param methodNode Node with object information.
     * @param roleMap RoleMap object used in deserialization process.
     * @return RoleTemplateMethod instanted from node.
     */
    public static RoleTemplateMethod toObject(Node methodNode, RoleMap roleMap) {
        return new RoleTemplateMethod(RoleTemplateMethodType.valueOf(methodNode
                .getAttribute("type")), roleMap.getScopeByRef(methodNode
                .getAttribute("scopeRef")));
    }
}
```

D.1.6 Scope.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.util.ArrayList;
import java.util.List;

import org.planx.xmlstore.Node;

```

```java
/**
 * A scope contains information about which XPath expressions are allowed
 * or denied.
 *
 * @author pt
 */
public class Scope {
    private ScopeBase base;

    private String name;

    private List<Shape> shapes;

    /**
     * Constructs new object based on specified properties.
     *
     * @param base
     * @param name
     */
    public Scope(ScopeBase base, String name) {
        this.base = base;
        this.name = name;
    }

    /**
     * @return Returns the base.
     */
    public ScopeBase getBase() {
        return base;
    }

    /**
     * @return Returns the name.
     */
    public String getName() {
        return name;
    }

    /**
     * @param shapes
     *            The shapes to set.
     */
    public void setShapes(List<Shape> shapes) {
        this.shapes = shapes;
    }

    /**
     * @return Returns the shapes.
     */
    public List<Shape> getShapes() {
        return shapes;
    }

    public String toXml() {
        StringBuffer xml = new StringBuffer(String.format(
                "<scope base=\"%s\" name=\"%s\"", new Object[] { this.base,
                        this.name }));
        if (this.shapes != null && this.shapes.size() > 0) {
            xml.append(">");
            for (Shape s : this.shapes) {
                xml.append(s.toXml());
            }
            xml.append("</scope>");
        }
        else {
            xml.append(" />");
        }

        return xml.toString();
    }

    /**
     * Deserializes object from node.
     *
     * @param scopeNode Node used in deserialization process.
     * @return Scope object deserialized.
     */
    public static Scope toObject(Node scopeNode) {
        Scope a = new Scope(ScopeBase.valueOf(scopeNode.getAttribute("base")),
                scopeNode.getAttribute("name"));

        List<Node> shapeNodes = (List<Node>) scopeNode.getChildren();
        if (shapeNodes != null && shapeNodes.size() > 0) {
            List<Shape> shapes = new ArrayList<Shape>();
            for (Node shapeNode : shapeNodes) {
                shapes.add(Shape.toObject(shapeNode));
            }
            a.setShapes(shapes);
        }

        return a;
    }
}
```

D.1.7 ScopeBase.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

/**
 * Type of scopes which is either everything (indicated by 't') or nothing
 * (indicated by 'nil').
 *
 * @author pt
 */
public enum ScopeBase {
    t, nil
}
```

D.1.8 Shape.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import org.planx.xmlstore.Node;

/**
 * A shape contains information about a single XPath expression to include
 * in a scope as either an 'allow' or 'deny' node set.
 *
 * @author pt
 */
public class Shape {
    private ShapeType type;

    private String select;

    /**
     * Constructs new object based on specified properties.
     *
     * @param type
     * @param select
     */
    public Shape(ShapeType type, String select) {
        this.type = type;
        this.select = select;
    }

    /**
     * @return Returns the type.
     */
    public ShapeType getType() {
        return type;
    }

    /**
     * @return Returns the select.
     */
    public String getSelect() {
        return select;
    }

    /**
     * Serializes object into Xml string.
     *
     * @return Returns serialized object in Xml.
     */
    public String toXml() {
        return String.format("<shape type=\"%s\" select=\"%s\" />",
                new Object[] { type != null ? type : ShapeType.exclude,
                        select != null ? select : "" });
    }

    /**
     * Deserializes node into object.
     *
     * @param shapeNode Node to deserialize.
     * @return Shape object deserialized from node.
     */
    public static Shape toObject(Node shapeNode) {
        return new Shape(
                "include".equals(shapeNode.getAttribute("type")) ? ShapeType.include
                        : ShapeType.exclude, shapeNode.getAttribute("select"));
    }
}
```

D.1.9 ShapeType.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

/**
 * Types of shape available which may be either 'include' or 'exclude'
 * indicating a given node set should be fully included or fully excluded.
```

```java
 *
 * @author pt
 */
public enum ShapeType {
    include , exclude
}
```

D.1.10 Subject.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import org.planx.xmlstore.Node;

/**
 * A subject contains information about a user using its id. This subject is
 * used in a role object to determine what kind of access this user conforms
 * to.
 *
 * @author pt
 */
public class Subject {
    private String userId;

    /**
     *
     * @return Returns the userId.
     */
    public String getUserId() {
        return userId;
    }

    /**
     *
     * @param userId
     */
    public Subject(String userId) {
        this.userId = userId;
    }

    /**
     * Serializes object into an Xml string.
     *
     * @return String with object serialized as Xml.
     */
    public String toXml() {
        return String.format("<subject userId=\"%s\" />",
                new Object[] { this.userId });
    }

    /**
     * Deserializes object from node.
     *
     * @param n Node with information used in deserialization process.
     * @return Subject object deserialized.
     */
    public static Subject toObject(Node n) {
        return new Subject(n.getAttribute("userId"));
    }
}
```

D.2 org.planx.authx.filter

D.2.1 BaseFilter.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.filter;

import java.io.IOException;

import org.planx.authx.Role;
import org.planx.authx.RoleTemplateMethodType;
import org.planx.authx.Subject;
import org.planx.xmlstore.DocNode;
import org.planx.xmlstore.Node;
import org.planx.xpath.Environment;
import org.planx.xpath.XMLStoreNavigator;
import org.planx.xpath.XPath;
import org.planx.xpath.XPathException;
import org.planx.xpath.object.XNodeSet;
import org.planx.xpath.object.XObject;
import org.planx.xpath.object.XString;

/**
 * Core filter which may be used a base class for filters if filter wants to
 * use XPath expressions in its filtering process.
 *
 * @author pt
 */
public class BaseFilter {

    /**
     * Evaluate this XPath expression using the specified context node.
     *
     * @param node Node used as context node i.e. root of evaluation.
     * @param query String with XPath expression to be applied.
     * @throws IOException
     */
    public Node[] evaluateExpression(Node node, String query, Subject subject)
            throws IOException {
        try {
            XPath xp = new XPath(query, new XMLStoreNavigator());

            // include variable 'callerId' with every XPath query to allow
            // shapes to use this for filtering out 'own' nodes
            Environment<Node> e = new Environment<Node>();
            e.bindVariable("$callerId", new XString(subject.getUserId()));

            DocNode contextNode = null;
            if (!(node instanceof DocNode)) {
                contextNode = new DocNode(node);
            }
            else {
                contextNode = (DocNode)node;
            }

            XObject o = xp.evaluate(contextNode, e);
            if (!(o instanceof XNodeSet)) {
                throw new IOException("Expression did not return expected XNodeSet.");
            }

            return ((XNodeSet<Node>) o).toArray(new Node[0]);
        }
        catch (XPathException e) {
            throw new IOException(e.getMessage());
        }
    }
}
```

D.2.2 Filter.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.filter;

import java.io.IOException;

import org.planx.authx.Role;
import org.planx.authx.RoleTemplateMethodType;
import org.planx.xmlstore.Node;

/**
 * Interface used for filtering methods. Class instances must implement this
 * interface in order to evaluate 'query' and 'insert' filtering processes
 * before load/save in XML Store.
 *
 * @author pt
 */
public interface Filter {
    /**
     * Evaluates specified XPath expression based on context node and method
     * type. Expression is evaluted using role objects scope.
     *
     * @param contextNode Root node used for evaluation.
     * @param methodType Type of method used for filtering.
     * @param query XPath expression to be evaluted.
     * @param role Role to be used for evaluation.
     * @return Set of Node objects (might be empty set) of evaluated query.
     * @throws IOException
     */
    Node[] evaluate(Node contextNode, RoleTemplateMethodType methodType, String query,
        Role role) throws IOException;
}
```

D.2.3 QFilter.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.filter;

import java.io.IOException;
import java.security.AccessControlException;
import java.util.ArrayList;
import java.util.List;

import org.planx.authx.Role;
import org.planx.authx.RoleList;
import org.planx.authx.RoleMap;
import org.planx.authx.RoleTemplate;
import org.planx.authx.RoleTemplateMethod;
import org.planx.authx.RoleTemplateMethodType;
import org.planx.authx.ScopeBase;
import org.planx.authx.Shape;
import org.planx.authx.ShapeType;
import org.planx.xmlstore.Node;

/**
 * A filtering method based on "QFilter".
 *
 * @author pt
 */
public class QFilter extends BaseFilter implements Filter {

    private Role role;

    /* (non-Javadoc)
     * @see org.planx.authx.filter.Filter#evaluate(org.planx.xmlstore.Node, org.planx.
         authx.RoleTemplateMethodType, java.lang.String, org.planx.authx.Role)
     */
    public Node[] evaluate(Node contentNode, RoleTemplateMethodType methodType,
            String query, Role role) throws IOException {
        this.role = role;

        String safeQuery = qFilter(query, methodType);
        if (safeQuery == null) {
            // expression cannot be evaluted since it's a full 'deny'
          return new Node[0];
        }

        return super.evaluateExpression(contentNode, safeQuery, role.getSubject());
    }


    /**
     * Applies qfiltering to query returning a 'safe' query which may be
     * executed on an Xml document to return nodesets with only allowed
     * nodes.
     *
     * @param query XPath expression to evaluate.
     * @param type Type of method to evaluate.
     * @return String with 'safe' query based on qfiltering.
     */
    public String qFilter(String query, RoleTemplateMethodType type) {
        // intersect: $set1[count(.|$set2)==count($set2)]
        // except : $set1[count(.|$set2)!=count($set2)]

        RoleTemplate rt = role.getRoleTemplate();
        if (rt == null) {
            // no template associated with role -> fail
            throw new AccessControlException("Failed to lookup role template for role ");
        }

        List<Shape> shapes = new ArrayList<Shape>();

        // get shapes from RoleTemplate
        for (RoleTemplateMethod roleTemplateMethod : rt.getMethods()) {
            if (roleTemplateMethod.getType().equals(type)) {
                if (roleTemplateMethod.getScope().getBase() == ScopeBase.t) {
                    shapes.add(new Shape(ShapeType.include, "//*"));
                }
                if (roleTemplateMethod.getScope().getShapes() != null) {
                    for (Shape s : roleTemplateMethod.getScope().getShapes()) {
                        shapes.add(new Shape(s.getType(), (s.getSelect().endsWith("//*"))
                             ?
                                              s.getSelect() :
                                              s.getSelect() + "/descendant-or-self::*"));

                    }
                }
            }
        }

        // get shapes from RoleList
        if ((role.getScope() != null) && (role.getScope().getBase() == ScopeBase.t)) {
          shapes.add(new Shape(ShapeType.include, "//*"));
        }

        if (role.getScope() != null) {
          for (Shape s : role.getScope().getShapes()) {
            shapes.add(new Shape(s.getType(), (s.getSelect().endsWith("//*") ?
                                 s.getSelect() :
                                 s.getSelect()+"/descendant-or-self::*")));
          }
        }

        StringBuffer allowString = new StringBuffer();
        StringBuffer denyString = new StringBuffer();

        for (Shape shape : shapes) {
            if (shape.getType() == ShapeType.include) {
                allowString.append(shape.getSelect() + " | ");
            }
            else {
                denyString
                        .append(shape.getSelect() + "/ancestor-or-self::* | ");
            }
        }

        if (allowString.length() > 0) {
            allowString.setLength(allowString.length() - 3);
        }

        if (denyString.length() > 0) {
            denyString.setLength(denyString.length() - 3);
        }

        // intersect: $set1[count(.|$set2)=count($set2)]
        // except : $set1[count(.|$set2)!=count($set2)]

        if (allowString.length() == 0) {
            return null;
        }

        if (denyString.length() == 0) {
            // query intersect allow
          // q[count(.|A)=count(A)]
            return (query == "/" ? query + "node()" : query) + "[count(.|"
                    + allowString + ")=count(" + allowString + ")]";
        }

        // Q intersect (A \ D) = (Q intersect A) \ D
        // (q[count(.|A)=count(A)])[count(.|D)!=count(D)]

        String safeXpath = (query == "/" ? query + "node()" : query) +
              "[count(.|" + allowString + ")=count(" + allowString + ")]";
        safeXpath += "[count(.|" + denyString + ")!=count(" + denyString
              + ")]";
        return safeXpath;
    }
}
```

D.3 org.planx.authx.store

D.3.1 AuthNameServer.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

import java.io.IOException;

import org.planx.authx.Subject;
import org.planx.xmlstore.NameServer;
import org.planx.xmlstore.NameServerException;
import org.planx.xmlstore.Reference;
import org.planx.xmlstore.references.ValueReference;

/**
 * An authorized name server implemented as a decorator to wrap existing
 * NameServer instances and thus provide authorization to that instance.
 *
 * @author pt
 */
public class AuthNameServer<R extends NameServer<Reference>> implements
        NameServer<AuthReference>, BindNames {
    private R nameServer;

    /**
     * Constructs new AuthNameServer instance.
     *
     * @param nameServer
     * @param subject
     */
    public AuthNameServer(R nameServer) {
        this.nameServer = nameServer;
    }


    /* (non-Javadoc)
     * @see org.planx.xmlstore.NameServer#lookup(java.lang.String)
     */
    public AuthReference lookup(String name) throws IOException,
            NameServerException {
        return lookup(name, AuthReferenceDocumentType.content);
    }


    /**
     * Looks up reference in name server based on specified name. In case no
     * reference exists with this name, the method returns null.
     *
     * @param name String with name to lookup.
     * @param documentType Type of document to lookup.
     * @return AuthReference binded to specified name or null if no
     *  reference existed for that name.
     * @throws IOException
     * @throws NameServerException
     */
    public AuthReference lookup(String name,
            AuthReferenceDocumentType documentType) throws IOException,
            NameServerException {

        // wrap instance into an authorized reference to be returned
        AuthReference authReference = new AuthReference();

        String aclBindName = (AuthReferenceDocumentType.content == documentType) ? name
                : null;

        // perform lookup for "creator" document node
        authReference.setCreatorReference(getCreatorReference(name));

        // perform lookup for "roleMap" document node
        authReference.setRoleMapReference(getRoleMapReference(aclBindName));

        // perform lookup for "roleList" document node
        authReference.setRoleListReference(getRoleListReference(aclBindName));

        // set "content" reference based on requested document type
        if (AuthReferenceDocumentType.content == documentType) {
            authReference.setContentReference(nameServer.lookup(name));
        }
        else if (AuthReferenceDocumentType.roleMap == documentType) {
            authReference.setContentReference(getRoleMapReference(name));
        }
        else if (AuthReferenceDocumentType.roleList == documentType) {
            authReference.setContentReference(getRoleListReference(name));
        }

        return authReference;
    }


    /**
     * Returns "Creator" bind name.
     *
     * @param bindName String with user id.
     * @return Bind name for "Creator" reference.
     */
    private String getCreatorBindName(String bindName) {
        return String.format(CreatorName, bindName);
    }


    /**
     * Returns "RoleMap" bind name.
     *
     * @param bindName String with user id or null if "system" lookup.
     * @return Bind name for "RoleMap" or "SystemRoleMap" reference.
     */
    private String getRoleMapBindName(String bindName) {
        return (bindName == null) ? SystemRoleMapName : String.format(RoleMapName,
            bindName);
    }


    /**
     * Returns "RoleList" bind name.
     *
     * @param bindName String with user id or null if "system" lookup.
     * @return Bind name for "RoleList" or "SystemRoleList" reference.
     */
    private String getRoleListBindName(String bindName) {
        return (bindName == null) ? SystemRoleListName : String.format(RoleListName,
            bindName);
    }


    /**
```

```
     * Looks up "Creator" reference.
     *
     * @param bindName String with user id.
     * @return A reference to "Creator" node or null if no reference could
     *  be located.
     * @throws IOException
     * @throws NameServerException
     */
    private Reference getCreatorReference(String bindName) throws IOException,
            NameServerException {
        return nameServer.lookup(getCreatorBindName(bindName));
    }


    /**
     * Looks up "RoleMap" reference.
     *
     * @param bindName Name of reference to lookup.
     * @return A reference to "RoleMap" node or null if no reference could
     *  be located.
     * @throws NameServerException
     * @throws IOException
     */
    private Reference getRoleMapReference(String bindName) throws IOException,
            NameServerException {
        return nameServer.lookup(getRoleMapBindName(bindName));
    }


    /**
     * Looks up "RoleList" reference.
     *
     * @param bindName Name of reference to lookup.
     * @return A reference to "RoleList" node or null if no reference could
     *  be located.
     * @throws NameServerException
     * @throws IOException
     */
    private Reference getRoleListReference(String bindName) throws IOException,
            NameServerException {
        return nameServer.lookup(getRoleListBindName(bindName));
    }


    /**
     * Binds name to all authorization references in name server.
     *
     * @param name String with name to bind.
     * @param ref AuthReference with references to bind.
     * @throws IOException
     * @throws NameServerException
     */
    public void bind(String name, AuthReference ref) throws IOException,
        NameServerException {
        // TODO the following four lines of code should be one atomic operation
        nameServer.bind(getCreatorBindName(name), ref.getCreatorReference());
        nameServer.bind(getRoleMapBindName(name), ref.getRoleMapReference());
        nameServer.bind(getRoleListBindName(name), ref.getRoleListReference());
        nameServer.bind(name, ref.getContentReference());
    }


    /**
     * Rebinds name in name server.
     *
     * @param name String with name to bind.
     * @param oldRef Old AuthReference to bind to new name.
     * @param newRef New AuthReference to replace old reference.
     * @throws IOException
     * @throws NameServerException
     */
    public void rebind(String name, AuthReference oldRef, AuthReference newRef)
        throws IOException, NameServerException {
        rebind(name, oldRef, newRef, AuthReferenceDocumentType.content);
    }


    /**
     * Rebinds name in name server for a given type of document.
     *
     * @param name String with name to bind.
     * @param oldRef Old AuthReference to bind to new name.
     * @param newRef New AuthReference to replace old reference.
     * @param documentType Type of document.
     * @throws IOException
     * @throws NameServerException
     * @throws IllegalArgumentException If type of document is unrecognized.
     */
    public void rebind(String name, AuthReference oldRef, AuthReference newRef,
        AuthReferenceDocumentType documentType) throws IOException, NameServerException {
        if (AuthReferenceDocumentType.content.equals(documentType)) {
            nameServer.rebind(name, oldRef.getContentReference(), newRef.
                getContentReference());
        }
        else if (AuthReferenceDocumentType.roleMap.equals(documentType)) {
            nameServer.rebind(getRoleMapBindName(name), oldRef.getContentReference(),
                newRef.getContentReference());
        }
        else if (AuthReferenceDocumentType.roleList.equals(documentType)) {
            nameServer.rebind(getRoleListBindName(name), oldRef.getContentReference(),
                newRef.getContentReference());
        }
        else {
            throw new IllegalArgumentException("Document type \"" + documentType + "\"
                not recognized.");
        }
    }


    /* (non-Javadoc)
     * @see org.planx.xmlstore.NameServer#close()
     */
    public void close() throws IOException {
        nameServer.close();
    }
}
```

D.3.2 AuthReference.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

import org.planx.xmlstore.Reference;

/**
 * An AuthReference implements Reference interface but is actually a
 * container for multiple references.
 *
 * @author pt
 */
public class AuthReference implements Reference {
    private Reference creator;

    private Reference roleMap;

    private Reference roleList;

    private Reference content;

    /**
     * @return
     */
    Reference getCreatorReference() {
        return creator;
    }

    /**
     * @param creator
     */
    void setCreatorReference(Reference creator) {
        this.creator = creator;
    }

    /**
     * @return Returns the roleMap.
     */
    Reference getRoleMapReference() {
        return roleMap;
    }

    /**
     * @param roleMap
     *            The roleMap to set.
     */
    void setRoleMapReference(Reference roleMap) {
        this.roleMap = roleMap;
    }

    /**
     * @return Returns the content.
     */
    Reference getContentReference() {
        return content;
    }

    /**
     * @param content
     *            The content to set.
     */
    void setContentReference(Reference content) {
        this.content = content;
    }

    /**
     * @return Returns the roleList.
     */
    Reference getRoleListReference() {
        return roleList;
    }

    /**
```

```
     * @param roleList
     *            The roleList to set.
     */
    void setRoleListReference(Reference roleList) {
        this.roleList = roleList;
    }
}
```

D.3.3 AuthReferenceDocumentType.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

/**
 * Types of documents used for authorization. Documents of type "roleList" and
 * "roleMap" are based on a specific schema where "content" is any abitrary
 * type of document.
 *
 * @author pt
 */
public enum AuthReferenceDocumentType {
    content, roleList, roleMap
}
```

D.3.4 AuthXMLStore.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

import java.io.IOException;
import java.io.StringReader;
import java.security.AccessControlException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.planx.authx.*;
import org.planx.authx.filter.Filter;
import org.planx.authx.filter.QFilter;
import org.planx.xmlstore.DocNode;
import org.planx.xmlstore.NameServer;
import org.planx.xmlstore.Node;
import org.planx.xmlstore.Reference;
import org.planx.xmlstore.UnknownReferenceException;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.XMLStore;
import org.planx.xmlstore.input.SAXBuilder;
import org.planx.xmlstore.koala.nodes.DVMElementNode;
import org.planx.xmlstore.koala.nodes.SystemNode;
import org.planx.xpath.Environment;
import org.planx.xpath.XMLStoreNavigator;
import org.planx.xpath.XPath;
import org.planx.xpath.XPathException;
import org.planx.xpath.object.XNodeSet;
import org.planx.xpath.object.XObject;

/**
 * An implementation of a <code>XMLStore</code> providing authorization on
 * stored <code>Reference</code>s.
 *
 * @author pt
 */
public class AuthXMLStore implements XMLStore<AuthReference>, BindNames {
    private XMLStore<Reference> xmlStore;

    private AuthNameServer nameServer;

    private Subject subject;

    private Filter filter;

    /**
     * Initializes XML Store instance providing authorization. Defaults to
     * currently logged on user by reading "USERNAME" system property with
     * a 'QFilter' filtering method.
     *
     * @param xmlStore Instance of XML Store to be wrapped.
     * @throws Exception
     */
    public AuthXMLStore(XMLStore xmlStore) throws Exception {
        this(xmlStore, new Subject(System.getProperty("USERNAME")));
    }


    /**
     * Initializes XML Store instance providing authorization based on
     * specified user and using a 'QFilter' filtering method.
     *
     * @param xmlStore Instance of XML Store to be wrapped.
     * @param subject
     * @throws Exception
     */
    public AuthXMLStore(XMLStore xmlStore, Subject subject) throws Exception {
        this(xmlStore, subject, new QFilter());
    }


    /**
     * Initializes XML Store instance providing authorization based on
     * specified user and filtering method.
     *
     * @param xmlStore Instance of XML Store to be wrapped.
     * @param subject
     * @throws Exception
     */
    public AuthXMLStore(XMLStore xmlStore, Subject subject,
            Filter filter) throws Exception {
        this.xmlStore = (XMLStore<Reference>) xmlStore;
        this.subject = subject;
        this.filter = filter;

        AuthReference srmRef = getNameServer().lookup(SystemRoleMapName);
        if (srmRef == null) {
            // create generic roleMap (and bind to name "SystemRoleMap")
            srmRef = save(getSystemRoleMapNode());
            getNameServer().bind(SystemRoleMapName, srmRef);
        }

        AuthReference srlRef = getNameServer().lookup(SystemRoleListName);
        if (srlRef == null) {
            // create generic roleList (and bind to name "SystemRoleList")
            srlRef = save(getSystemRoleListNode());
            getNameServer().bind(SystemRoleListName, srlRef);
        }
    }


    /* (non-Javadoc)
     * @see org.planx.xmlstore.XMLStore#close()
     */
    public void close() throws IOException {
        if (xmlStore != null) {
            xmlStore.close();
            xmlStore = null;
        }
    }


    /*
     * (non-Javadoc)
     *
     * @see org.planx.xmlstore.stores.AbstractXMLStore#getNameServer()
     */
    public AuthNameServer getNameServer() {
        if (nameServer == null) {
            // lazy loading of name server reference
            nameServer = new AuthNameServer(xmlStore.getNameServer());
        }

        return nameServer;
    }


    /**
     * Loads Node from reference.
     *
     * @param ref Reference to load.
     * @return Node object loaded from reference.
     * @throws IOException
     * @throws UnknownReferenceException
     */
    public Node load(AuthReference ref) throws IOException,
            UnknownReferenceException {
        Node[] nodes = load(ref, "/");
        return (nodes.length > 0) ? nodes[0] : null;
    }


    /**
     * Loads set of Node objects from specified XPath query.
     *
     * @param ref Reference to load.
     * @param query XPath query pointing to node(s) to load.
     * @return Set of Node objects or null if query resulted in zero nodes.
     * @throws IOException
     * @throws UnknownReferenceException
     */
    public Node[] load(AuthReference ref, String query) throws IOException,
            UnknownReferenceException {

        Node creatorNode = null;
        Node roleMapNode = null;
        Node roleListNode = null;

        Reference r = null;

        if ((r = ref.getCreatorReference()) != null) {
            creatorNode = xmlStore.load(r);
        }

        if ((r = ref.getRoleMapReference()) != null) {
            roleMapNode = xmlStore.load(r);
        }

        if ((r = ref.getRoleListReference()) != null) {
            roleListNode = xmlStore.load(r);
        }

        // a full authorization reference node contains three additional
        // nodes beside its "content" node. If either of these nodes are
        // not available, authorization is not in effect
        if (creatorNode == null || roleMapNode == null || roleListNode == null) {
            // no authorization available for stored node
            return new Node[] { xmlStore.load(ref.getContentReference()) };
        }

        Subject creatorSubject = Subject.toObject(creatorNode);
        RoleMap roleMap = RoleMap.toObject(roleMapNode);
        RoleList roleList = RoleList.toObject(roleListNode, roleMap);

        Role role = null;
        if (creatorSubject.getUserId().equals(this.subject.getUserId())) {
            role = roleList.getRoleByUserId(CreatorId);
        }

        if (role != null) {
            // replace role with creator stored in xmlstore to allow generic
            // RoleList instances
            role = new Role(role.getRoleTemplate(), creatorSubject);
        }
        else {
            role = roleList.getRoleByUserId(this.subject.getUserId());
        }

        if (role == null) {
            throw new AccessControlException("Failed to lookup role for "
                    + this.subject.getUserId());
        }

        // always retrieve "content" node which should be available always
        Node contentNode = xmlStore.load(ref.getContentReference());

        return filter.evaluate(contentNode, RoleTemplateMethodType.query,
                query, role);
    }

    /* (non-Javadoc)
     * @see org.planx.xmlstore.XMLStore#save(org.planx.xmlstore.Node)
     */
    public AuthReference save(Node node) throws IOException {
        AuthReference a = new AuthReference();
        try {
            // TODO following save operations should be one atomic operation
            a.setCreatorReference(xmlStore.save(getCreatorNode()));
            a.setRoleMapReference(xmlStore.save(getRoleMapNode()));
            a.setRoleListReference(xmlStore.save(getRoleListNode()));
            a.setContentReference(xmlStore.save(node));
        }
        catch (XMLException x) {
            throw new IOException("Failed to setup one or more authorization nodes.");
        }
        return a;
    }
```

```java


    /**
     * Returns creator node.
     *
     * @return Node with creator details.
     * @throws IOException
     * @throws XMLException
     */
    private Node getCreatorNode() throws IOException, XMLException {
        return SAXBuilder.build(new StringReader("<subject userId=\""
                + this.subject.getUserId() + "\" />"));
    }


    /**
     * Returns a generic role map node.
     *
     * @return Node with role map.
     * @throws IOException
     * @throws XMLException
     */
    private Node getRoleMapNode() throws IOException, XMLException {
        return SAXBuilder.build(new StringReader("<roleMap>"
                + "<scope base=\"t\" name=\"all\" />"
                + "<roleTemplate name=\"rt1\">"
                + "<roleTemplateMethod type=\"query\" scopeRef=\"all\" />"
                + "<roleTemplateMethod type=\"insert\" scopeRef=\"all\" />"
                + "</roleTemplate>"
                + "<roleTemplate name=\"rt2\">"
                + "<roleTemplateMethod type=\"query\" scopeRef=\"all\" />"
                + "</roleTemplate>"
                + "</roleMap>"));
    }


    /**
     * Returns a system role map node.
     *
     * @return Node with system role map.
     * @throws IOException
     */
    private Node getSystemRoleMapNode() throws IOException, XMLException {
        return getRoleMapNode();
    }


    /**
     * Returns a generic role list node with full access to creator.
     *
     * @return Node with role list.
     * @throws IOException
     * @throws XMLException
     */
    private Node getRoleListNode() throws IOException, XMLException {
        return SAXBuilder.build(new StringReader(
                "<roleList><role roleTemplateRef=\"rt1\"><subject userId=\"" + this.
                    subject.getUserId() + "\" /></role></roleList>"));
    }


    /**
     * Returns a system role list node which is used a placeholder for any
     * creator saving nodes in XML Store.
     *
     * @return Node with system role list.
     * @throws IOException
     * @throws XMLException
     */
    private Node getSystemRoleListNode() throws IOException, XMLException {
        return SAXBuilder.build(new StringReader(
                "<roleList><role roleTemplateRef=\"rt1\"><subject userId=\"" + CreatorId
                    + "\" /></role></roleList>"));
    }
}
```

D.3.5 BindNames.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

/**
 * List of string names used as binding names when performing a 'bind'
 * on a given NameServer instance.
 *
 * @author pt
 */
public interface BindNames {
    String SystemRoleMapName = "urn:authx:SystemRoleMap";

    String SystemRoleListName = "urn:authx:SystemRoleList";

    String RoleMapName = "urn:authx:%1$s RoleMap";

    String RoleListName = "urn:authx:%1$s RoleList";

    String CreatorName = "urn:authx:%1$s Creator";

    // used for generic role list subject
    String CreatorId = "urn:authx:creator";
}
```

E Test cases

Appendix including developed test cases grouped into packages and sorted alphabetically by filename.

E.1 org.planx.authx

E.1.1 RoleListTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;

import org.planx.xmlstore.Node;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
public class RoleListTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.RoleList.toXml()'
     */
    public void testToXml() {
        RoleList a;
        List<Scope> scopes;
        List<Role> roles;

        a = new RoleList();
        assertEquals("<roleList></roleList>", a.toXml());

        a = new RoleList();
        scopes = new ArrayList();
        scopes.add(new Scope(ScopeBase.t, "s1"));
        a.setScopes(scopes);
        assertEquals("<roleList>" + scopes.get(0).toXml() + "</roleList>", a
                .toXml());

        a = new RoleList();
        roles = new ArrayList();
        roles.add(new Role(new RoleTemplate("r1"), new Subject("s1")));
        a.setRoles(roles);
        assertEquals("<roleList>" + roles.get(0).toXml() + "</roleList>", a
                .toXml());
    }

    public void testToObjectFull() throws XMLException, IOException {
        Node roleMapNode = SAXBuilder
                .build(new StringReader(
                        "<roleMap><scope base=\"t\" name=\"Everything\" /><scope base=\"
                            nil\" name=\"Creator\"><shape type=\"include\" select=\"//*[
                            @creator='$callerId']\" /></scope><scope base=\"nil\" name=\"
                            Public\"><shape type=\"include\" select=\"//*[@creator='
                            $callerId']\" /><shape type=\"include\" select=\"//*[@type='
                            public']\" /></scope><scope base=\"nil\" name=\"Unauthorized
                            \" /><roleTemplate name=\"Owner\" description=\"Create, read
                            and modify all items.\"><roleTemplateMethod type=\"insert\"
                            scopeRef=\"Everything\" /><roleTemplateMethod type=\"query\"
                            scopeRef=\"Everything\" /></roleTemplate><roleTemplate name
                            =\"Author\" description=\"Create and read items and modify
                            items you create.\"><roleTemplateMethod type=\"insert\"
                            scopeRef=\"Everything\" /><roleTemplateMethod type=\"query\"
                            scopeRef=\"Everything\" /></roleTemplate><roleTemplate name
                            =\"Reviewer\" description=\"Read items only.\"><
                            roleTemplateMethod type=\"query\" scopeRef=\"Everything\"
                            /></roleTemplate></roleMap>"));
        Node n = SAXBuilder
                .build(new StringReader(
                        "<roleList><role scopeRef=\"Everything\" roleTemplateRef=\"Owner
                            \"><subject userId=\"pt\" /></role><role scopeRef=\"
                            Everything\" roleTemplateRef=\"Author\"><subject userId=\"mb
                            \" /></role><role roleTemplateRef=\"Reviewer\"><subject
                            userId=\"fb\" /></role></roleList>"));

        RoleList a = RoleList.toObject(n, RoleMap.toObject(roleMapNode));
        assertNotNull(a);
        assertEquals("pt", a.getRoles().get(0).getSubject().getUserId());
        assertNotNull(a.getRoles().get(0).getRoleTemplate());
        assertEquals("Owner", a.getRoles().get(0).getRoleTemplate().getName());
    }

}
```

E.1.2 RoleMapTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;

import org.planx.xmlstore.Node;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
public class RoleMapTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.RoleMap.FindRoleTemplateByName(String)'
     */
    public void testFindRoleTemplateByName() {

    }

    /*
     * Test method for 'org.planx.authx.RoleMap.toXml()'
     */
    public void testToXmlEmpty() {
        RoleMap a = new RoleMap();
        assertEquals("<roleMap></roleMap>", a.toXml());
    }

    public void testToXmlSingleRoleTemplate() {
        RoleMap a = new RoleMap();
        List<RoleTemplate> roleTemplates = new ArrayList<RoleTemplate>();
        roleTemplates.add(new RoleTemplate("r1"));
        a.setRoleTemplates(roleTemplates);
        assertEquals("<roleMap>" + roleTemplates.get(0).toXml() + "</roleMap>",
                a.toXml());
    }

    public void testToXmlSingleScope() {
        RoleMap a = new RoleMap();
        List<Scope> scopes = new ArrayList<Scope>();
        scopes.add(new Scope(ScopeBase.t, "s1"));
        a.setScopes(scopes);
        assertEquals("<roleMap>" + scopes.get(0).toXml() + "</roleMap>", a
                .toXml());
    }

    public void testToObjectPlain() throws XMLException, IOException {
        Node n = SAXBuilder.build(new StringReader("<roleMap />"));

        RoleMap a = RoleMap.toObject(n);
        assertNotNull(a);
        assertEquals(0, a.getScopes().size());
        assertEquals(0, a.getRoleTemplates().size());
    }

    public void testToObjectScopes() throws XMLException, IOException {
        Node n = SAXBuilder.build(new StringReader(
                "<roleMap><scope base=\"t\" name=\"s1\" /></roleMap>"));

        RoleMap a = RoleMap.toObject(n);
        assertNotNull(a);
        assertEquals(1, a.getScopes().size());
        assertEquals(0, a.getRoleTemplates().size());
        assertEquals("s1", a.getScopes().get(0).getName());
    }

    public void testToObjectRoleTemplates() throws XMLException, IOException {
        Node n = SAXBuilder
                .build(new StringReader(
                        "<roleMap><scope base=\"t\" name=\"s1\" /><roleTemplate name=\"
                            rt1\" /></roleMap>"));

        RoleMap a = RoleMap.toObject(n);
        assertNotNull(a);
        assertEquals(1, a.getScopes().size());
        assertEquals(1, a.getRoleTemplates().size());
        assertEquals("s1", a.getScopes().get(0).getName());
        assertEquals("rt1", a.getRoleTemplates().get(0).getName());
    }

    public void testToObjectFull() throws XMLException, IOException {
        Node n = SAXBuilder
                .build(new StringReader(
                        "<roleMap><scope base=\"t\" name=\"Everything\" /><scope base=\"
                            nil\" name=\"Creator\"><shape type=\"include\" select=\"//*[
                            @creator='$callerId']\" /></scope><scope base=\"nil\" name=\"
                            Public\"><shape type=\"include\" select=\"//*[@creator='
                            $callerId']\" /><shape type=\"include\" select=\"//*[@type='
                            public']\" /></scope><scope base=\"nil\" name=\"Unauthorized
                            \" /><roleTemplate name=\"Owner\" description=\"Create, read,
                             modify, and delete all items.\"><roleTemplateMethod type=\"
                            insert\" scopeRef=\"Everything\" /><roleTemplateMethod type
                            =\"query\" scopeRef=\"Everything\" /></roleTemplate><
                            roleTemplate name=\"Author\" description=\"Create and read
                            items, and modify and delete items you create.\"><
                            roleTemplateMethod type=\"insert\" scopeRef=\"Everything\"
                            /><roleTemplateMethod type=\"query\" scopeRef=\"Everything\"
                            /></roleTemplate><roleTemplate name=\"Reviewer\" description
                            =\"Read items only.\"><roleTemplateMethod type=\"query\"
                            scopeRef=\"Everything\" /></roleTemplate></roleMap>"));

        RoleMap a = RoleMap.toObject(n);
        assertNotNull(a);
        assertEquals(4, a.getScopes().size());
        assertEquals(3, a.getRoleTemplates().size());
        assertEquals("Everything", a.getScopes().get(0).getName());
        assertEquals("Owner", a.getRoleTemplates().get(0).getName());
    }
}
```

E.1.3 RoleTemplateMethodTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
public class RoleTemplateMethodTest extends TestCase {

    public void testEmptyName() {
        try {
            RoleTemplateMethod a = new RoleTemplateMethod(null, new Scope(
                    ScopeBase.t, "s1"));
            assertTrue(false);
        }
        catch (IllegalArgumentException x) {
            assertTrue(true);
        }
    }

    public void testEmptyScope() {
        try {
            RoleTemplateMethod a = new RoleTemplateMethod(
                    RoleTemplateMethodType.insert, null);
            assertTrue(false);
        }
        catch (IllegalArgumentException x) {
            assertTrue(true);
        }
    }

    /*
     * Test method for 'org.planx.authx.RoleTemplateMethod.toXml()'
     */
    public void testToXml() {
        RoleTemplateMethod a = new RoleTemplateMethod(
                RoleTemplateMethodType.insert, new Scope(ScopeBase.t, "s1"));
        assertEquals("<roleTemplateMethod type=\"insert\" scopeRef=\"s1\" />",
                a.toXml());

        a = new RoleTemplateMethod(RoleTemplateMethodType.query, new Scope(
                ScopeBase.t, "s1"));
        assertEquals("<roleTemplateMethod type=\"query\" scopeRef=\"s1\" />", a
                .toXml());
    }

}
```

E.1.4 RoleTemplateTest.java

```
/**
 * Authorization in XML Store
 *
 */
```

```
package org.planx.authx;

import java.util.ArrayList;
import java.util.List;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
public class RoleTemplateTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.RoleTemplate.toXml()'
     */
    public void testToXml() {
        RoleTemplate a = new RoleTemplate("rt1");
        assertEquals("<roleTemplate name=\"rt1\" />", a.toXml());

        a = new RoleTemplate("rt1");
        assertEquals("<roleTemplate name=\"rt1\" />", a.toXml());

        a = new RoleTemplate("rt1");
        List<RoleTemplateMethod> rtml = new ArrayList<RoleTemplateMethod>();
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, new Scope(ScopeBase.t, "s1"));
        rtml.add(rtm);
        a.setMethods(rtml);
        assertEquals("<roleTemplate name=\"rt1\">" + rtm.toXml()
                + "</roleTemplate>", a.toXml());
    }

    /*
     * Test method for 'org.planx.authx.RoleTemplate.toObject(Node, RoleMap)'
     */
    public void testToObject() {

    }

}
```

E.1.5 RoleTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import junit.framework.TestCase;

import org.planx.xmlstore.DocNode;
import org.planx.xmlstore.Node;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;
import org.planx.xpath.Navigator;
import org.planx.xpath.XMLStoreNavigator;
```

```java
import org.planx.xpath.XPath;
import org.planx.xpath.XPathException;
import org.planx.xpath.object.XNodeSet;
import org.planx.xpath.object.XNumber;
import org.planx.xpath.object.XObject;

import com.sun.org.apache.xerces.internal.xni.XNIException;

/**
 *
 *
 * @author pt
 *
 */
public class RoleTest extends TestCase {
    /*
     * Test method for 'org.planx.authx.Role.toXml()'
     */
    public void testToXml() {
        RoleTemplate rt = new RoleTemplate("t1");
        Subject s = new Subject("s1");
        Scope scope = new Scope(ScopeBase.t, "scope1");

        Role a = null;

        a = new Role(rt, s);
        assertEquals("<role roleTemplateRef=\"t1\">" + s.toXml() + "</role>", a
                .toXml());

        a = new Role(rt, s);
        a.setScope(scope);
        assertEquals("<role roleTemplateRef=\"t1\" scopeRef=\"scope1\">"
                + s.toXml() + "</role>", a.toXml());
    }

    /**
     * @throws XMLException
     * @throws IOException
     */
    public void testToObjectNoScope() throws XMLException, IOException {
        Node n = SAXBuilder
                .build(new StringReader(
                        "<role roleTemplateRef=\"r1\"><subject userId=\"pt\" /></role>"))
                            ;

        RoleMap roleMap = new RoleMap();
        List<RoleTemplate> roleTemplates = new ArrayList<RoleTemplate>();
        roleTemplates.add(new RoleTemplate("r1"));
        roleMap.setRoleTemplates(roleTemplates);

        Role a = Role.toObject(n, roleMap);
        assertNotNull(a);
        assertEquals("pt", a.getSubject().getUserId());
        assertEquals("r1", a.getRoleTemplate().getName());
        assertTrue(a.getScope() == null);
    }

    /**
     * @throws XMLException
     * @throws IOException
     */
    public void testToObjectOneScope() throws XMLException, IOException {
        Node n = SAXBuilder
                .build(new StringReader(
                        "<role roleTemplateRef=\"r1\" scopeRef=\"s1\"><subject userId=\"
                            foo\" /></role>"));

        RoleMap roleMap = new RoleMap();
        List<RoleTemplate> roleTemplates = new ArrayList<RoleTemplate>();
        roleTemplates.add(new RoleTemplate("r1"));
        roleMap.setRoleTemplates(roleTemplates);
        List<Scope> scopes = new ArrayList<Scope>();
        scopes.add(new Scope(ScopeBase.t, "s1"));
        roleMap.setScopes(scopes);

        Role a = Role.toObject(n, roleMap);
        assertNotNull(a);
        assertEquals("foo", a.getSubject().getUserId());
        assertEquals("r1", a.getRoleTemplate().getName());
        assertEquals("s1", a.getScope().getName());
    }
}
```

E.1.6 ScopeTest.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;

import org.planx.xmlstore.Node;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;

import junit.framework.TestCase;

public class ScopeTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.Scope.toXml()'
     */
    public void testToXml() {
        Scope s;
        List<Shape> shapes;

        s = new Scope(ScopeBase.nil, "t0");
        assertEquals("<scope base=\"nil\" name=\"t0\" />", s.toXml());

        s = new Scope(ScopeBase.t, "t1");
        assertEquals("<scope base=\"t\" name=\"t1\" />", s.toXml());

        s = new Scope(ScopeBase.t, "t1");
        s.setShapes(new ArrayList());
        assertEquals("<scope base=\"t\" name=\"t1\" />", s.toXml());

        s = new Scope(ScopeBase.t, "t1");
        shapes = new ArrayList();
        shapes.add(new Shape(ShapeType.include, "//*"));
        s.setShapes(shapes);
        assertEquals("<scope base=\"t\" name=\"t1\">" + shapes.get(0).toXml()
                + "</scope>", s.toXml());
    }

    public void testToObjectT() throws XMLException, IOException {
        Node n = SAXBuilder.build(new StringReader(
                "<scope base=\"t\" name=\"s1\" />"));

        Scope a = Scope.toObject(n);
        assertNotNull(a);
        assertEquals(ScopeBase.t, a.getBase());
        assertEquals("s1", a.getName());
    }

    public void testToObjectNil() throws XMLException, IOException {
        Node n = SAXBuilder.build(new StringReader(
                "<scope base=\"nil\" name=\"s1\" />"));

        Scope a = Scope.toObject(n);
        assertNotNull(a);
        assertEquals(ScopeBase.nil, a.getBase());
        assertEquals("s1", a.getName());
    }
}
```

E.1.7 ShapeTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import junit.framework.TestCase;

public class ShapeTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.Shape.toXml()'
     */
    public void testToXml() {
        Shape s;

        s = new Shape(ShapeType.include, "//Contact");
        assertEquals("<shape type=\"include\" select=\"//Contact\" />", s
                .toXml());

        s = new Shape(ShapeType.exclude, "//Contact");
        assertEquals("<shape type=\"exclude\" select=\"//Contact\" />", s
                .toXml());

        s = new Shape(ShapeType.exclude, "");
        assertEquals("<shape type=\"exclude\" select=\"\" />", s.toXml());
    }

}
```

E.1.8 SubjectTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx;

import junit.framework.TestCase;

```

```java
public class SubjectTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.Subject.toXml()'
     */
    public void testToXml() {
        Subject s;

        s = new Subject("pt");
        assertEquals("<subject userId=\"pt\" />", s.toXml());
    }

}
```

E.2 org.planx.authx.filter

E.2.1 QFilterTest.java

```java
/**
 *
 */
package org.planx.authx.filter;

import java.io.IOException;
import java.io.StringReader;
import java.util.ArrayList;
import java.util.List;

import org.planx.authx.Role;
import org.planx.authx.RoleTemplate;
import org.planx.authx.RoleTemplateMethod;
import org.planx.authx.RoleTemplateMethodType;
import org.planx.authx.Scope;
import org.planx.authx.ScopeBase;
import org.planx.authx.Shape;
import org.planx.authx.ShapeType;
import org.planx.authx.Subject;
import org.planx.xmlstore.DocNode;
import org.planx.xmlstore.Node;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
public class QFilterTest extends TestCase {

    /*
     * Test method for 'org.planx.authx.filter.QFilter.evaluate(Node,
         RoleTemplateMethodType, String, Role)'
     */

  Node contacts;

  protected void setUp() throws Exception {
        super.setUp();

        contacts = SAXBuilder.build(new StringReader("" +
      "<Contacts>" +
      "  <Contact Id=\"pt\">" +
      "    <Name>" +
      "       <Title>Mr.</Title>" +
      "       <GivenName>Peter</GivenName>" +
      "       <SurName>Theill</SurName>" +
      "     </Name>" +
      "     <EmailAddress>" +
      "      <Cat Ref=\"Private\" />" +
      "       <Email>peter@theill.com</Email>" +
      "       <Name>Personal E-mail</Name>" +
      "     </EmailAddress>" +
      "     <EmailAddress>" +
      "      <Cat Ref=\"Public\" />" +
      "       <Email>pt@commanigy.com</Email>" +
      "       <Name>Business E-mail</Name>" +
      "     </EmailAddress>" +
      "   </Contact>" +
      "   <Contact Id=\"mb\">" +
      "     <Name>" +
      "       <GivenName>Morten</GivenName>" +
      "       <SurName>Bartvig</SurName>" +
      "     </Name>" +
      "     <EmailAddress>" +
      "       <Email>bartvig@gmail.com</Email>" +
      "       <Name>Personal E-mail</Name>" +
      "     </EmailAddress>" +
      "   </Contact>" +
      "</Contacts>"));
  }


    public void testEvaluate() throws IOException {
        QFilter a = new QFilter();

        Role role = null;
        Node[] result;

        // TODO finalize tests
//          result = a.evaluate(null, RoleTemplateMethodType.query, "/", role);
//          result = a.evaluate(null, RoleTemplateMethodType.query, "//Contact", role);
//          result = a.evaluate(null, RoleTemplateMethodType.query, "//Name", role);
//          result = a.evaluate(null, RoleTemplateMethodType.query, "//*", role);

    }

     //Moved from "RoleTest.java"


        // except: $set1[count(.|$set2)!=count($set2)]
    // intersection: $set1[count(.|$set2)=count($set2)]
    public void testQFilterOneInclude() throws XMLException, IOException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "//*")); // $set1
        Scope scope = new Scope(ScopeBase.nil, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        QFilter q = new QFilter();

        Node[] resp = q.evaluate(contacts, RoleTemplateMethodType.query,"/Contacts/*", r)
            ;

//        String resp = r.qFilter(query,RoleTemplateMethodType.query);
//        XMLStoreNavigator nav = new XMLStoreNavigator();
//        XPath xp = new XPath(resp,nav);
//        XNodeSet xnodeset = (XNodeSet)xp.evaluate(new DocNode(n));

        assertEquals(2,resp.length);
        assertEquals("Contact",resp[0].getNodeValue());
        assertEquals(3,((List<Node>)resp[0].getChildren()).size());
        assertEquals("pt",resp[0].getAttribute("Id"));
        assertEquals("mb",resp[1].getAttribute("Id"));
        assertEquals(2,((List<Node>)resp[1].getChildren()).size());
//      System.out.println(xnodeset.toString());

        resp = q.evaluate(contacts, RoleTemplateMethodType.query, "//Name",r);
//        System.out.println(resp.length);
    }

    public void testQFilterMultipleIncludes() throws XMLException, IOException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "/Contacts/Contact[@Id='pt']/Name"));
            // $set1
        shapelist.add(new Shape(ShapeType.include,"/Contacts/Contact[@Id='pt']/
            EmailAddress"));
        Scope scope = new Scope(ScopeBase.nil, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        QFilter q = new QFilter();

        Node[] resp = q.evaluate(contacts,RoleTemplateMethodType.query,"/Contacts",r);

        assertEquals(0,resp.length); // no include policy applies

        Node[] resp2 = q.evaluate(contacts,RoleTemplateMethodType.query,"//EmailAddress",
            r);
        assertEquals(2,resp2.length);

        ArrayList<Node> n = new ArrayList();
        n.addAll(((List<Node>)resp2[0].getChildren()).get(2).getChildren());
        assertEquals ("Personal E-mail",n.get(0).getNodeValue());
        ArrayList<Node> n2 = new ArrayList();
        n2.addAll(((List<Node>)resp2[1].getChildren()).get(2).getChildren());
        assertEquals ("Business E-mail",n2.get(0).getNodeValue());

        Node[] resp3 = q.evaluate(contacts,RoleTemplateMethodType.insert,"//*",r);
        assertEquals(0,resp3.length);
    }

    public void testQFilterIncludeAndExclude() throws XMLException, IOException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "//Name")); // $set1
        shapelist.add(new Shape(ShapeType.exclude,"//EmailAddress"));
        Scope scope = new Scope(ScopeBase.nil, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        QFilter q = new QFilter();

        Node[] resp = q.evaluate(contacts,RoleTemplateMethodType.query,"/Contacts/Contact
            [@Id='mb']/*",r);

        assertEquals(1,resp.length);

        ArrayList<Node> name = new ArrayList();
        name.addAll((List<Node>)resp[0].getChildren());
        assertEquals("Morten",((List<Node>)name.get(0).getChildren()).get(0).getNodeValue
            ());
        assertEquals("Bartvig",((List<Node>)name.get(1).getChildren()).get(0).
            getNodeValue());

        Node[] resp2 = q.evaluate(contacts,RoleTemplateMethodType.query,"/",r);
        assertEquals(0,resp2.length);
    }

    public void testQFilterIncludeAndMultipleExcludes() throws XMLException, IOException
        {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "//*")); // $set1
        shapelist.add(new Shape(ShapeType.exclude,"/Contacts/Contact[@Id='pt']/Name"));
        shapelist.add(new Shape(ShapeType.exclude,"/Contacts/Contact[@Id='pt']/
            EmailAddress"));
        Scope scope = new Scope(ScopeBase.nil, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        QFilter q = new QFilter();

        // try to get root document
        Node[] resp = q.evaluate(contacts,RoleTemplateMethodType.query,"/Contacts",r);

        assertEquals(0,resp.length);

        // try to get all child nodes from root (gets only a)
        resp = q.evaluate(contacts,RoleTemplateMethodType.query,"/Contacts/*",r);

        assertEquals(1,resp.length);
        assertEquals("mb",resp[0].getAttribute("Id").toString());
    }

    /*
    public void testRawQFilterOneInclude() throws XMLException, IOException,
            XPathException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "/root//*")); // $set1
        Scope scope = new Scope(ScopeBase.nil, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        // allow1 = "//* | /root//*";
        // deny1 = "()";
        // allow except deny
        // String allowExDeny1 = "//* | /root//*";
        String query1 = "/foo_xpath";
        String expQuery1 = "/foo_xpath[count(.|/root//*)=count(/root//*)]";
        String resp1 = r.qFilter(query1, RoleTemplateMethodType.query);
        assertEquals(expQuery1, resp1);

        // allow2 = "()"
        // deny2 = "()";
        // String allowExDeny2 = "";
        String query2 = "/foo_xpath";
        String expQuery2 = "";
        String resp2 = r.qFilter(query2, RoleTemplateMethodType.insert);
        assertEquals(expQuery2, resp2);

        // allow3 = "//* | /root//*";
        // deny2 = "()";
        String query3 = "/";
        // String allowExDeny3 = "//* | /root//*";
        String expQuery3 = "/node()[count(.|/root//*)=count(/root//*)]";
        String resp3 = r.qFilter(query3, RoleTemplateMethodType.query);
        assertEquals(expQuery3, resp3);
    }

    public void testRawQFilterMultipleIncludes() {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "/root")); // $set1
        shapelist.add(new Shape(ShapeType.include, "/bar/foo"));
        Scope scope = new Scope(ScopeBase.t, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        // allow1 = "//* | /root//* | /bar/foo//*";
        // deny1 = "()";
        // allow except deny
        // String allowExDeny1 = "//* | /root//* | /bar/foo//*";
        String query1 = "/foo_xpath";
        String expQuery1 = "/foo_xpath[count(.|//* | /root | /bar/foo)=count(//* | /root
            | /bar/foo)]";
        String resp1 = r.qFilter(query1, RoleTemplateMethodType.query);
        assertEquals(expQuery1, resp1);

        // allow2 = "()"
        // deny2 = "()";
        // String allowExDeny2 = "";
        String query2 = "/foo_xpath";
        String expQuery2 = "";
        String resp2 = r.qFilter(query2, RoleTemplateMethodType.insert);
        assertEquals(expQuery2, resp2);
    }

    public void testRawQFilterIncludeExclude() throws XNIException, IOException,
            XMLException, XPathException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "/root"));
        shapelist.add(new Shape(ShapeType.exclude, "/root/bar/foo"));
        Scope scope = new Scope(ScopeBase.t, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        // allow1 = "//* | /root//*";
        // deny1 = "/bar/foo";
        // allow except deny
        String allowExDeny1 = "(//* | /root)[count(.|/root/bar/foo/ancestor-or-self::*)!=
            count(/root/bar/foo/ancestor-or-self::*)]";
        String query1 = "/foo_xpath";
        String expQuery1 = "/foo_xpath[count(.|" + allowExDeny1 + ")=count("
                + allowExDeny1 + ")]";
        String resp1 = r.qFilter(query1, RoleTemplateMethodType.query);
         assertEquals(expQuery1, resp1);

        // allow2 = "()"
        // deny2 = "()";
        // String allowExDeny2 = "";
        String query2 = "/foo_xpath";
        String expQuery2 = "";
        String resp2 = r.qFilter(query2, RoleTemplateMethodType.insert);
        assertEquals(expQuery2, resp2);

        // System.out.println(resp1);
        // System.out.println(resp2);

    }

    public void testFilter() throws XMLException, IOException, XPathException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "/root/*[not(./g)]"));
        // shapelist.add(new Shape(ShapeType.include, "/root/a"));
        // shapelist.add(new Shape(ShapeType.include, "/root/c"));
        // shapelist.add(new Shape(ShapeType.exclude, "/root/b/g"));
        Scope scope = new Scope(ScopeBase.nil, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        String out = "";
        Node node = SAXBuilder.build(new StringReader("<root>" + "<a><g/></a>"
                + "<b>" + "<g/>" + "</b>" + "<c/>" + "</root>"));
        XMLStoreNavigator nav = new XMLStoreNavigator();
        String queryQ1 = r.qFilter("/root/*", RoleTemplateMethodType.query);
        // System.out.println(queryQ);
        XPath xp = new XPath(queryQ1, nav);
        XObject obj = xp.evaluate(new DocNode(node));

        if (obj instanceof XNodeSet) {
            XNodeSet ns = (XNodeSet) obj;
            Iterator it = ns.iterator();
            while (it.hasNext()) {
                Node c = (Node) it.next();
                out += "\n--\n";
                out += c.toString();
            }
        } else {
            out = obj.toString();
        }

//      System.out.println(out);
    }

    public void testXpathCount() throws XMLException, IOException,
            XPathException {
        Node node = SAXBuilder.build(new StringReader(
                "<a><b id=\"1\" /><b id=\"2\" /></a>"));
        Navigator nav = new XMLStoreNavigator();
        XPath xp = new XPath("count(//b)", nav);
        DocNode docnode = new DocNode(node);
        XNumber xnr = (XNumber) xp.evaluate(docnode);
        assertEquals(2, xnr.intValue());

        xp = new XPath("count(/a/b[@id='1'] | /a/b[@id='2'])", nav);
        xnr = (XNumber) xp.evaluate(docnode);
        assertEquals(2, xnr.intValue());

        String out = "";
        xp = new XPath("/a/b[@id='1'] | /a/b[@id='1']", nav);
        XObject obj = xp.evaluate(docnode);
        // System.out.println("Found:");
        if (obj instanceof XNodeSet) {
            XNodeSet ns = (XNodeSet) obj;
            Iterator it = ns.iterator();
            while (it.hasNext()) {
                Node c = (Node) it.next();
                out += c.toString();
                // System.out.println(c.toString());
            }
        } else {
            out = obj.toString();
            // System.out.println(obj.toString());
        }

        assertEquals("<ELEMENT:b id=\"1\"/>", out);

    }

    public void testSSS() throws XMLException, IOException, XPathException {
        List<Shape> shapelist = new ArrayList<Shape>();
        shapelist.add(new Shape(ShapeType.include, "/root"));
        shapelist.add(new Shape(ShapeType.exclude, "//b"));
        Scope scope = new Scope(ScopeBase.t, "foo_scope");
        scope.setShapes(shapelist);
        RoleTemplateMethod rtm = new RoleTemplateMethod(
                RoleTemplateMethodType.query, scope);
        List<RoleTemplateMethod> rtl = new ArrayList<RoleTemplateMethod>();
        rtl.add(rtm);
        RoleTemplate rt = new RoleTemplate("foo_roletemplate");
        rt.setMethods(rtl);
        Subject s = new Subject("foo_id");
        Role r = new Role(rt, s);

        String out = "";
        Node node = SAXBuilder.build(new StringReader(
                    "<root>"
                +       "<a/>"
                +       "<b/>"
                +       "<c/>"
                +   "</root>"));
        XMLStoreNavigator nav = new XMLStoreNavigator();
        String queryQ1 = r.qFilter("//*", RoleTemplateMethodType.query);
//      System.out.println(queryQ1);
        XPath xp = new XPath(queryQ1, nav);
        XNodeSet obj = (XNodeSet)xp.evaluate(new DocNode(node));
//      System.out.println(obj.toString());
    }







     */


}
```

E.3 org.planx.authx.store

E.3.1 AuthLocalXmlStoreTest.java

```
package org.planx.authx.store;

import java.io.File;
import java.io.IOException;
import java.io.StringReader;
import java.security.AccessControlException;

import junit.framework.TestCase;

import org.planx.authx.Subject;
import org.planx.authx.filter.QFilter;
import org.planx.io.Locator;
import org.planx.xmlstore.DocNode;
import org.planx.xmlstore.NameServer;
import org.planx.xmlstore.NameServerException;
import org.planx.xmlstore.Node;
import org.planx.xmlstore.Reference;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;
import org.planx.xmlstore.koala.nodes.DVMNodeFactory;
import org.planx.xmlstore.references.LocalLocator;
import org.planx.xmlstore.references.ValueReference;
import org.planx.xmlstore.stores.DistributedXMLStore;
import org.planx.xmlstore.stores.LocalXMLStore;
import org.planx.xmlstore.stores.TranslatorXMLStore;
import org.planx.xpath.Navigator;
import org.planx.xpath.XPath;
import org.planx.xpath.object.XObject;

public class AuthLocalXmlStoreTest extends TestCase {

    protected void setUp() throws Exception {
        super.setUp();

        // remove created store files
        new File("test-store.data").delete();
        new File("test-store.free").delete();
        new File("test-store.localns.map").delete();
    }

    protected void tearDown() throws Exception {
        super.tearDown();

        // remove created store files
        new File("test-store.data").delete();
        new File("test-store.free").delete();
        new File("test-store.localns.map").delete();
    }

    public void testSaveLoadXmlStore() throws Exception {
        // 1. make new LocalXMLStore, save a node and bind it to a name
        LocalXMLStore xmlStore = new LocalXMLStore("test-store");
        NameServer<LocalLocator> nameServer = xmlStore.getNameServer();
        Node n = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));
        LocalLocator r = xmlStore.save(n);
        // System.out.println(r.toString());
        assertNotNull(r);

        nameServer.bind("foo.xml", r);
        xmlStore.close();

        // 2. Load old node from xmlstore via a lookup and determine if node is
        // correct
        xmlStore = new LocalXMLStore("test-store");
        nameServer = xmlStore.getNameServer();
        Node nShouldBe = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));

        LocalLocator r2 = nameServer.lookup("foo.xml");
        // System.out.println(r2.toString());

        assertNotNull(r2);

        n = xmlStore.load(r2);
        assertNotNull(n);
        assertTrue(nShouldBe.equals(n));
        xmlStore.close();

        // 3. open xmlstore with AuthXMLStore and load the node and determine if
        // it's correct
        AuthXMLStore authXmlStore = new AuthXMLStore(new LocalXMLStore(
                "test-store"), new Subject("fb"), new QFilter());
        AuthNameServer authNameServer = (AuthNameServer) authXmlStore
                .getNameServer();
        Node nAuthShouldBe = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));

        AuthReference r3 = authNameServer.lookup("foo.xml");
        assertNotNull(r3);

        Node n2 = authXmlStore.load(r3);
        assertNotNull(n2);
        assertTrue(nAuthShouldBe.equals(n2));
        authXmlStore.close();
    }

    public void testSaveAuthXmlStore() throws Exception, IOException,
            XMLException, NameServerException {
        // 1. Save node in AuthXMLStore
        AuthXMLStore xmlStore = new AuthXMLStore(
                new LocalXMLStore("test-store"), new Subject("fb"),
                new QFilter());
        AuthNameServer authNameServer = (AuthNameServer) xmlStore
                .getNameServer();
        Node n = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));
        AuthReference r = xmlStore.save(n);

        // System.out.println(r.toString());
        assertNotNull(r);

        // n = xmlStore.load(r);
        // System.out.println(n.toString());

        authNameServer.bind("bar.xml", r);
        xmlStore.close();

        // 2. Load node in LocalXMLStore
        LocalXMLStore localXmlStore = new LocalXMLStore("test-store");
        NameServer<LocalLocator> nameServer = localXmlStore.getNameServer();
        Node nShouldBe = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));

        LocalLocator r2 = nameServer.lookup("bar.xml");
        assertNotNull(r2);

        Node n2 = localXmlStore.load(r2);
        assertNotNull(n2);
        assertTrue(nShouldBe.equals(n2));
        localXmlStore.close();

        // 3. Load node in AuthXMLStore
        AuthXMLStore authXmlStore = new AuthXMLStore(new LocalXMLStore(
                "test-store"), new Subject("fb"), new QFilter());
        authNameServer = (AuthNameServer) authXmlStore.getNameServer();

        AuthReference r3 = authNameServer.lookup("bar.xml");
        // System.out.println(r3.toString());
        assertNotNull(r3);

        n = authXmlStore.load(r3);
        assertNotNull(n);

        nShouldBe = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));
        assertTrue(nShouldBe.equals(n));
        authXmlStore.close();
    }

    public void testAuthXmlStoreWrongUser() throws Exception {
        // 1. Save node foo.xml in xmlstore with user fb
        AuthXMLStore xmlStore = new AuthXMLStore(
                new LocalXMLStore("test-store"), new Subject("fb"),
                new QFilter());
        AuthNameServer authNameServer = (AuthNameServer) xmlStore
                .getNameServer();
        Node n = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));
        AuthReference r = xmlStore.save(n);
```

```java
        assertNotNull(r);
        authNameServer.bind("foo.xml", r);

        xmlStore.close();

        // 2. Try to open node foo.xml with invalid user foobar
        xmlStore = new AuthXMLStore(new LocalXMLStore("test-store"),
                new Subject("foobar"), new QFilter());
        authNameServer = (AuthNameServer) xmlStore.getNameServer();
        AuthReference r2 = authNameServer.lookup("foo.xml");
        assertNotNull(r2);

        Node n2 = null;
        try {
            n2 = xmlStore.load(r2);
        }
        catch (AccessControlException e) {
            assertTrue(true);
            // e.printStackTrace();
        }
        catch (IOException e) {
            assertTrue(false);
            e.printStackTrace();
        }

        assertNull(n2);
        xmlStore.close();
    }

    public void testRebind() throws Exception {
        // 1. Test rebind with valid user
        AuthXMLStore xmlStore = new AuthXMLStore(
                new LocalXMLStore("test-store"), new Subject("fb"),
                new QFilter());
        AuthNameServer authNameServer = (AuthNameServer) xmlStore
                .getNameServer();
        Node n = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));
        AuthReference r = xmlStore.save(n);
        assertNotNull(r);
        authNameServer.bind("foobar.xml", r);

        Node n2 = SAXBuilder.build(new StringReader(
                "<foo id='11'><foobar id='44'/></foo>"));
        AuthReference r2 = xmlStore.save(n2);
        try {
            authNameServer.rebind("foobar.xml", r, r2);
        }
        catch (Exception e) {
            assertTrue(false);
            // e.printStackTrace();
        }

        xmlStore.close();

        xmlStore = new AuthXMLStore(new LocalXMLStore("test-store"),
                new Subject("fb"), new QFilter());
        authNameServer = (AuthNameServer) xmlStore.getNameServer();
        AuthReference r3 = authNameServer.lookup("foobar.xml");
        Node n3 = xmlStore.load(r3);

        assertTrue(n2.contentEquals(n3));
        xmlStore.close();
    }

    public void testSmth() throws IOException, XMLException {
        // AuthXMLStore xmlStore = new AuthXMLStore(
        // new LocalXMLStore("test-store"), new Subject("fb"));

        // AuthXMLStore xmlStore = new AuthXMLStore(new DistributedXMLStore(new
        // TranslatorXMLStore
        // (new LocalXMLStore("test-store")),
        // 9000 /* UDP port for routing */,
        // 9000 /* TCP port for data */,
        // null /* no bootstrap peer */));

        DistributedXMLStore xmlStore = new DistributedXMLStore(
                new TranslatorXMLStore(new LocalXMLStore("test-store")),
                9000 /* UDP port for routing */,
                9000 /* TCP port for data */, null /* no bootstrap peer */);

        NameServer nameServer = (NameServer) xmlStore.getNameServer();
        Node n = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));
        Node n2 = SAXBuilder.build(new StringReader(
                "<foo><bar id='1'/><bar id='2'/></foo>"));

        ValueReference r = xmlStore.save(n);
        ValueReference r2 = xmlStore.save(n2);

        // r.
        System.out.println(r.equals(r2));

        Node node = SAXBuilder.build(new StringReader(
                "<a><b id='1'/><b id='2'/><b id='3'/></a>"));
        Node child = SAXBuilder.build(new StringReader("<c/>"));

        Node newN = DVMNodeFactory.instance().insertChild(node, 2, child);
        System.out.println(newN.toString());
        newN = DVMNodeFactory.instance().removeChild(newN, 1);
        System.out.println(newN.toString());
        xmlStore.close();
    }
}
```

E.3.2 AuthNameServerTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

import java.io.File;
import java.io.IOException;
import java.io.StringReader;

import org.planx.authx.Subject;
import org.planx.authx.filter.QFilter;
import org.planx.xmlstore.NameAlreadyBoundException;
import org.planx.xmlstore.NameServerException;
import org.planx.xmlstore.Node;
import org.planx.xmlstore.XMLException;
import org.planx.xmlstore.input.SAXBuilder;
import org.planx.xmlstore.koala.nameserver.LocalNameServer;
import org.planx.xmlstore.koala.nodes.DVMNodeFactory;
import org.planx.xmlstore.stores.LocalXMLStore;

import junit.framework.TestCase;
```

```

/**
 * @author pt
 *
 */
public class AuthNameServerTest extends TestCase {
    private AuthXMLStore xmlStore;

    /*
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception {
        super.setUp();

        Node n = SAXBuilder
                .build(new StringReader(
                        "<contacts><contact>Peter Theill</contact><contact>Morten Bartvig
                            </contact></contacts>"));

        Subject subject = new Subject("foo");

        xmlStore = new AuthXMLStore(new LocalXMLStore("myTestStore"), subject,
                new QFilter());
        xmlStore.getNameServer().bind("myContacts", xmlStore.save(n));
    }

    /*
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception {
        super.tearDown();

        if (xmlStore != null) {
            xmlStore.close();
            xmlStore = null;
        }

        // remove created store files
        new File("myTestStore.data").delete();
        new File("myTestStore.free").delete();
        new File("myTestStore.localns.map").delete();
    }

    /*
     * Test method for 'org.planx.authx.store.AuthNameServer.lookup(String)'
     */
    public void testLookupString() throws IOException, NameServerException {
        AuthReference r = xmlStore.getNameServer().lookup("myContacts");
        assertNotNull(r);
        assertNotNull(r.getCreatorReference());
        assertNotNull(r.getRoleMapReference());
        assertNotNull(r.getRoleListReference());
        assertNotNull(r.getContentReference());
        assertNotSame(r.getCreatorReference(), r.getRoleMapReference());
        assertNotSame(r.getCreatorReference(), r.getRoleListReference());
        assertNotSame(r.getCreatorReference(), r.getContentReference());
    }

    /*
     * Test method for 'org.planx.authx.store.AuthNameServer.lookup(String,
     * AuthReferenceDocumentType)'
     */
    public void testLookupStringAuthReferenceDocumentType() throws IOException,
            NameServerException {
        AuthReference r = null;
```

```java

        AuthNameServer ns = (AuthNameServer) xmlStore.getNameServer();

        r = ns.lookup("myContacts", AuthReferenceDocumentType.content);
        assertNotNull(r);

        r = ns.lookup("myContacts", AuthReferenceDocumentType.roleMap);
        assertNotNull(r);

        r = ns.lookup("myContacts", AuthReferenceDocumentType.roleList);
        assertNotNull(r);
    }

    /*
     * Test method for 'org.planx.authx.store.AuthNameServer.bind(String,
     * AuthReference)'
     */
    public void testBind() throws IOException, NameServerException,
            XMLException {
        AuthNameServer ns = (AuthNameServer) xmlStore.getNameServer();
        AuthReference r = ns.lookup("myContacts");

        Node n = SAXBuilder.build(new StringReader("<unused />"));

        try {
            ns.bind("myContacts", xmlStore.save(n));
            assertTrue(false);
        }
        catch (NameAlreadyBoundException x) {
            // assertEquals("Name myContacts Creator already bound to " +
            // r.getCreatorReference().toString(), x.getMessage());
        }
    }

    /*
     * Test method for 'org.planx.authx.store.AuthNameServer.rebind(String,
     * AuthReference, AuthReference)'
     */
    public void testRebind() throws IOException, NameServerException,
            XMLException {
        AuthNameServer ns = (AuthNameServer) xmlStore.getNameServer();
        AuthReference r = ns.lookup("myContacts");

        Node n = SAXBuilder.build(new StringReader(
                "<contact>Sergey Brin</contact>"));
        n = DVMNodeFactory.instance().insertChild(xmlStore.load(r), 0, n);

        ns.rebind("myContacts", r, xmlStore.save(n));
        assertTrue(true);
    }
}
```

E.3.3 AuthReferenceTest.java

```java
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
public class AuthReferenceTest extends TestCase {

    /*
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception {
        super.setUp();
    }

    /*
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception {
        super.tearDown();
    }

    /*
     * Test method for 'org.planx.authx.store.AuthReference.AuthReference()'
     */
    public void testAuthReference() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for 'org.planx.authx.store.AuthReference.getCreator()'
     */
    public void testGetCreator() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for 'org.planx.authx.store.AuthReference.getRoleMap()'
     */
    public void testGetRoleMap() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for
     * 'org.planx.authx.store.AuthReference.setRoleMap(Reference)'
     */
    public void testSetRoleMap() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for
     * 'org.planx.authx.store.AuthReference.getContentReference()'
     */
    public void testGetContentReference() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for
     * 'org.planx.authx.store.AuthReference.setContentReference(Reference)'
     */
    public void testSetContentReference() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for 'org.planx.authx.store.AuthReference.getRoleList()'
     */
    public void testGetRoleList() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for
     * 'org.planx.authx.store.AuthReference.setRoleList(Reference)'
     */
    public void testSetRoleList() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for 'java.lang.Object.toString()'
     */
    public void testToString() {
        // TODO Auto-generated method stub

    }

}
```

E.3.4 AuthXMLStoreTest.java

```
/**
 * Authorization in XML Store
 *
 */
package org.planx.authx.store;

import java.io.File;
import java.io.IOException;
import java.io.StringReader;
import java.security.AccessControlException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import org.planx.authx.*;
import org.planx.authx.filter.QFilter;
import org.planx.xmlstore.*;
import org.planx.xmlstore.input.SAXBuilder;
import org.planx.xmlstore.koala.nodes.DVMHandler;
import org.planx.xmlstore.koala.nodes.DVMNodeFactory;
import org.planx.xmlstore.references.LocalLocator;
import org.planx.xmlstore.stores.LocalXMLStore;
import org.planx.xpath.*;
import org.planx.xpath.object.*;

import junit.framework.TestCase;

/**
 * @author pt
 *
 */
```

```
public class AuthXMLStoreTest extends TestCase {
    private AuthXMLStore xmlStore;

    /*
     * @see TestCase#setUp()
     */
    protected void setUp() throws Exception {
        super.setUp();

        // remove created store files
        new File("test-store.data").delete();
        new File("test-store.free").delete();
        new File("test-store.localns.map").delete();

        xmlStore = new AuthXMLStore(new LocalXMLStore("test-store"),
                new Subject("fb"), new QFilter());
    }


    /*
     * @see TestCase#tearDown()
     */
    protected void tearDown() throws Exception {
        super.tearDown();

        xmlStore.close();
        xmlStore = null;

        // remove created store files
        new File("test-store.data").delete();
        new File("test-store.free").delete();
        new File("test-store.localns.map").delete();
    }


    /*
     * Test method for 'org.planx.authx.store.AuthXMLStore.AuthXMLStore(R)'
     */
    public void testAuthXMLStoreAccessControl() throws IOException, Exception {
        // delete existing store files
        new File("mystore.data").delete();
        new File("mystore.free").delete();
        new File("mystore.localns.map").delete();

        String bindName = "Peter Theill's Contacts";

        // Step 1: initialize a document with basic authorization by owner
        // 'pt' i.e. only this user is able to access document

        // load xml store with no authorization
        AuthXMLStore xmlStore = new AuthXMLStore(new LocalXMLStore("mystore"), new
            Subject("pt"));

        // create initial document structure
        Node myDocument = SAXBuilder.build(new StringReader("<Contacts><Contact Id=\"pt
            \"><Name><Title>Mr.</Title><GivenName>Peter</GivenName><SurName>Theill</
            SurName></Name></Contact><Contact Id=\"mb\"><Name><GivenName>Morten</
            GivenName><SurName>Bartvig</SurName></Name></Contact></Contacts>"));

        // store this document into xml store
        AuthReference myDocumentRef = xmlStore.save(myDocument);
        assertNotNull(myDocumentRef);

        // bind reference so it's possible to lookup this document later
        xmlStore.getNameServer().bind(bindName, myDocumentRef);

        // close store and free resources
        xmlStore.close();
        xmlStore = null;


        // Step 2: user 'mb' tries to load data but fails caused by no valid
        // role can be found

        // load store again with another user
        xmlStore = new AuthXMLStore(new LocalXMLStore("mystore"), new Subject("mb"));

        // lookup reference to previously stored document
        myDocumentRef = xmlStore.getNameServer().lookup(bindName);
        assertNotNull(myDocumentRef);

        try {
            myDocument = null;

            // try to load reference and catch exception thrown
            myDocument = xmlStore.load(myDocumentRef);
            assertTrue(false);
        }
        catch (AccessControlException x) {
            // exception is expected
            assertEquals(x.getMessage(), "Failed to lookup role for mb");
        }

        // ensure no document was loaded
        assertNull(myDocument);

        // close store and free resources
        xmlStore.close();
        xmlStore = null;


        // Step 3: document owner (i.e. user 'pt') loads role map document
        // and adds new scope and role template for a limited reader

        // load store with document owner
        xmlStore = new AuthXMLStore(new LocalXMLStore("mystore"), new Subject("pt"));

        // lookup reference to role map
        AuthReference roleMapRef = xmlStore.getNameServer().lookup(bindName,
            AuthReferenceDocumentType.roleMap);
        assertNotNull(roleMapRef);

        // load role map
        Node roleMap = xmlStore.load(roleMapRef);
        assertNotNull(roleMap);
        assertEquals(roleMap.getChildren().size(), 3); // one scope and two role
            templates

        // build new scope for 'limited read'
        Scope roleMapScope = new Scope(ScopeBase.nil, "contacts-with-title");
        List<Shape> roleMapShapes = new ArrayList<Shape>();
        roleMapShapes.add(new Shape(ShapeType.include, "//Contact[Name/Title]"));
        roleMapScope.setShapes(roleMapShapes);
        Node scope = SAXBuilder.build(new StringReader(roleMapScope.toXml()));

        // add scope to role map
        roleMap = DVMNodeFactory.instance().insertChild(roleMap, 0, scope);
        assertEquals(roleMap.getChildren().size(), 4); // two scopes and two role
            templates

        // build new role template
        RoleTemplate roleMapRoleTemplate = new RoleTemplate("rt3");
        List<RoleTemplateMethod> roleMapRoleTemplateMethods = new ArrayList<
            RoleTemplateMethod>();
        roleMapRoleTemplateMethods.add(new RoleTemplateMethod(RoleTemplateMethodType.
            query, roleMapScope));
        roleMapRoleTemplate.setMethods(roleMapRoleTemplateMethods);
        Node roleTemplate = SAXBuilder.build(new StringReader(roleMapRoleTemplate.toXml()
            ));

        // add role template to role map
        roleMap = DVMNodeFactory.instance().insertChild(roleMap, 0, roleTemplate);
        assertEquals(roleMap.getChildren().size(), 5); // two scopes and three role
            templates

        // explicit store 'updated' role map to store
        AuthReference updatedRoleMapRef = xmlStore.save(roleMap);

        // update reference to role map for document
        xmlStore.getNameServer().rebind(bindName, roleMapRef, updatedRoleMapRef,
            AuthReferenceDocumentType.roleMap);


        // Step 4: document owner (i.e. user 'pt') loads role list document
        // and adds user 'mb' as a limited 'reader' (rt3 indicates a reader)

        // lookup reference to role list
        AuthReference roleListRef = xmlStore.getNameServer().lookup(bindName,
            AuthReferenceDocumentType.roleList);
        assertNotNull(roleListRef);

        // load role list
        Node roleList = xmlStore.load(roleListRef);
        assertNotNull(roleList);
        assertEquals(roleList.getChildren().size(), 1);

        // build new role for user 'mb'
        Node role = SAXBuilder.build(new StringReader("<role roleTemplateRef=\"rt3\"><
            subject userId=\"mb\" /></role>"));

        // add role to role list
        roleList = DVMNodeFactory.instance().insertChild(roleList, 0, role);
        assertEquals(roleList.getChildren().size(), 2);

        // explicit store 'updated' role list to store
        AuthReference updatedRoleListRef = xmlStore.save(roleList);

        // update reference to role list for document
        xmlStore.getNameServer().rebind(bindName, roleListRef, updatedRoleListRef,
            AuthReferenceDocumentType.roleList);

        // reload role list reference
        roleListRef = xmlStore.getNameServer().lookup(bindName, AuthReferenceDocumentType
            .roleList);
        assertNotNull(roleListRef);

        // load updated role list node
        roleList = xmlStore.load(roleListRef);
        assertNotNull(roleList);
        assertEquals(roleList.getChildren().size(), 2);

        // close store and free resources
        xmlStore.close();
        xmlStore = null;

```

```java

        // Step 5: user 'mb' loads data now causing node to be read since
        // authorization has been updated

        // load store again with another user
        xmlStore = new AuthXMLStore(new LocalXMLStore("mystore"), new Subject("mb"));

        // lookup reference to previously stored document
        myDocumentRef = xmlStore.getNameServer().lookup(bindName);
        assertNotNull(myDocumentRef);

        // try to load reference and catch exception thrown
        Node[] contacts = xmlStore.load(myDocumentRef, "//Contact");
        assertNotNull(contacts);
        assertEquals(contacts.length, 1);
        assertEquals(contacts[0].getChildren().size(), 1); // one 'Name' node
        assertEquals(contacts[0].getChildren().get(0).getChildren().size(), 3); // one '
            Title', one 'GivenName' and one 'SurName' node

        // close store and free resources
        xmlStore.close();
        xmlStore = null;

    }


    /*
     * Test method for 'org.planx.authx.store.AuthXMLStore.AuthXMLStore(R,
     * Principal)'
     */
    public void testAuthXMLStoreRPrincipal() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for 'org.planx.authx.store.AuthXMLStore.close()'
     */
    public void testClose() {
        // TODO Auto-generated method stub

    }

    /*
     * Test method for 'org.planx.authx.store.AuthXMLStore.getNameServer()'
     */
    public void testGetNameServer() {
        // TODO Auto-generated method stub

    }

    public void testXPath() throws IOException, XMLException,
            NameServerException, XPathException {

        new File("local-store.data").delete();
        new File("local-store.free").delete();
        new File("local-store.localns.map").delete();

        LocalXMLStore a = new LocalXMLStore("local-store");

        Node n = SAXBuilder.build(new StringReader("<a><b><c /></b></a>"));
        Node child = SAXBuilder
                .build(new StringReader("<c><d id=\"2\" /></c>"));

        LocalLocator ll = a.getNameServer().lookup("test-a");
        if (ll == null) {
            ll = a.save(n);
            // System.out.println(ll.get().id().toString());

            a.getNameServer().bind("test-a", ll);

            ll = a.getNameServer().lookup("test-a");
            // System.out.println(ll.get().id().toString());
        }

        n = a.load(ll);

        XMLStoreNavigator nav = new XMLStoreNavigator();
        XPath xp = new XPath("//b", nav);
        XObject obj = xp.evaluate(new DocNode(n));
        if (obj instanceof XNodeSet) {
            XNodeSet ns = (XNodeSet) obj;
            Iterator it = ns.iterator();
            while (it.hasNext()) {
                Node c = (Node) it.next();
                // System.out.println(c);
                c = DVMNodeFactory.instance().insertChild(c, 0, child);
                LocalLocator ll2 = a.save(c);
                // System.out.println(c);
                a.getNameServer().rebind("test-a", ll, ll2);
            }
        }
        else {
            // System.out.println(obj);
        }

        ll = a.getNameServer().lookup("test-a");
        Node fullNode = a.load(ll);
        //System.out.println(fullNode);

        a.close();
    }

    /*
     * Test method for 'org.planx.authx.store.AuthXMLStore.load(AuthReference)'
     */
    public void testLoad() throws IOException, XMLException {
        Node n = SAXBuilder.build(new StringReader("<a><b /></a>"));
        AuthReference reference = xmlStore.save(n);

        n = xmlStore.load(reference);
        assertNotNull(n);
        assertEquals(1, n.getChildren().size());
    }

    public void testLoadQuery() throws IOException, XMLException {
        Node n = SAXBuilder.build(new StringReader("<a><b /><b /><b /></a>"));
        AuthReference reference = xmlStore.save(n);

        Node[] result = xmlStore.load(reference, "/");
        assertNotNull(result);
        assertEquals(1, result.length);

        result = xmlStore.load(reference, "//b");
        assertNotNull(result);
        assertEquals(3, result.length);

        result = xmlStore.load(reference, "/b");
        assertNotNull(result);
        assertEquals(0, result.length);
```

```java
    }

    /*
     * Test method for 'org.planx.authx.store.AuthXMLStore.save(Node)'
     */
    public void testSave() throws IOException, XMLException {
        try {
            xmlStore.save(null);
            fail("Failed to throw expected exception");
        }
        catch (NullPointerException npe) {
            // this is the expected exception to be thrown
        }

        Node n = SAXBuilder.build(new StringReader("<a><b /></a>"));
        AuthReference reference = xmlStore.save(n);
        assertNotNull(reference);
        assertNotNull(reference.getContentReference());
    }

    /*
     * Test method for 'java.lang.Object.toString()'
     */
    public void testToString() {
        // TODO Auto-generated method stub

    }

}
```