
x.core and x.click components

MASTER THESIS

INFORMATION SYSTEM GROUP
UNIVERSITY OF FRIBOURG, SWITZERLAND
Student: Amos Brocco
Email: amos.brocco@unifr.ch

October 21., 2005

Supervisors:
Prof. Dr. Andreas Meier
Nicolas Werro

Make everything as simple as possible, but not simpler. (Albert Einstein)

# Abstract

This master thesis studies a solution for the secure long term storage of data. The increasing quantity of sensible data in today's information society, makes security and protection of personal data a key point for its usage. Storing data usually involves the use of databases that only grant a certain level of access security; simple login passwords are not enough to guarantee data protection.

Long term information storage is a problem faced by many companies that have old and unused data still stored in their databases. This causes overhaul, slows operations and does not assure a sufficient protection of data. As current regulations obligate firms to store data for a certain period, it is important that beside database systems companies have alternative long term storage applications. Viable solutions should be based on standardized format to ensure longevity and accessibility of information, and should provide data protection by mean proven cryptographic algorithms.

The proposed framework provides a long term secure XML data storage solution designed on a two-tier architecture, and composed of three components: x.core (the server application), x.click (the graphical user interface) and x.mill (a database migration tool integrated within the graphical user interface). Data storage is natively based on the XML format by using a DOM and security is provided by mean of the XML Encryption format. The application also offers querying and updating languages such as XPath and XUpdate.

To shield users from manually performing cryptographic operations on data, and to ease database querying and modification inside encrypted portions of data, we advocate the use of a transparent encryption and decryption process which automatically allows the user to directly operate on ciphered information provided it has the right to.

.

# Acknowledgments

Thanks to Andrea Ghirlanda, Prof. Dr. Andreas Meier, Nicolas Werro, my mother and my father for their patience and my dog Laika. This project is dedicated to a very special person...

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

## 1.1 Chapter overview

This chapter introduces the aspects that motivate the research for a long term data storage solution, along with some real world examples; next the goals and requirements for a viable way to work out this problem are discussed. At the end of the chapter typographic conventions and some general notes about this document are found.

## 1.2 Motivation

The increasing quantity of sensible data in today's information society, makes security and protection of personal data a key point for its safe day-to-day usage. Storing data usually involves the use of databases that only grant a certain level of access security; simple login passwords are not enough to guarantee data protection, furthermore they don't provide any method for data authentication: for example, a person with access to the database can modify data without restriction. Sometimes it is necessary to lower security barriers in favor of ease of access, but if the goal is long term data storage, security could be better managed. In fact future data protection laws will require digital data to be stored for at least five years, and signed and authenticated in order to be legally valid [27].

The fact that long term information storage could become a big problem for business if data is to be protected using encryption software and algorithms, requires the development of valid solutions.

A storage solution should offer a way to preserve information structure and provide abstraction to represents data as close as possible to human's way of managing information in real life. Ways for retrieving contents by mean of querying and search languages as well as methods for updating information or to add new content to existing databases are also important.

The long term storage problem imposes additional constraint to the whole system: the solution must guarantee that information will be still accessible and readable in the future. One of the major problems until now was the lack of a reliable data format assuring all the above properties. This goal is often difficult to achieve with proprietary legacy systems, because they commonly use binary formats strongly tied with the underneath platform. Companies looking for a long term storage solution would certainly not want to find themselves locked in a situation where the software supposed to manage their data is no longer available, or that the format has been deprecated by the software firm that was supposed to mantain it (and support for it is subsequently removed from new versions of the program) because rescuing old information becomes very difficult and costly. This problem is much more evident in the case of binary data, where nothing can assure that the same byte sequences could be read and correctly interpreted by future computing systems. A solution to these problems is relying on open and standardized formats to ensure that data has better chances to survive technological changes [29]. Open standards guarantee freedom from patent threats in the future, while the choice of a standard and internationally recognized data format improves interoperability between applications.

A third problem is security: a secure storage solution must provide not only software protection against unauthorized access but also a physical protection by mean of cryptographic methods. Encrypting data is important because it is the only way to guarantee that sensible data cannot be read in any way by someone gaining unauthorized access to the database system.

In summary, there exist some strong constraint on the choice of a valuable long term data storage system. Some of these exclude the use of proprietary or binary formats and further restrict the choice of a solution.

Fortunately there are data formats that have been designed to be a viable alternative or complement to proprietary formats and give better guarantees of survival: one of these is the eXtended Markup Language, also known as XML [16]. XML is a text-based standard and open format mantained by the World Wide Web Consortium [11], that does not depend on a specific platform, architecture or software, it is already well supported by existing applications and is easily suitable for many uses.

The XML format also offers many standardized extensions to support data encryption (XML ENC) and authentication (XML DIGSIG) [19], meaning that a complete long term data storage solution can be built around it.

Such software application can be used, for example, to backup data coming from old databases: some businesses maintain old and unused database systems only to be able, if needed, to recover data that would unless be unaccessible from modern and supported databases. By having a reliable long term data storage solution, coupled with a suitable database migration tool, the goal of mantaining old data would be easily achieved: provided that the software is based on a open and standardized format, future access to information will not require the presence of old database applications and

systems, lowering maintainance costs while increasing security thanks to available security and data encryption options.

## 1.3 Goals of this project

The goal of this project is to develop a software application that offers a long term data storage solution built around the XML format; the key features that should be available are:

- Data security and authentication by mean of proven encryption methods and digital signatures compatible with the XML syntax.
- Methods for searching and modifying information stored in the database.
- The possibility to access and perform queryies on encrypted fragments of data without requiring explicit deciphering.
- A graphical user interface to access and manage data in a simple and intuitive way.

The application should be developed with a client-server design, and must be able to deal with concurrent access; furthermore the program would be structured as a compounded and extensible framework, to allow simple replacement of existing components and/or the injection of new parts.

## 1.4 Typographic conventions

In this document important terms or definitions are written with a **bold** style. Code fragments or references to objects in the code are written with a monospaced `courier` font.

## 1.5 Copyright and trademarks

This application uses and contains parts (either modified under the terms of the respective licenses or unmodified) of the following software:

- Bouncy Castle Crypto APIs (C) 2000 - 2004 The Legion Of The Bouncy Castle
- JDOM Copyright (C) 2004 Jason Hunter & Brett McLaughlin.
- JAXEN Copyright 2003 (C) The Werken Company. All Rights Reserved.
- JAXUP Copyright 2002-2003 Erwin Bolwidt.
- Base64 Tool (C) Robert Harder

- JArgs (C) 2001-2003 Steve Purcell (C) 2002 Vidar Holen (C) 2002 Michal Ceresna (C) 2005 Ewan Mellor. and others. Please refer to documentation or source code for complete copyright info.

# Chapter 2

# Technical Background

## 2.1 Chapter overview

This chapter describes some technical concepts on which this project is based so that implementation choices described in the following chapters will be clearer. First, an introduction of the XML language is given as it is the choosen data storage format for this application, along with an introduction to syntax definition languages that are used in this document; then an overview of information encryption and authentication is provided, specifically by distinguishing two common algorithm classes. Then some methods and languages that can be used to browse XML documents and to modify and update information inside them are discussed. Finally concurrency issues related to multi-user database system are examined.

## 2.2 XML Format

One of the basic requirements for effective data storage is a way to structure information in a coherent and reliable way, so that retrieving it later in time it is still possible and convenient: here the term "convenient" is used to indicate that data retrival should be possible with not much of hassle. A first step in this direction is done by using a structured data format, which does not impose limitations or limits the applicability of such a solution; as a matter of fact, it is possible to affirm with freedom of doubt that everything a little bit more complex than a plain text file on a computers is already structured data, meaning that conversion to a different structured format is only a matter of different syntax and lexical rules.

Other requirements include maintainability, platform independency, longevity (properties acquired by mean of a simple textual, unicode based, and standardized format [29]) and extensibility. One of the most important structured data formats owning these qualities is XML [20], acronym of eXtensible Markup Language, which is maintained by the World Wide Web Consortium [11], an

international organization responsible for a number of web standards.

Major advantages of XML are it's widely application support, implementation and usage and the fact that it is open format; these are also probably the reasons of such a wide acceptance and adoption of this standard and the reason that there exist so many XML extension that cover almost every need, from graphical formats to music or document storage.

Basically, XML is a set of rules or conventions used to create text and text files containing structured data that is accessible and can be effectively used by large public. In fact XML is perhaps the simplest way to generate and store data on a computer, because it provides a simple syntax with very few syntactical constraints and a wide freedom of application. Being a tag based language similar to HTML (and equal to its successor, XHTML) but providing free named tags and attributes, means that it's syntax is not too much different and it is easy to learn for most computer users; though it is a text based format, XML is only a way of storing data, not a method to present it to the user. For this reason this framework is not intended to be used for normal data access operations, nor to show data to a customer; instead, it can be considered as a middle-ware application that other applications can use to store and retrieve plain or encrypted XML data. This is made possible by the fact programs based on XML can interact with each other in a simple manner, due of the common base format.

In this section, a brief introduction of the XML language is given.

### 2.2.1 XML Fundamentals

This section provides some of the basics of the XML format so that concepts and terms used later in this document can be better understood. XML files are commonly called XML documents and are build up of various entities called elements. An element can contain some information and, like a box, it is delimited by text tags. Elements can be combined one inside each other, without limitation on the number of children an element can have, so that the whole document can be represented like a tree. Suppose, for example, that information about a group of singers, like their name, age and sex, is to be stored. The corresponding XML document could be the one illustrated in Example 2.2.1.

**Example 2.2.1** *A very simple XML document*

```
<singers>
 <singer>
  <name>Avril Lavigne</name>
  <age>21</age>
  <sex>F</sex>
 </singer>
 <singer>
```

```
  <name>Anna Nalick</name>
  <age>20</age>
  <sex>F</sex>
 </singer>
</singers>
```

The first line in the document is called the XML declaration and should always be included: it defines the XML version of the document, which in this case conforms to the 1.0 specification of XML. It is important to note that XML only contains information: so that it doesn't provide information about how to present this information; similarly tag names do not provide contextual information and the fact that tags are labeled in ways that give some information of what their content is, it is up to the application that reads the XML content to interpret the actual information. For example, an XML document describing some objects could contain an element `that` clearly has a different meaning, than that of elements with the same name from Example 2.1.

**Binary data** An XML document cannot contain binary data, so if we want to insert a picture in an XML document we first have to convert binary data to a suitable textual representation such as the Base64 format (also used in email attachments). This restriction is not to be viewed as a negative point: being a text-only format means that it is possible to edit content with any text editor, without needing a specific application. This choice also enforces platform independency.

**Elements** The document of the previous example is composed of a single element named singers. Elements are delimited by start-tags and end-tags, and for the element `singers`, these are `<singers>` and `</singers>`. Everything inside these two tags is said to be the content of the element and it can be plain text (as in this example), also called **character data**, or other elements. Beside character data, every other content is simply referenced as **markup**, carrying no specific information beside the structural one.

**Empty elements** Empty elements, that is elements carrying no content (beside attributes), can be specified using an abbreviated but equivalent syntax for enclosing tags. For example, instead of writing `<status id="puffo"></status>` the form `<status id="puffo"/>` can be used.

**Case sensitivity** The XML format is case-sensitive, meaning that `<guitar>` is different from `<Guitar>` or `<GUITAR>` (one of the most common syntax errors is to mistype either the opening or the closing tag). For every open tag there must be a corresponding closing tag (except for abbreviated tags, which are already closed entities) and it is not possible to interlace tags.

**Example 2.2.2** *Some non valid XML*

```
<planes>
<cars>
</planes
</cars>

<time>
 <hour>2</hour>
 <min>56</min>
</Time>
```

**Element parenthood** As said before, elements in a document form a hierachical tree structure that introduces the concept of **child**, **parent** and **sibling** node or element. From now the term node is used as being equivalent to element. In Example 2.2.1, `singers` was the **parent** of both `singer` elements, which in turn were the **children** of `singers`. Each `singer` element is the **sibling** of other `singer` elements. Additionnally, `name`, `age`, `singer` and `sex` nodes are all **descendants** of the `singers` node. Understanding the tree nature of XML is an essential step in managing data in XML format, as the parenthood between elements is important to navigate inside the tree to retrieve related information.

**Root element** Each document contains at most an element which has no parent, and that contains all other content: this element is called the root node or root element of the XML document. A root node cannot have any sibling.

**Attributes** Elements not only can contain content, but can also have associated attributes. An attribute consist in a **name** and a string **value**, and is declared inside the starting tag (or inside the complete tag if the abbreviated syntax is used). Example 2.2.3 shows some kind of attributes. Note that the value of an attribute must be enclosed in quotation marks and there cannot be two attributes with the same name within the same element.

**Example 2.2.3** *Elements with attributes*

```
<product id="1209.932">Naturla water</product>

<song style="pop" year="2004">Don't tell me</song>
```

Attributes can be used to add additional information to an element, where there is no need to add additional child elements for this. In the previous example, the element **product** has an attribute **id** that can be useful to distinguish it from other product elements, but it should be clear that it is up to the user to assure that id attributes have different values for different products, as XML does not enforce that.

**Namespaces** The XML syntax is not very strong and there is no way of distiguing elements with the same name but that carry different information. This problem arises frequently when some nodes are to be referenced by mean of querying languages, such as XPath, or when information must be extrapolated from the document without any precise reference to its structure. Example 2.2.4 provides and example XML document that better explains the point.

**Example 2.2.4** *Nodes with same name can be difficult to reference*

```
<factory>
 <name>ChristmasDream Inc.</name>
 <products>
  <product>
   <name>Teddy Bear</name>
  </product>
 </products>
</factory>
```

The example shows that the tag **name** is used in two different locations, each one with different meaning: in the first case the tag refers to the name of the factory, in the second case it refers to the name of the product. What lacks in the XML definition given up there is a way to differentiate between elements with the same name being used in different contexts: the solution is to use namespaces. Namespaces make possible to define that a given tag name has a certain meaning within a certain context and a different meaning in another context. Namespaces are declared in a similar way as attributes, by inserting some sort of declaration inside the opening tag of an element: before the equal sign goes the label **,** a colon and (optionally) a namespace prefix; the declared namespace value follows the equal sign and is called URI (Uniform Resource Locator) and, as for the value of attributes, it must be always enclosed in quotation marks. The prefix can also be omitted, so that namespaces can be declared in two forms: with a specific prefix (named namespace) or without (default namespace). To declare a named namespace named **mynamespace** we should add the following code to the element from which we want the declaration to be valid:

```
<something xmlns:mynamespace="http://www.myenterprise.com/2005/idea#ns">
```

```
...
</something>
```

And to declare that an element should be interpreted as inside this namespace, we have to add the namespace name this way:

```
<mynamespace:myelement id="1"/>
```

Note that the namespace `mynamespace` should have been declared before its usage, so in this example, the element `myelement` must be a descendant of node `something` (where the namespace `mynamespace` has been previously declared).

It is also possible to declare a namespace without a prefix, and this is done by omitting the colon and the prefix name after the `xmlns` label, like:

```
<books xmlns="http://www.syscall.org/#def2005"/>
```

By not giving a prefix to the namespace what happens is that the namespace is implicitly applied to the element and all contents inside it, where by giving it, namespaces can be used only by nodes that specify it explicitly. Namespaces are important in XML document browsing because they provide a way to extract information that belong only to certain context.

In the given example, the URI associated to namespaces always look like web addresses: there is no specific rule, meaning that the URI can be any valid XML string, but it is a common use to refer at least to the website of "creators" of the namespace.

**Strings** Character data or attributes values, as well as element names or attribute keys must be valid Unicode (UTF-8) strings. Accented letters, language dependend characters and control or reserved symbols (such as the "greater than" and the "less than" simbols used to indicate tags) must be properly replaced with special codes (please refer to a more complete XML reference for a list of illegal characters and their equivalent code, such as [16]).

### 2.2.2 XML Syntax definition

Despite the fact that the XML format gives the user a great degree of freedom of expression, it is often required that documents respect a certain syntax in order to be correctly interpreted by the application that's reading them. Having a defined syntax for XML documents is also essential if those are to be shared between people, and for shared data in general. For these reasons complementary languages have been designed to express syntax rules that apply to documents: applications or users can hereby validate XML against a syntax definition, or use it to generate documents valid for a particular application or XML language extension.

It is important to distinguish between valid XML documents and those that are validated against a particular syntax: in the first case, a document must only follow the rules and restrictions imposed

by the XML standard as explained in the previous section in order to be valid; in the latter case, the syntax is required by a particular application in order to be able to correctly parse the document. A document that is validated against an XML syntax definition is also a valid XML document, but not vice-versa.

### XML Document Type Definition (DTD)

An XML Document Type Definition is used to define which elements can be used inside an XML document and how they can be structured. A DTD can be declared inside the XML document itself (commonly referred as inline declaration, as shown in Example 2.2.5) or can be referenced as an external resource (Example 2.2.6).

**Example 2.2.5** *Inline DTD declaration*

```
<?xml version="1.0"?>
<!DOCTYPE mydocument [
  <!ELEMENT content (title,body)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
<mydocument>
 <content>
  <title>Reminder</title>
  <body>Foire Fribourg</body>
 </content>
</mydocument>
```

**Example 2.2.6** *External DTD declaration*

```
<?xml version="1.0"?>
<!DOCTYPE mydocument SYSTEM "mydtd.dtd">
<mydocument>
 <content>
  <title>Reminder</title>
  <body>Foire Fribourg</body>
 </content>
</mydocument>
```

*File* ***mydtd.dtd*** *would then contain:*

```
<?xml version="1.0"?>
<!DOCTYPE mydocument [
  <!ELEMENT content (title,body)>
  <!ELEMENT title (#PCDATA)>
  <!ELEMENT body (#PCDATA)>
]>
```

The document type is to be declared within a `!DOCTYPE` element: this special element does not interfere with the syntax of the XML document, and is silently ignored by applications not supporting document validation.

A document type declaration, either inline or external, must contain the name of the root element it refers to, in this case mydocument: this means that syntax is defined only for this kind of element.

External declarations contain an identifier that is used to determine the nature of the DTD: common XML formats that are described by public and univocal identifiers (typically whose published and/or recognized by the W3C) require the `PUBLIC` keyword, wheras private or local type declarations must use the `SYSTEM` keyword.

The actual syntax declaration is to be expressed by mean of declarations that must be placed in between the square brackets whithin the `DOCTYPE` element.

**Element declaration** An element is declared by mean of and `!ELEMENT` block: elements can be declared to allow them contain text, other elements, or to be empty. The syntax is:

```
<!ELEMENT name (content)>
```

The first parameter is the valid local name of the element. Valid element content is listed inside parentheses as second parameter, and can be:

- `EMPTY` : used to declare empty elements
- `#CDATA` : the element contains character data that is not to be parsed.
- `#PCDATA` : character data that is going to be parsed (if actual data contains XML elements, these need to be correctly declared in the DTD)
- `ANY` : declares an element with any content
- a list of children element names, separated by commas

When declaring children elements it is possible to put some occurrence constraints by adding a special sign after the child element name (if no sign is present the child element can only occur one time):

- `+` : declares that the child element must occur one or more times

- `*` : the child element can occur zero or more times

- `?` : the child element is optional, and can be omitted or occur one time

**Example 2.2.7** *Element declaration*

```
<!ELEMENT robot (head,arms+,legs+,torso)>
<!ELEMENT head (#CDATA)>
<!ELEMENT arms (EMPTY)>
<!ELEMENT legs (EMPTY)>
<!ELEMENT torso (ANY)>
```

In Example 2.2.7, the robot element must contain the `head`, `arms`, `legs` and `torso` children; `arms` and `legs` elements must occur one or more times, wheras `head` and `torso` elements exactly one time.

**Attribute declaration** Attributes of an element can be declared after the element's declaration itself, by mean of the `!ATTLIST` block.

```
<!ATTLIST element_name attribute_name type default_value>
```

The `element_name` is the name of the element the attribute applies to. The type of the attribute can have the following values:

- `CDATA` : for character data

- `(evalue|evalue|..)` : for an enumerated value

- `ID` : the value is an unique id

- `IDREF` : if the value is the id of another element

- `IDREFS` : if the value is a list of other identifiers

- `NMTOKEN` : for a valid XML name

- `NMTOKENS` : for a list of valid XML names

- `ENTITY` : if the value is an entity

- `ENTITIES` : if the value is a list of entities

- `NOTATION` : if the value is a name of a notation

- `xml` : if the value is predefined

The `default_value` can have the following values:

- `value` : the attribute value is assumed to be `value` if no declaration is found in the XML document
- `#REQUIRED` : the attribute must be explicitly declared
- `#IMPLIED` : the attribute is optional
- `#FIXED value` : the attribute value is fixed

**Example 2.2.8** *Attribute declaration*

```
<!ELEMENT hair (EMPTY)>
<!ATTLIST hair color CDATA #REQUIRED>
<!ATTLIST hair bald (true|false) "false">
```

**Entity declaration** Entities can be viewed as variables used to define shortcuts to common text. Entities can reference themselves, and can be declared externally or internally by mean of the `!ENTITY` block.

External declarations require the URI of the document declaring them internally:

```
<!ENTITY entity_name SYSTEM "uri">
```

The entity name can be any valid XML label; an entity can then be used inside the XML document by prepending the `&` character to the entity name. An example of external entity declaration is shown in Example 2.2.9, where the author entity can be subsequently referenced by mean of the `&author;` label.

**Example 2.2.9** *External entity declaration*

```
<!ENTITY author SYSTEM "http://www.syscall.org/entity.dtd">
```

In contrast, internal declarations set the entity value in place:

```
<!ENTITY entity_name "entity_value">
```

An example of internal entity declaration is listed in Example 2.2.10; references to internal entities are done in the same way as for external ones.

**Example 2.2.10** *Internal entity declaration*

```
<!ENTITY author "Elton John">
```

The Document Type Definition is included in the XML 1.0 standard, but it is often seen as limited because it does not support some newer XML features, such as namespaces. For this reason other syntax description languages have emerged, for example XML Schema.

#### XML Schema

XML Schema is an XML syntax definition language elaborated by the W3C and published in 2001. This language is much more powerful and expressive than DTD for describing syntaxes, by providing fine control over the format and data types of element and attribute values. The XML Schema standard from W3C [38] includes simple and complex data types, type derivation and inheritance, better element occurrence constraint and namespace-aware element and attribute declarations.

Because this language is not as simple as DTD a formal description is omitted here, because an at least satisfactory explaination would require a chapter on its own; by the way more information can be found in [16] and some tutorials are provided at [44].

## 2.3 Security and authentication

Data protection has an important role in long term storage, much more important than fast access, fast data throughput or concurrent access by multiple clients. As the data is in XML format, it was obvious to look for an XML-compatible security method, meaning that it is important that secured data is still valid XML, in order not to invalidate the assumptions made for this project. Fortunately there exist a language extension called XML-Encryption designed to produce encrypted but valid XML from other data. Another way to protect data from unwanted modification is to authenticate it using a digital signature based on public key cryptography, but, as we will see later in this report, it was not implemented as part of the server as it can be easily added as a client feature.

In this section a technical overview of this format and its capabilities is offered.

### 2.3.1 Cryptography

The term “encryption” means a way of protecting some data from unwanted access. Data encryption is mostly used when data must be transmitted over an insecure channel or stored on an insecure data storage media. The term “insecure” means that there exist the possibility that a third person listens on the communication channel or steal the storage media and get access to sensible data.

The etymological meaning of the word "cryptography" is due to Greek words "kryptós", which means "hidden", and "gráphein", "to write". Cryptography is the study of various means of converting information from its normal, plain and comprehensible form in to an (human and machine) incomprehensible format, in order to make it unreadable to persons without some secret knowledge.

Cryptography has been used since many centuries, mostly by armies and governments, to store sensible data and ensure secrecy of important communications. Clearly there are big differences in methods used in the past and what is required for today's security: in ancient times cryptography was related to linguistics studies, because information was essentially textual and because of the fact that encryption and decryption would have been done by humans (which limited the use of too complex algorithms).

In recent times, the development in computing power has permitted the exploitation of better encryption methods that are strongly tied with mathematics, especially discrete mathematics, including number theory, information theory, computational complexity, statistic and combinatorics. Along with better and more secure encryption algorithms, cryptography has seen a development not only in the governmental or army field, but also in civilian usage, for example digital signatures or digital payment systems. A help to this wide usage has come from the fact that its use has been greatly simplified, in a manner that now cryptographic technology is transparently built into much of today's computing and telecommunications infrastructure, without users being aware of it. The counterpart to cryptography is called cryptanalysis, and its a science that studies ways to circumvent cryptographical protections.

### Glossary of terms

The original information which is to be protected by cryptography (or encrypted) is called **plain text**. The plain text gets encrypted or enciphered by mean on an **encryption algorithm** or cipher to produce an unreadable version called **cipher text** or cryptogram. Most encryption algorithms require a key parameter or variable that controls the way the ciphered text gets produced. An encryption algorithm can be viewed as a two parameters function, taking an argument for the plain text and one for the encryption key.

The process of recovering the plain text from the ciphered one is called **decryption** or deciphering. Also for this reverse process a decryption key (maybe different from the encryption one) and a decryption algorithm is needed.

Protocols specify the details of how ciphers (and other cryptographic primitives) are to be used to achieve specific tasks. A suite of protocols, ciphers, key management, user-prescribed actions implemented together as a system constitute a **cryptosystem**.

### Goals of cryptography

Cryptography has essentially five goals to achieve (not necessarily at the same time):

(i) **Confidentiality**: the cryptosystem must be able to give access to data only to an authorized recipient. This means that there must not be any possibility to access (decipher) ciphered data without the secret knowledge. Additionally the system must not give any significant information about plain contents to unauthorized recipients.

(ii) **Integrity**: who receives the data should be able to check if contents have been modified during transmission.

(iii) **Authentication**: the recipient should be able to identify the sender of the message univocally, so that it is possible to check if data comes from a wrong source.

(iv) **Non-repudiation**: the sender should not be able to deny having sent a message.

(v) **Antireplay**: data should not be allowed to be sent multiple times to the recipient without the sender knowing.

Although cryptography is able to provide every of the above goals, there are situations where only some of them are desirable. Consider, for example, when a message needs not to be ciphered, but the content itself is required to come from a particular sender; in this case, only authentication and integrity would be useful.

### Ciphers

Unfortunately there is generally no "one fits all" algorithm, but there exist two categories of algorithms trying to achieve some of the goals listed in the previous sub-section: **public key** cryptography and **symmetric key** cryptography. The difference between these two methods is simple (at least from the user point of view): symmetric key algorithms use the same key for both encryption and decryption as the public key algorithms (also called asymmetric) use different keys to cipher and decipher data. Algorithms in both categories can then be grouped into stream ciphers and block ciphers. Stream cipher have no size limit to the data they can deal with, wheras block cipher can only work on pieces of data (blocks) of a certain length at time, meaning that encrypting more data requires that it is divided in many blocks first. The block ciphers DES, IDEA and AES, and the stream cipher RC4, are among the most well-known symmetric key ciphers.

Public key and simmetric key cryptography have both advantages and drawbacks. Usually symmetric ciphers are faster than asymmetric ones, but also have the problem that both sender and recipiend have to share the same key. As this key must also be transmitted from the sender to the

recipient someway it must be exchanged in a secure way. Public key encryption has the advantage of having different keys for encryption and decryption, meaning that the public key (as the name says) can be freely distributed over an insecure channel.

Public key algorithms are designed to make use of usually hard mathematical problems, such as factorization of big numbers. For efficiency reasons, hybrid encryption systems are used in practice; a key is exchanged using a public-key cipher, and the rest of the communication is encrypted using a symmetric-key algorithm (which is typically much faster).

#### Cryptographic hash and digital signatures

Criptography is not only concerned with protecting data from unwanted access but also to authenticate it to prevent illegal modification. Digital signatures are a way to authenticate data by the use of public key algorithms and hash functions. A cryptographic hash function is a one-way function (easy to compute, difficult to invert) that produces new data (an hash) that identifies univocally the data from which it was generated. Well known hash functions are MD5 and SHA1. The generated hash is then ciphered using a public key algorithm to produce a digital signature, that can be verified by recalculating the hash on the received data and comparing it with the one in the signature.

#### Public key criptography

As stated before, symmetric key algorithms use the same key for both encryption and decryption, and public key algorithms make use of a public key for encryption and a different, private, key for decryption. The public key can be derived from the private key, but it must be difficult (ideally impossible) to derive the private one from the public one. This means that the sender can send the public key to someone else and be sure that he would be the only person that can decrypt and read encrypted messages because he owns the related private key.

### 2.3.2 XML Encryption

XML Encryption [19] specifies a syntax to encrypt XML content, while maintaining compatibility with the XML syntax. XML encryption is a specification developed by World Wide Web Consortium (W3C) in 2002 that contains the steps to encrypt data, to decrypt data, the XML syntax to represent encrypted data, the information used to decrypt the data, and a list of encryption algorithms such as triple DES, AES, and RSA. This means that, instead of using common text or binary encryption formats, encrypted information is still XML, which also means that it can be created, managed and processed by traditional XML tools. XML Encryption can be used to encrypt almost arbitrary data, but if it is used to encrypt XML, it is limited to encrypting an entire element or the entire content of an element. For example, it is not possible to encrypt a single attribute value; thus the minimal granulatity offered by XML Encryption is single element encryption.

### Enveloping and Detached encryption

Encrypted XML data is usually defined inside an `EncryptedData` element, which contains, beside the cipher data itself, also information on how it was produced (algorithm used, key,...).

The XML Encryption allows two types of encryption: either **enveloping** or **detached**. The first format is used if the encrypted data is contained in the `EncryptedData` structure, the latter if the encrypted block is only referenced and stored outside the `EncryptedData` element. Note that this framework only support creation and processing of enveloping encryption. Figure 2.1 shows a schematical view of enveloping encryption, wheras Figure 2.2 shows detached encryption.

### XML Encryption syntax

Before analyzing in detail how XML Encryption works, lets specify its complete XML syntax by using the XML example shown in note 2.3.1.

**Note 2.3.1** *XML Encryption syntax example*

```
<EncryptedData Id? Type?>
 <EncryptionMethod/>?
 <ds:KeyInfo>
  <EncryptedKey/>?
  <AgreementMethod/>?
  <ds:KeyName/>?
  <ds:RetrievalMethod/>?
  <ds:*/>?
 </ds:KeyInfo>?
 <CipherData>
  <CipherValue/>?
  <CipherReference URI?/>?
 </CipherData>
 <EncryptionProperties/>?
</EncryptedData>
```

It is worth to note that the XML Encryption syntax does not have a version number field; instead the namespace definition is used for this purpose.

### The EncryptedType

`EncryptedType` is an abstract type from which both `EncryptedData` and `EncryptedKey` elements are derived. It essentially describes some information about the method and key used to perform

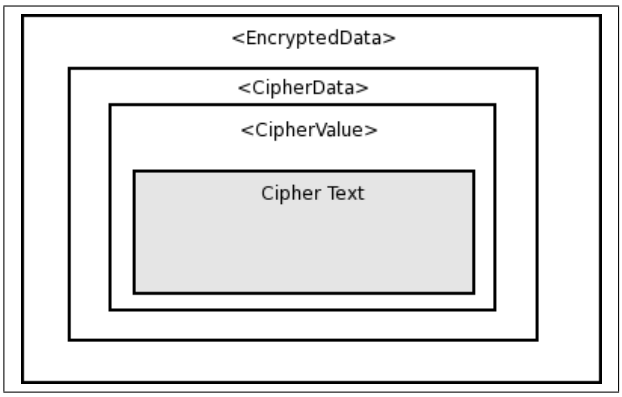


Figure 2.1: Enveloping encryption

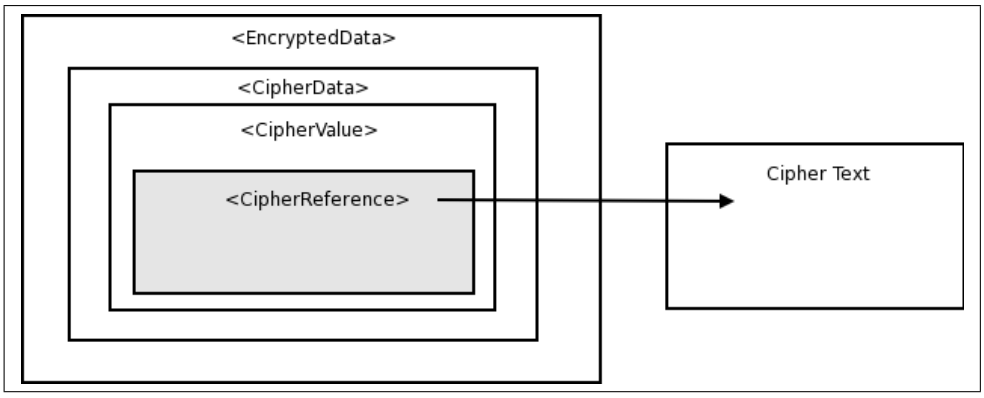


Figure 2.2: Detached encrytion

encryption of the respective content (the `CipherData` for the first, the key for the latter). The schema relative to this node is shown in Note 2.3.2.

**Note 2.3.2** *EncryptedType Schema*

```
<complexType name="EncryptedType" abstract="true">
  <sequence>
    <element name="EncryptionMethod"
             type="xenc:EncryptionMethodType"
             minOccurs="0"/>
    <element ref="ds:KeyInfo" minOccurs="0"/>
    <element ref="xenc:CipherData"/>
    <element ref="xenc:EncryptionProperties"
             minOccurs="0"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
  <attribute name="Type" type="anyURI" use="optional"/>
  <attribute name="MimeType" type='string' use="optional"/>
  <attribute name="Encoding" type="string" use="optional"/>
</complexType>
```

The "abstract" schema attribute means that it is not possible to use an `EncryptedType` element directly, but only by mean of one of its "implementation" as `EncryptedData` or `EncrypteKey` elements. There are many attributes that are marked as optional.

The `EncryptionMethod` element contains informations about the encryption method used to encrypt data, but can be omitted if the data itself is not encrypted (identity is a valid, although meaningless, encryption method) or if the encryption method can be determined by the application context.

The optional `ds:Keyinfo` element contains the original definition of this element, which carries information about the key that was used to encrypt the data.

The `CipherData` element, which cannot be omitted, contains or points to the actual cipher text (a pointer must be described by mean of a `CipherReference` element).

The `EncryptionProperties` child can contain additional information concerning the encryption, such as the time it was performed.

The `EncryptedType` element can also declare some optional attributes:

- `Id` , an identifier used to reference the node within the document
- `Type` , which provides information about the plain text and/or its processing before encryption (for example compression)

- `MimeType` , which sets the MIME type of the encrypted data (for example **image/png**)

- `Encoding` , which determines the encoding of the encrypted data (for example `Base64`)

As these attributes are optional, no check is done; in most cases the application is able to deduce some of them from the `Type` attribute.

### EncryptionMethod element

The `EncryptionMethod` is an optional element that can descend from elements "implementing" the `EncryptedType` syntax, that describes the encryption algorithm applied to the cipher data. As this element is optional, if missing, the application must be able to determine the encryption algorithm from the context. The schema for this element is shown in Note 2.3.3.

**Note 2.3.3** *EncryptionMethod Schema*

```
<element name="EncryptionMethod"
         type="xenc:EncryptionMethodType"/>
<complexType name="EncryptionMethodType" mixed="true">
  <sequence>
     <element name="KeySize" minOccurs="0"
              type="xenc:KeySizeType"/>
     <element name="OAEPparams" minOccurs="0"
              type="base64Binary"/>
     <any namespace="##other" minOccurs="0"
                              maxOccurs="unbounded"/>
     <!-- (0,unbounded) elements from
          (1,1) external namespace -->
  </sequence>
  <attribute name="Algorithm"
             type="anyURI" use="required"/>
</complexType>
```

Children allowed for an `EncryptionMethod` element depend on the algorithm chosen (specified by the `Algorithm` attribute). Example algorithm types are element encryption, declared by setting the `type` attribute as **http://www.w3.org/2001/04/xmlenc#Element** (supported by this framework for transparent decryption) and content encryption, with the type value set to **http://www.w3.org/2001/04/xmlenc#Content** .

### CipherData Element

The `CipherData` element is a mandatory child of the `EncryptedData` and `EncryptedKey` elements. Essentially it is "where" ciphered data is contained or referenced; the encrypted data is encoded as a Base64 character sequence. The schema for `CipherData` is listed in Note 2.3.4.

**Note 2.3.4** *CipherData Schema*

```
<element name="CipherData"
         type="xenc:CipherDataType"/>
<complexType name="CipherDataType">
  <sequence>
    <choice>
      <element name="CipherValue"
               type="ds:CryptoBynary"/>
      <element ref="xenc:CipherReference"/>
    </choice>
  </sequence>
</complexType>
```

As mentioned before, encrypted data can be either enveloped or detached; in the first case it is a `CipherValue` element to be used, in the latter a `CipherReference` one. The `CipherReference` element must then point to the cipher text, and the syntax used to retrieve the external node by mean of an URI and its pre-processing using a `Transform` is found in Note 2.3.5.

**Note 2.3.5** *CipherReference Schema*

```
<element name="CipherReference"
         type="xenc:CipherReferenceType"/>
<complexType name="CipherReferenceType">
  <choice>
    <element name="Transforms" minOccurs="0"/>
  </choice>
  <attribute name="URI" type="anyURI" use="required"/>
</complexType>
<complexType name="TransformsType">
  <sequence>
    <element ref="ds:Transform"
             maxOccurs="unbounded"/>
  </sequence>
</complexType>
```

The `URI` attribute is mandatory, and must point to the XML data to be processed; actual cipher text is generated by applying the given `Tranform` (if specified).

### EncryptionProperties element

The `EncryptionProperties` element can be used to add additional information concerning the `EncryptedData` or `EncryptedKey` elements. For example, it could be useful to insert a timestamp to be able to determine when encryption took place.

**Note 2.3.6** *EncryptionProperties schema*

```
<element name="EncryptionProperties"
         type="xenc:EncryptionPropertiesType"/>
<complexType name="EncryptionPropertiesType">
  <sequence>
    <element ref="xenc:EncryptionProperty"
             maxOccurs="unbounded"/>
  </sequence>
  <attribute name="Id" type="ID" use="optional"/>
</complexType>

<element name="EncryptionProperties"
         type="xenc:EncryptionPropertyType"/>
<complexType name="EncryptionPropertyType" mixed="true">
  <choice maxOccurs="unbounded">
    <any namespace="##other" processContents="lax"/>
  </choice>
  <attribute name="Target" type="anyURI" use="optional"/>
  <attribute name="Id" type="ID" use="optional"/>
  <anyAttribute namespace="http://www.w3.org/XML/1998/namespace"/>
</complexType>
```

### EncryptedData element

The first type of node derived from the abstract type `EncryptedType`, is the `EncryptedData` element. As an element gets encrypted, it is replaced with an `EncryptedData` element, and the cipher text is contained in its `CipherData` child. Note 2.3.7 shows its schema.

**Note 2.3.7** *EncryptedData schema*

```
<element name="EncryptedData"
         type="xenc:EncryptedDataType"/>
<complexType name="EncryptedDataType">
  <complexContent>
    <extension base="xenc:EncryptedType"/>
  </complexContent>
</complexType>
```

### EncryptedKey element

The `EncryptedKey` element can be used to embed encryption keys which have been also encrypted. Encrypted data and keys can be either included in this element (inside a `CipherData` child) or simply referenced (as it was also possible with the `EncryptedData` element). This enveloped key structure means that it is possible to carry cipher data along with the necessary key within the same XML construct.

**Note 2.3.8** *EncryptedKey schema*

```
<element name="EncryptedKey"
         type="xenc:EncryptedKeyType"/>
<complexType name="EncryptedKeyType">
  <complexContent>
    <extension base="xenc:EncryptedType">
      <sequence>
        <element ref="xenc:ReferenceList"
                 minOccurs="0"/>
        <element name="CarriedKeyName" type="string"
                 minOccurs="0"/>
      </sequence>
      <attribute name="Recipient" type="string" use="optional"/>
    </extension>
  </complextContent>
</complexType>
```

The `ReferenceList` element can be used to point to data and keys encrypted with the key contained into a `CipherData` child. The optional `CarriedKeyName` element is used to associate an human-readable name to the key, and it shouldn't necessarely be unique within the document. Clearly this element type inherits other attributes and properties from the `EncryptedType` abstract type, such as a `Type` attribute used to specify the type of the encryption key.

### ReferenceList element

The `ReferenceList` element carries references to `EncryptedData` elements encrypted using the key defined in the `EncryptedKey` child. These references are stored inside `DataReference` elements, and these can point to multiple `EncryptedData` elements encrypted with the same key. As for `CipherData`, it is possible to associate a `Transform` element that specifies a transformation to be performed on the target URI to retrieve the actual data. Note 2.3.9 shows the schema associated with the `ReferenceList` element.

**Note 2.3.9** *ReferenceList schema*

```
<element name="ReferenceList">
  <complexType>
    <sequence>
      <element name="DataReference"
               type="xenc:ReferenceType"
               minOccurs="0" maxOccurs="unbounded"/>
      <element name="KeyReference"
               type="xenc:ReferenceType"
               minOccurs="0" maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>

<complexType name="ReferenceType">
  <sequence>
    <any namespace="##other" minOccurs="0"
                             maxOccurs="unbounded"/>
  </sequence>
  <attribute name="URI" type="anyURI" use="optional"/>
</complexType>
```

### An example

To better understand how an `EncryptedData` element looks like, Example 2.3.1 shows a fragment of generated code, referencing a key named **mykey**.

**Example 2.3.1** *An example of generated EncryptedData element*

```
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#"
               xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
               Type="http://www.w3.org/2001/04/xmlenc#Element">
  <EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#aes128-cbc"/>
  <ds:KeyInfo>
    <ds:KeyName>mykey</ds:KeyName>
  </ds:KeyInfo>
  <CipherData>
    <CipherValue>
    rmpcNKyxdWGObsNauugZT2P8w22Wuqoq69
GYqKQ+kaCQaoJoAtnFN98uk4dahp8p9vy2y/s9HDv+
eEdTDS9zavLCVD2cRtDPb5YAFJ6i4I9puFEyrIhnn8F9Q
HCM74xQLeIBNEoVFuQS3Gpk6nUK8xtVTb9vD1uw3X9
HYYVSWUi8KVV5gxEcKTjlr4nReL+60bazp0UXsNJEzM4
jSE6v6sEhUxs+rB5nJV6Y     VpjJ2xuegC3iGWZTyWrhjB
14hshVtdJEJXs2X8lna/cdK7sVhqRG7w7L0QHDHFOT6/
PnNg69bZjxFhmmOoPvD2EPnzHlIZhHIaa1aVoH7H/
oJ9ojTa2qMyi+DDNBqhad8UtefdMkhdaiXXoMjXfBl7LY
GKnYqUvjTvDx8beLo8JIcgI6g2d0jJiDJ4u4hdbiGBWnRF
MBpEVWkucfQCg
    </CipherValue>
  </CipherData>
</EncryptedData>
```

The chapter dedicated to the actual framework implementation discuss how this element is constructed and how it is then parsed to retrieve plain XML data.

## 2.4 Database querying

A data storage system, as well as a database, should not only offer a way to store information, but also a practical way to retrieve and modify it. The term "practical way" means that retrieval and update of data should be as simple and precise as possible. Traditional databases (MySQL, PostgreSQL,...) [2, 18] offer querying languages such as SQL, which is both powerful and easy to use, but that are bound to the relational model on which data is organized: the object oriented nature of XML requires other approaches to querying languages. In this section three querying languages are examined: XPath, XUpdate and XQuery.

### 2.4.1 XPath

XPath is the acronym of XML Path Language and it is a W3C Recommendation [11] used to identify particular nodes or a set of nodes inside of an XML document. This language is also used by other languages, such as XSLT, XSL Schema, XPointer and has been implemented in many other situations where a way to retrieve elements or fragments of code from an XML document was necessary.

As described before, an XML document is organized in a **tree structure**, where data is contained in objects called **nodes**; what an XPath query does, is retrieving nodes or sets of nodes out of this tree [35].

It is important to note that the XPath syntax is not derived in any way from XML, and in fact its queryies (mostly in the abbreviated form) look like paths inside a filesystem or web URLs.

#### Structure of an XML document

In one of the previous sections the basics of the XML syntax have been explained: an XML document is based on the concept of node, so the XPath language is also based on this concept to perform search inside the XML tree. Before explaining how to perform searches, lets have a look on the different types of nodes and their type. There are also additional XML objects that are also considered as nodes within the XPath specification.

#### Node types

Most ofnodes inside an XML document (or a document fragment) are from either one of the following three node types:

**Root Node** This node is the root of the document, and contains all other nodes. For each document there is exactly one root node, also there is always only one path inside the XML tree from the root to any other node.

**Example 2.4.1** `<rootnode> ... </rootnode>`

**Element Node** Element nodes represent, as the name says, XML elements. Elements can also have children elements or some text content that is described by other node types.

**Example 2.4.2** `<anelement> ... </anelement>`

**Attribute Node** This kind of node represents an attribute associated with an element.

**Example 2.4.3** `<element attribute="Value" ...>`

Beside these types of nodes, to which everyone with at least a minimal knowledge of XML (from the dedicated section above) is probably familiar, there are some other nodes, which are less frequent inside an XML document:

**Namespace Node** They represent namespace definitions (either prefixed or not).

**Example 2.4.4** `<stewie xmlns:devil="http://www.stewie.com" ...>`

**Processing Instruction Node** These nodes describe some data processing; although the format is similar to an XML document type declaration, mostly because of quite the same start and end tags, the document type declaration is not a processing instruction node, in fact it is not considered at all, because of being part of the XML syntax for storing and transmitting and recognizing XML data.

**Example 2.4.5** `<? ... ?>`

**Comment node** These node are used to insert user targeted comments inside an XML document. A comment does not interfere with the XML syntax, and can be used regardless of a DTD or Schema, without invalidating the document.

**Example 2.4.6** `<!-- Here's a comment -->`

**Text node** They represent the text (that is, non-XML content) inside an element node.

**Example 2.4.7** `<element> Some text </element>`

There are some constructs that are not listed, but that belongs to an XML document: CDATA sections, entity references, and document type declarations. The XPath language cannot simply deal with these entities and ignores them, therefore the root node of the XPath data model is different from the root element in the Document Object Model of the same XML document.

Now that all available node types that can be found inside an XML document are clear to the user, it is time to explain the XPath syntax used to retrieve a particular node or a set of nodes.

### Location Path

The Location Path is the essential part of the XPath syntax. By mean of a set of rules it is possible to construct complex expressions to locate every portion of an XML document, and extract valuable information from it.

These expressions are evaluated by the XPath engine against the document, and the returned value of this evaluation can be of one of the following types:

**Booleans** Boolean values are tipically returned by comparation operators, and are binary data whose value can be either true or false. There exist no Boolean literals in XPath; to get boolean values it is necessary to use the **true()** and **false()** functions as substitutes.

**Numbers** All numbers in XPath are 8-byte, IEEE 754 floating point doubles: for comparison their data type similar to the double numerical type found, for example, in the Java programming language. An XPath number can represent positive or negative values, with a range from 5e-324 (minimum positive value) to 1.7976931348623157e+308 (maximum positive value). Additionally also special values defined by the specification are included, such as **Inf** (Positive Infinity), **NInf** (Negative Infinity) and **NaN** (Not a Number).

**Node-Sets** Represents a set of nodes that can be of one of the previous listed types. A node-set can contain zero, one or more nodes.

**Strings** Strings are sequences of Unicode characters, enclosed in single or double quotation marks. The quotes are not themselves part of the string, and a string cannot contain the same type of quotes that delimits it.

Now, lets have look how to retrieve data, that is how to construct an XPath expression that can be evaluated by an XPath engine in order to return specific information from an XML document. First, it must be said that there are two ways to write valid XPath expression, an **abbreviated** form and a **long** one.

The abbreviated form looks very similar to Unix paths, and is constructed with slashes, dots, etc. whereas the long form is constructed of tree parts: the axis, one or more node tests and an optional part called predicate.

Location paths can be either **absolute** or **relative**: absolute location path starts with a slash ( `/` ) and a relative location path does not. In both cases the location path consists of one or more location steps, each separated by a slash. The difference between these two is that the first is valid from every part of the XML document we are at (because it always make references starting from the root node) whereas a relative location path is only valid from a particular position inside the tree. Each location step is evaluated relative to a particolar node in the document called context node, which is the result of the evaluation of the preceding location step.

**Example 2.4.8** *Absolute and relative location paths*

*Absolute:*

```
/path/to/my/node
```

*Relative:*

```
to/my/node
```

*Supposing that we are at node* `path`*, the two location paths point to the same node "*`node`*".*

The concept of relative location paths shows that XPath also defines a concept of context in which the system is currently at. This context can be any kind of node, and each successive relative

XPath expression is evaluated against it. We call every successive expression an **XPath step** that is evaluated against the nodes in the current node-set, resulting from the previous step. An XPath expression is **always** evaluated against a set of nodes resulting from another expression or path.

### Long form

This section analyzes the long form, which is the most complete form, but also the least used one because in most cases it is too much textttose for practical use. By the way, this description is useful to understand how the XPath location path is used to retrieve nodes.

An XPath expression has the following syntax:

**Note 2.4.1** *XPath expression syntax*

```
Axis:NodeTest[Predicate]
```

The current location is omitted, because it is implicit.

### Axis

The first part of an XPath expression is called Axis, and it is used to define in which direction, starting from the current location (or context) the search would continue. There are many available axes, which refer to the current node or current node set.

**self** It represents the node itself

**child** All child nodes of the current node set (it is the default node, so it can be omitted).

**descendant** All nodes contained in the set of nodes (children, children of the children, etc.).

**descendant-or-self** All descendants of the set of nodes and the node itself.

**parent** All parents of the nodes in the set or the parent of the node itself.

**ancestor** All elements and the root nodes that contain the set of nodes (it can be viewed as a reversed descendant axis).

**ancestor-or-self** All elements and the root node that contain the set of nodes and the node itself.

**preceding** All nodes that precede the set of nodes, that is nodes that terminate before the set of nodes.

**preceding-sibling** All siblings that precede the current set of nodes, that is, siblings that terminate before the beginning of the set of nodes.

**following** All nodes that follows the current set of nodes, that is, nodes that begin after the current set of nodes.

**following-sibling** All following siblings of the current set, that is, siblings that begin after the current set of nodes.

**attribute** All attributes of the current set of nodes.

**namespace** Namespaces associated with the current set of nodes.

#### Note test

To filter out nodes resulting from the search starting from the current context, in the direction specified by the axis, a test parameter can be used. Available test nodes are:

**name** Each element or attribute with that name in the specified axis direction

***** Each element in the current axis

**prefix:*** Each element or attribute with the namespace 'prefix' in the axis direction

**comment()** Each comment node found

**text()** Each text node found

**node()** Each node found

**processing-instruction()** Each processing instruction found along the axis direction

**processing-instruction("target")** Each processing instruction that refers to target in the axis direction

#### Predicates

As the relevant set of nodes has been determined by mean of a selection, an axis and a node test, it is possible to use predicates to yet refine the search. The predicate (or operator) is written after the **axis::nodetest** part, inside square bracets [ ... ]. As predicates are the same for both long and short form, they will be discussed later in this section.

### Abbreviated form

The abbreviated form is the most used one because of its simplicity and clarity. By the way it is somewhat limited in respect to the long form, because there are eight axes missing: ancestor, following-sibling, preceding-sibling, following, preceding, namespace, descendant, ancestor-or-self.

### Selecting nodes

To select nodes using the abbreviated form, to which successive relative XPath expressions will refer, expressions shown in Table 2.1 can be used.

| **Expression** | **Description** |
|---|---|
| nodename | Selects all child nodes of the node |
| / | Selects from the root node |
| // | Selects all nodes nodes in the document from the current node |
| . | Selects the current node |
| .. | Selects the parent of the current node |
| @ | Selects the attributes |

Table 2.1: Abbreviated form for selecting nodes

### Operators

In XPath there exist two types of operators, arithmetic and boolean ones. Each operator has a given priority, that is used to determine the order in which operators are applied to the current set of nodes. Operators with the lower priority value are executed first, whereas operators with the same priority value are executed in no particular order (in fact they are executed in sequence, one after another).

### Arithmetic operators

Arithmetic operators can be used to perform some simple computations. The result of arithmetic operations is always a number.

| **Operator** | **Meaning** | **Priority** |
|---|---|---|
| + | Addition | 3 |
| - | Substraction | 3 |
| * | Multiplication | 2 |
| div | Division | 2 |
| mod | Remainder | 2 |

Table 2.2: Arithmetic operators

### Boolean operators

These operators are used to compare objects; the result is a boolean value either **true** or **false**.

| Operator | Meaning | Priority |
|---|---|---|
| - | Negation | 3 |
| < | Less than | 4 |
| <= | Less than or equal | 4 |
| > | Greater than | 4 |
| >= | Greater than or equal | 4 |
| = | Equal | 5 |
| != | Not equal | 5 |
| and | Boolean AND | 6 |
| or | Boolean OR | 7 |
| \| | Union | 8 |

Table 2.3: Boolean operators

### Functions

XPath offers a number of functions that can be used inside expressions. These functions act on nodes or objects, and can be grouped inside four categories:

- Functions on sets of nodes
- Boolean functions
- Functions on strings
- Numerical functions

### Functions on sets of nodes

These functions can be used when the current context (resulting from the previous location step) results in a set of nodes.

**last()** Returns the number of nodes inside the set.

**position()** Returns the number corresponding to the position of the current node inside the set.

**count(node_set)** Returns the number of nodes inside the set provided as argument

**id(objet)** Returns a node set containing the elements that match the values specified as argument to the function; arguments must be separated with a colon.

**local-name(node_set)** Without an argument it returns the name of the context node; by specifying as argument a set of nodes it will return the name of the first node of the set.

**namespace-uri(node_set)** Without an argument it returns the URI of the associated namespace of the current context; by specifying a set of nodes as argument it will return the URI associated to the first node.

**name(node_set)** Without argument it returns the qualified name of the current context, with a set of nodes as arguments it will return the qualified name of the first node.

#### Boolean functions

Boolean functions apply on objects and result in true or false values.

**boolean(object)** Converts the specified arguments into boolean values; zeroes, NaN, empty strings and empty node sets are converted into false, everything else into true.

**not(object)** Inverts its arguments (that is, inverts the boolean value returned from objects).

**true()** Always returns true.

**false()** Always returns false

**lang(String languageCode)** Returns true if the selected node is written in the same language as the argument.

#### Numerical functions

Numerical functions are used to perform arithmetic operations or to convert other objects to a numerical value.

**number(object)** Converts the object to a number; the following rules apply: a string, if not representing a number, is converted to a NaN (Not a Number), a boolean true value is converted in 1, a false one into 0.

**sum(node_test)** Converts each node into a set of nodes (each node converted to a number by mean of the same rules as for the number() function), and calculates the sum.

**floor(number)** Returns the greatest integer value lower than the argument.

**ceiling(number)** Returns the smallest integer value greater than the argument.

**round(number)** Returns the integer value closer to the argument.

#### Considerations

XPath is a very powerful language to reference nodes (and information) inside an XML document. It is also extremely easy to use, because it relates directly to the tree structure of data. Inside roXanne Framework, almost every operation is performed on nodes, so XPath becomes extremely important as a way to locate them. Currently the "stable" XPath recommendation from the W3C working group is 1.0 (the same version available in this framework); version 2.0, currently being a draft,,, will be much more powerful and able to better deal with some less used features of the XML syntax, such as namespaces.

### 2.4.2 XUpdate

XUpdate is an XPath-based XML transformation language [15], like XSLT [14]. An XUpdate document is an XML document that specifies what changes should be made to another XML document.

Actually, XUpdate is neither a W3C Recommendation nor an ISO or IETF standard; it is just a project of the XML:DB Initiative's XUpdate Working Group, and it never advanced beyond a Working Draft published in September, 2000. By the way it is extremely simple to use and got implement in various XML based products, such as this framework.

In this section a brief description of current XUpdate features is given; the complete DTD is found in Appendix A.

#### Selections

An XUpdate expression always refers to a target node. This node is selected by mean of an XPath query set by mean of the `select` attribute. If the XPath search is invalid an error is returned, if no node is found at the specified location, no error is raised.

#### Modifications

An update command is represented by an `xupdate:modifications` element in an XML document. An `xupdate:modifications` element has some mandatory attributes, such as a `version` attribute, indicating the version of XUpdate that the update requires (currently the supported XUpdate version is 1.0).

The `xupdate:modifications` element may contain different types of elements, each describing a possible modification: `xupdate:insert-before`, `xupdate:insert-after`, `xupdate:append`,

`xupdate:update`, `xupdate:remove`, `xupdate:rename`, `xupdate:variable`, `xupdate:value-of`, and `xupdate:if`.

Because of the draft status of this language, some instruction (such as conditional processing or element renaming) have not yet been defined and implemented.

### Inserting content

Insertion commands allow to insert XML data inside the document, at a specified position. It is not only possible to insert single elements, but also tree fragments (an element and some descendants), attributes and text content. Insertion commands are defined by two elements: `xupdate:insert-before` and `xupdate:insert-after`. As the name says, the first type defines an insertion **before** the selected node, wheras the second defines an insertion **after** the selected node. The selected target node is to be specified with a `select` attribute of one of these nodes, by mean of an XPath expression that must evaluate to a set of nodes.

Insertion nodes can contain additional elements that specify what should be inserted:

**xupdate:element** This element is used to add an new element. It accepts a `name` attribute to specify the name of this new element.

**Example 2.4.9** *xupdate:element command and result*

*This example:*

```
<xupdate:element name="book">
  <title>Bald in the land of big hair</title>
</xupdate:element>
```

*Would produce this:*

```
<book>
  <title>Bald in the land of big hair</title>
</book>
```

**xupdate:attribute** This command is used to create new attributes inside the target node, which is to be specified using the `xupdate:element` element.

**Example 2.4.10** *xupdate:attribute command and result*

*This example:*

```
<xupdate:element name="book">
  <xupdate:attribute name="ISBN">0060955260</xupdate:attribute>
</xupdate:element>
```

*Would produce this:*

```
<book ISBN="0060955260"/>
```

**xupdate:text** This element is used to insert text content at a specified location.

**xupdate:processing-instruction** This will create a processing instruction node. The `name` attribute is used to specify the name of the processing node. Every other attribute and the content of this node will be added to the processing instruction element.

**Example 2.4.11** *xupdate:processing-instruction command and result*
*This example:*

```
<xupdate:processing-instruction name="example-process">
  type="xlst"
</xupdate:processing-instruction>
```

*Would produce this:*

```
<?example-process type="xlst"?>
```

**xupdate:comment** With this instruction it is possible to insert a comment. The text of the comment is to be set as the content of this element.

**Example 2.4.12** *xupdate:comment command and result*
*This example:*

```
<xupdate:comment>
  This is a comment
</xupdate:comment>
```

*Would produce this:*

```
<!-- This is a comment -->
```

### Appending content

Beside inserting new content in an XML document, XUpdate also allows to append new data to existing elements by mean of an `xupdate:append` node. Usage of this instruction is similar to inserts, as the target node to which data is to be append must be specified by mean of a select attribute. Additionally, the `xupdate-append` element also accepts a `child` attribute to specify an

XPath expression that must evaluate to an integer value; this value is used to determine the position within the children nodes of the target element where new content is to be append at.

Nodes types that can be appended are the same as for the insertion operation: `xupdate:element`, `xupdate:attribute`, `xupdate:text`, `xupdate:processing-instruction`, `xupdate:comment`.

**Example 2.4.13** *Append content example*

*This example:*

```
<xupdate:append select="/addresses" child="last()">
  <xupdate:element name="address">
    <town>San Francisco</town>
  </xupdate:element>
</xupdate:append>
```

*Would produce this:*

```
<addresses>
  <address>
    <town>Los Angeles</town>
  </address>
  <address>
    <town>San Francisco</town>
  </address>
</addresses>
```

### Updating content

It is also possible to modify (update) existing content, by mean of the `xupdate:update` instruction. This element is similar to previous modifications, in that a target node or nodes must be specified using the `select` attribute.

**Example 2.4.14** *Updating content example*

*This example:*

```
<xupdate:update select="/addresses/address[2]/town">
  New York
</xupdate:update>
```

*Would produce this:*

```
<addresses>
  <address>
```

```
    <town>Los Angeles</town>
  </address>
  <address>
    <town>New York</town>
  </address>
</addresses>
```

#### Removing content

Removing content is just as easy as adding it and is performed by mean of an `xupdate:remove` instruction. This element takes only one attribute, `select`, to specify which node or nodes to remove.

### 2.4.3 XQuery

The XPath description in the previous section was very extensive, and this is motivated by the fact that, as explained, it is an essential language to manage data stored within roXanne Framework. On the other side, XUpdate provides simple, intuitive and effective methods to update existing information. In the XML world there exist also another language meant to retrieve and update data in XML documents called XQuery [4].

XQuery is a query language (that also offers some programming language features) that is designed to query collections of XML data, and can be compared to SQL. XQuery and parts of XSLT 2.0 and XPath 2.0 are being jointly developed by the XML Query working group of the W3C. By the way, the XQuery language has not yet achieved the W3C's Recommendation status, and for this reason there are not so many engine implementations available (nor it is currently supported by this framework). Beside that, we feel that it was at least important to mention it, because in the future, it will probably became the language of choice for querying XML databases. By the way as this framework only needs a simple solution to the querying issue, this language it is somewhat "too much" complicated for everyday use.

#### Features

XQuery provides a mechanism to extract and manipulate data from XML documents or any data source that can be viewed as XML, such as relational databases or office documents. XQuery uses XPath expression syntax to address specific parts of an XML document on which additional functions and operators apply. XQuery provides a mechanism to dynamically generate new content (interpreted results) using a declarative, functional, expression (programming)-based syntax.

One of the advantages of XQuery is the fact that XML is considered as a native data type, meaning that it can be included inside queryies without requiring quoted strings or object calls (in

fact, XML elements are separated from enclosed expressions using curly braces).

## 2.5 Concurrency

Database systems providing parallel access to multiple users must ensure that concurrent access do not conflict. Concurrency management that guarantees data integrity is essential, and is strongly related to the concept of transaction.

A transaction is a set of operations executed by the database system that always leaves the data in a consistent state [46]. Integrity is enforced by requiring that transaction processing features the four ACID properties: **atomicity**, **consistency**, **isolation**, and **durability** [10].

Atomicity forces a transaction to be completely executed: if execution fails at some point, all intermediary results in the database must be canceled.

Consistency means that a transaction executed on consistent data must result in consistent data: if there are no conflicts at the start of the transaction, there must be no conflicts at the end.

Isolation ensures that two or more concurrent transactions do not interfere each other. In other words, the result of parallel execution of transactions must be the same as for serial execution of them.

Durability guarantees that once a transaction has been completed it persist, and can't be undone. This property is important because it ensures consistency in case of system failure (either software or hardware).

### 2.5.1 Serialization

Concurrent transaction processing requires a synchronization mechanism that must ensure the principle of serialization to ensure that conflicts are correctly solved. The principle states that a correctly synchronized system must ensure that result from concurrent execution must be the same as for serial execution.

To determine if in a set of transactions there exist conflicts that can preclude serialization, an in-deep analysis of read and write instructions of each transactions is required.

Figure 2.3 illustrates an example of concurrent transaction execution. Each transaction reads and writes some shared variables (A, B and C) and does some computation (increment and decrement). As every modification is not immediately (atomically) written to memory, concurrency problems can arise: for example in transaction 2, between B+1 calculation and writing of the result back in B, transaction 1 reads the same variable and gets a wrong value. The result is that variable B is only decremented by 1 because its value after increment by transaction 1 is overwritten. In particular, conflics happen when a read operation is followed by a write, or if a write operation is followed by another write.

By writing a journal of read and write operations of variables used by every concurrent transaction it is possible to identify possible conflicting operations; actual conflicts can then tested by mean of a precedence graph.

Figure 2.4 depicts the journal and precedence graph for the shared variable B: first read-write and write-write situations are identified in the journal, then they are represented in the graph as arcs going from one transaction to another, depending on the execution order.

To determine if involved transactions are serializable, the following criterium applies:

> "A set of transactions is serializable iff the corresponding precedence graph does not contain any cycle."

The previous example is clearly non-serializable. Serializability of transactions must be verified for every shared data accessed: if the test fails in at least one case, the considered set of transactions is not serializable.

To allow serializability there exist pessimistic approaches, which try to avoid conflicts between transactions as soon as possible, and optimistic approaches, which solve the serialization problem by canceling conflicting transactions.

### 2.5.2 Pessimistic approaches

A way to solve conflicting situations is to introduce **exclusive locks** on shared data. As a transaction needs access to a shared object it has to acquire a lock on it; if other transactions have to access the same object they are set to wait until the lock is freed.

In order to guarantee serializability, the locking and unlocking phases must follow a locking protocol.

**Two-phase locking protocol** The two-phase locking protocol requires that locking and unlocking requests are performed in two phases:

- **Lock acquisition**: the transaction requests all the necessary locks, not necessarily at the same time. If the transaction releases a lock then it cannot obtain any new locks.
- **Lock release**: all acquired locks are progressively released. During this phase the transaction may not acquire any new lock.

It is important that once the transaction releases a lock, it enters the shrinking phase, that forbids issuing more lock requests. In general, the two-phases locking protocol guarantees seriability of concurrent transactions [46]. Unfortunately two-phase locking is not safe from deadlocks [34].

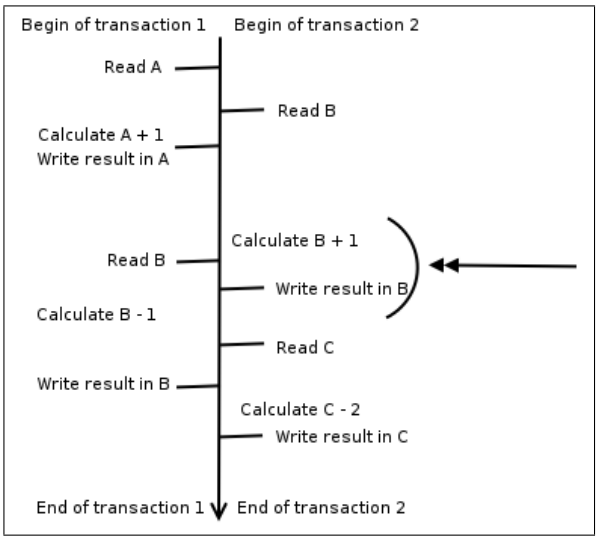


Figure 2.3: Example of concurrent transaction processing

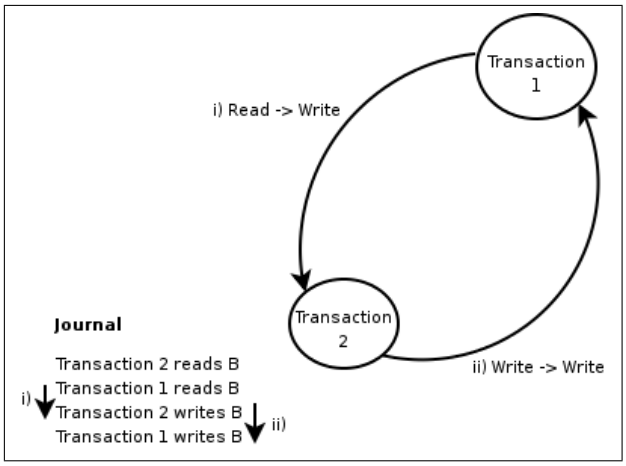


Figure 2.4: Journal and precedence graph for variable B

**Pessimistic concurrency control** When there exist no conflicts in concurrent transaction execution, two-phase lock can slow down considerably the processing. Additionally it is often not desirable to have global locks on the whole data structure, but only locks on smaller parts of the database. Locking granularity is therefore an important factor that influences processing speed. Data organized in a tree structure can also exploit hierachical dependences to optimize locking management.

Another pessimistic approach is to distinguish between read and write locks. Read locks, also called **shared locks**, allows accessing to a resource for reading only. In contrast, write locks, or **exclusive locks**, allow full read and write access to the object.

Finally also timestamps can be used to ensure serialization: every operation is marked with a timestamp, and before being applied the system verifies the chronological order of each one.

### 2.5.3 Optimistic approaches

Optimistic approches are based on the idea that conflicts between concurrent transactions are rare. Instead of employing locks each transaction is validated before changes take effect: this way transaction processing can be speed up considerably.

Transaction execution based on **optimistic synchronization** is divided in three phases: the **read** phase, the **validation** phase and the **write** phase. Modification are first executed on a private copy of the data, then in the validation phase checks are performed to verify that no conflicts exist between updates from concurrent transactions. If no conflicts exist changes are applied to the database, it they do exist the transaction is canceled.

To establish a correct transaction order, it is important that timestamps marking the start of the validation phase are used. To determine if two concurrent transactions are serializable the following method is used:

- Let A...Z be a set of transactions, sorted by the time they entered the validation phase
- Let L be the transaction to be validated.
- Let L,M,N...P be transactions concurrent to L being already validated as L was in the read phase.
- Objects read by transaction L **need to be verified**, because transactions L to P **could have modified them**.
- Let Read(L) the set of objects read by L and Write(L,M,N...P) the set of objects written by the other transactions.

With the above hypothesis, the seriability criterium says:

> "In an optimistic synchronization approach, a transaction L is serializable iff the Read(L) and Write(L,M,N...P) sets are disjoint."

### 2.5.4 Deadlocks

Concurrent transaction execution is tipically performed by mean of different threads. As mentioned before, a pessimistic approach requires the use of locks to prevent concurrent access to the database; unfortunately this introduces the risk of **deadlocks**. A deadlock condition involves threads and resources and happens when each thread is waiting for one of the resources, but all of them are already held. The problem is that all threads are waiting for each other, and are locked because none of them will release the resource it holds unless it can acquire the one it is waiting for.

Design and development of multithreaded applications involving shared resources and locks therefore requires attention to prevent these deadlock situations.

# Chapter 3

# RoXanne Framework

## 3.1 Chapter overview

This chapter describes the application that has been developed to solve the long-term data storage problem. This description of the framework is strictly from the logical point of view, as implementation details are better explained in the following chapters; nevertheless this chapter also motivates some of the design choices taken.

## 3.2 What is roXanne Framework

RoXanne is a software application written in Java meant for secure long term data storage based on the XML format. The key points of this solution are:

- It is an exploitable data storage solution, because it supports native XML storage as well XML querying and modification languages such as XPath and XUpdate.

  Being able to store data in a structured format is extremely important, because it helps preserving information organization. Most real world and computer data abstraction can be represented with the hierarchical nature of the XML language. The XML language is also suited for long term information storage because it is text-based, open, patent-free and platform independent.

  On the other side, querying languages helps retrieving and modifying information in a simple manner, which is an important feature for a data storage solution.

- It is secure, because it allows to protect sensible data by mean of encryption algorithms. Security in long term storage is essential, because there is no mean to guarantee privacy and

protection by mean of simple access control. A physical protection of data using cryptography is therefore a must. To preserve the long-term storage purpose, standardized encryption formats are used, such as XML-Encryption.

- It is designed as a modular client-server framework leaving place for many enhancements and extensions: each component can be replaced or extended.

  The two-tier architecture allows better exploitability in productive environments, where storage solutions are to be used by multiple client applications.

## 3.3 Framework structure

Roxanne Framework has been designed from the ground up as a client-server application: accessing XML data is done by using a client application that connects to the server and communicates with a defined protocol. Building a large monolithic application is probably the simplest way to test at least a prototype of a working application. In fact, avoiding concurrent access by multiple users can make the whole framework a lot easier to develop, but then instead of a large application, all the work could be reduced to just a simple dynamic library.

Separating the server part and the client one has several advantages: first it is possible to install and run the server on a much more capable machine, with big storage and memory space as well as much more processing power, so that managing information is a lot faster than it would be if everything is installed on a "standard" workstation. Another reason is security: a central server can be made secure easily than dozen of clients; also the fact that data is stored in a single location means that changes are immediately visible by all clients connected to the server, without needing to replicate and spread them to everyone.

It should be noted that this framework manages data directly in XML format, compared to common databases with non-native XML support that only provide XML by mean of output and input filters or middleware that performs conversion from the relational model to XML and vice-versa [30, 31].

The framework is composed of three components:

- **x.core** : this module defines the server application that stores and manages data, and accepts queries from client applications.

- **x.click** : this is an example client application offering access to all features of the server. This graphical user interface also allows simplified management of cryptographic keys and wizard procedures to speed up work.

- **x.mill** : a database migration library/tool that is integrated within **x.click**. With **x.mill** it is possible to import data from and to an external database in XML format.

### 3.3.1 Two-Tier Software Architecture

The framework is based on a two-tier software architecture [32], also known as client-server model, which is composed of a server layer (the provider of the service) and a client layer (the requester of the service). The architecture is divided in three components:

- Data management, data storage and retrieval
- Data processing
- User interface, user interaction management

Data management is performed by the server component (**x.core**) only, and consist in storing and retrieving XML data as well as offering methods to access information from external applications. Data processing is performed both on the server side (for example for things related to cryptography) and on the client side (because XML data is normally processed by an application on the client machine). Finally, user interaction management is completely done by the client application, which can be either the **x.click** component of the framework or a custom application making use of some kind of adapter library to issue requests to the server. Figure 3.1 illustrates an overview of the whole framework.

A two-tier architecture allows a clean and modular application design, because it is possible to separate the user interface from the data management and storage parts. Communication between the server and the client is low as most of the data processing (namely things that do not involve server's built-in security features) can be performed on the client layer. Overall data security also takes advantage of this design because cryptography is completely managed on the server.

The only major drawback is scalability: two-tier architectures do not scale beyond small environments because clients connections are directly managed by the server. By the way, goals of this project do not include high performance data serving, which would have been also difficult to consider due to the high computation requirements of cryptographic methods.

### 3.3.2 Use in a three-tier architecture

The **x.core** component can also be used in a three-tier model as a data storage component. In fact this use is perhaps the most common in productive environments, because XML data is expected to be further processed by other specialized applications. Figure 3.1 shows such implementation by mean of a database adapter which allows a middleware application retrieve and further process data before presenting results to the user. The adapter makes also possible to hide the client-server protocol details and exploit abstractions and features provided by the target language.

An advantage of such approach is that data can be really protected as soon as it gets generated by the client or middleware applications. Additionally by mean of a middleware software it would be possible to manage a cluster of servers running the **x.core** component.

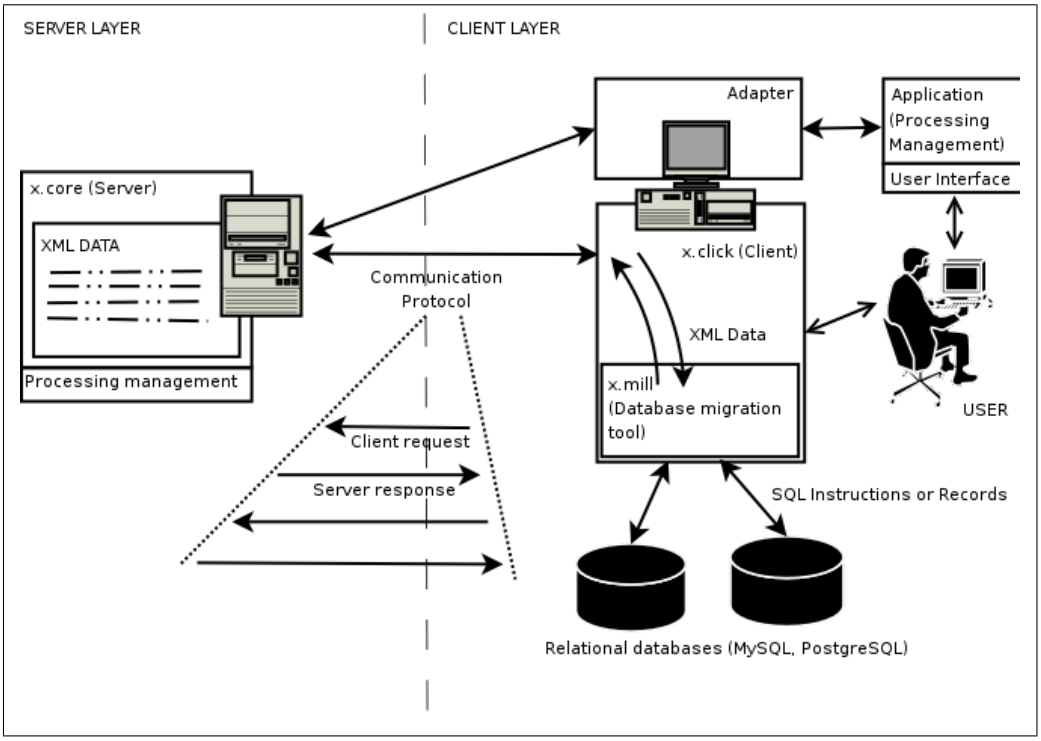


Figure 3.1: Framework overview

## 3.4 x.core component

The role of the **x.core** component is to manage the information and offer the client a way to access it; because of the security requirements of this project, there is some processing on the server to assure data protection using cryptographical methods.

Figure 3.2 shows a schematic overview of the component internal organization. The server block is responsible for accepting incoming client connections, and for parsing and executing requests.

Sessions provide some persistent user's information storage on the server: as it will be better discussed later, by mean of session objects it is possible to upload encryption and decryption keys on the server in order to simplify security management.

The data management part is responsible of storing the XML as well as offering methods to retrieve and modify data . Finally, the largest part of the **x.core** component is required for security management: this block interacts with other blocks to provide security features such as encryption and decryption of data.

### 3.4.1 Data management

XML data is managed by mean of a DOM structure that also offers support for transparent encryption. The data structure is not directly accessed by the user, and data retrieval and modification are performed by mean of the XPath and XUpdate languages.

The DOM structure has been extended to feature transparent cryptographic facilities: this enhancement allows searching and modifying data in document fragments encrypted with XML-Encryption.

Transparent encryption is an unique feature of roXanne Framework: it allows to access, search and manage information also in encrypted data blocks without requiring explicit intervention by the user to decrypt (and re-encrypt) them.

### 3.4.2 Security

Security facilities, such as cryptographic methods, are made available by a Java's JCA [24] provider. Different encryption methods are available, either relying on public key algorithms or on simmetric ciphers.

The security management part is the most important of the **x.core** component, due to the nature and the goals of the framework.

### 3.4.3 Server access

As the server and the client application can be executed on different machines, a way to allow remote communication has been devised. The protocol used is extremely simple and is based on the XML

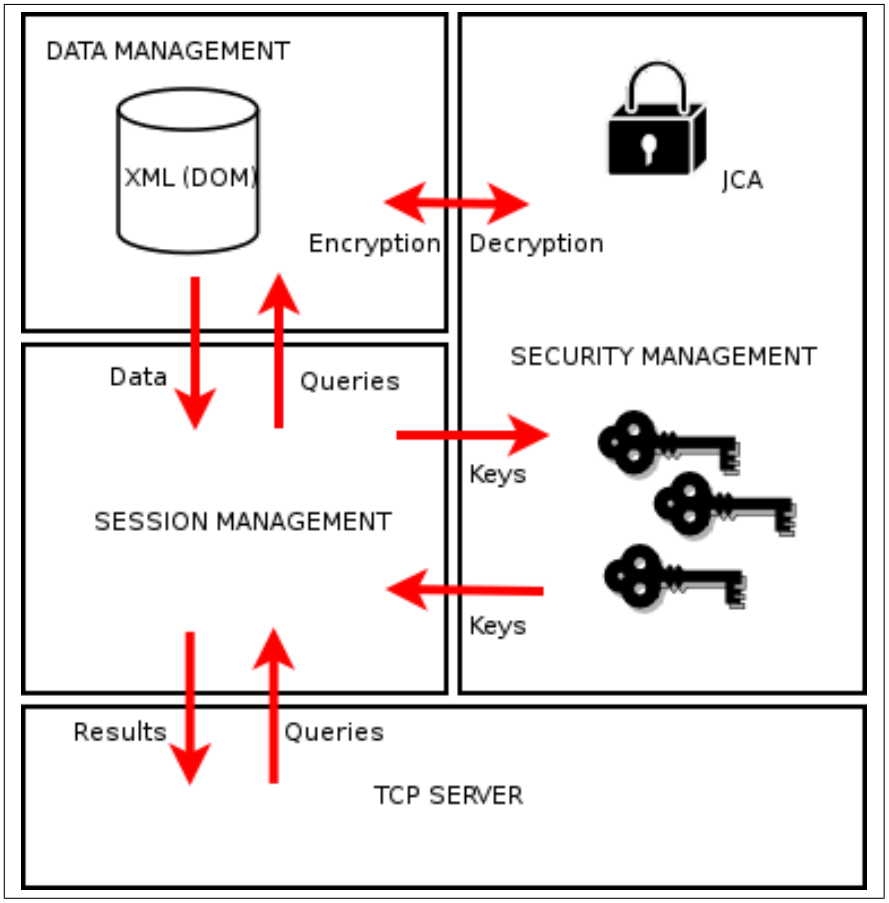


Figure 3.2: x.core component schematic overview

language sent over a TCP connection.

The server component also parses and processes requests concurrently, thanks to a multi-threaded design (which also requires a careful management of concurrent accesses).

## 3.5 x.click component

The **x.click** component is an example of possible graphical user interface and client application. It can be used to access information on the server as well as performing encryption and decryption of data.

The interface does not require knowledge of the underlying client-server protocol, and features graphical assisted procedures help the user perform some tasks, such has managing keys on the server or import and export data.

# Chapter 4

# Implementation

## 4.1 Chapter overview

This chapter describes the implementation details of the framework, with particular attention to the server component. Data management techniques, concurrent server access and the cryptographic infrastructure are also explained, as well as the protocol used for client-server communication. Additionally, useful information about framework operation and internal component interactions are also discussed. To better understand this chapter it is suggested to keep an eye on the source code; if needed, some code excerpts are provided.

## 4.2 Programming language and libraries

The whole framework has been developed using the Java Programming language. At this time, version 1.5 is available, and offers some advantages over older version, such as generic types. One important factor of choice was the availability of support libraries for the chosen programming language, and Java has a very large codebase. For the prototyping stage, the language of choice was Python; a big advantage of this language is that it allows to quickly transcribe ideas into working code, easing the development process; as concepts were tested, algorithms were coded with Java [3].

Some required parts of the framework have not been developed from scratch, but freely available software libraries have been used. By taking advantage of the open-source licenses applied to these components custom modifications have been possible, thus reducing the development effort and allowing to concentrate on this project's specific issues, such as cryptography and security.

### 4.2.1 JDOM

The JDOM library [21], developed by Jason Hunter and Brett McLaughlin, is, quite simply, a Java representation of an XML document. JDOM provides a way to represent that document for easy and efficient reading, manipulation, and writing. It has a straightforward API, is a lightweight and fast, and is optimized for the Java programmer. It's an alternative to DOM and SAX, although it integrates well with both DOM and SAX.

JDOM is not a wrapper for the W3C's DOM, or another version of DOM. JDOM is a Java-based "document object model" for XML files. JDOM serves the same purpose as DOM, but is easier to use.

JDOM is not an XML parser, like Xerces or Crimson. It is a document object model that uses XML parsers to build documents. JDOM's SAXBuilder class for example uses the SAX events generated by an XML parser to build a JDOM tree. The default XML parser used by JDOM is the JAXP-selected parser, but JDOM can use nearly any parser.

### 4.2.2 Jaxen

The Jaxen project is a Java XPath Engine [22]. Jaxen is a universal object model walker, capable of evaluating XPath expressions across multiple models. Currently supported are dom4j [33], JDOM, and all implementations of the W3C's DOM interface [36]. It was chosen for this project because of its compatibility with JDOM.

### 4.2.3 Jaxup

Jaxup [23] is Java XML Update engine developed by Erwin Bolwidt. It is used to parse and execute XUpdate queries with JDOM. This library makes use of the Jaxen engine to resolve nodes by evaluating XPath expressions.

### 4.2.4 Bouncy Castle Crypto package

The Bouncy Castle Crypto package [25] is a Java implementation of cryptographic algorithms, it was developed by the Legion of the Bouncy Castle. Use of separate JCA [24] provider is necessary because the Java classpath itself only contains the interfaces defining what methods should be provided by cryptographic libraries. Additionally current US Law Export Regulations [5] restrict the original Java JCE APIs to use solely within the United States Of America, so internationally available Sun's own JCE [26] implementation lacks some strong cryptography algorithms. In contrast, the Bouncy Castly implementation does not have any limitations and provides complete cryptographic support.

### 4.2.5 Jargs

Jargs [28] is a library aimed to parse command line arguments passed to an application. It is used in the server application to allow some command line switches and options.

## 4.3 x.core class overview

The conceptual schema of x.core component presented in the previous chapter, is reflected in the actual class organization, as shown in Figure 4.1. Each package corresponds to a different functional block:

- Classes related to client and server interaction, the TCP multi-threaded server and connection parsing are located in the `server` package.
- Classes used to manage session objects, persistent user's data on the server and concurrent requests isolation management are found in the `sessions` package.
- The `security` package contains all cryptographic related classes, such as encryption and decryption methods implementation, key and algorithms interfaces and key ring management classes.
- The `environment` package is related to data management, in particular to transparent access to encrypted fragments of the XML document.

The implementation is also based on a number of other support classes, which include the libraries presented in the previous section, classes to manage configuration inside the server and classes that initialize other components at server's startup.

## 4.4 Data management

To store the XML document during processing, the **x.core** component makes use of a DOM tree structure.. The Document Object Model is the standard method to manage this type of structured documents. Re-implementing a whole DOM library was not feasible (due to time constraints), so an already available (and well tested) implementation was chosen: JDOM. This library has several advantages over other DOM implementations, first the fact that it is freely available under an open source license (which means that its usage and its modification are legal under the terms of its license).

JDOM follows well the object oriented paradigm of Java, and resulted to be the easyiest one to modify by integrating encryption functionalities inside it, without requiring the rewrite of large parts of the library.

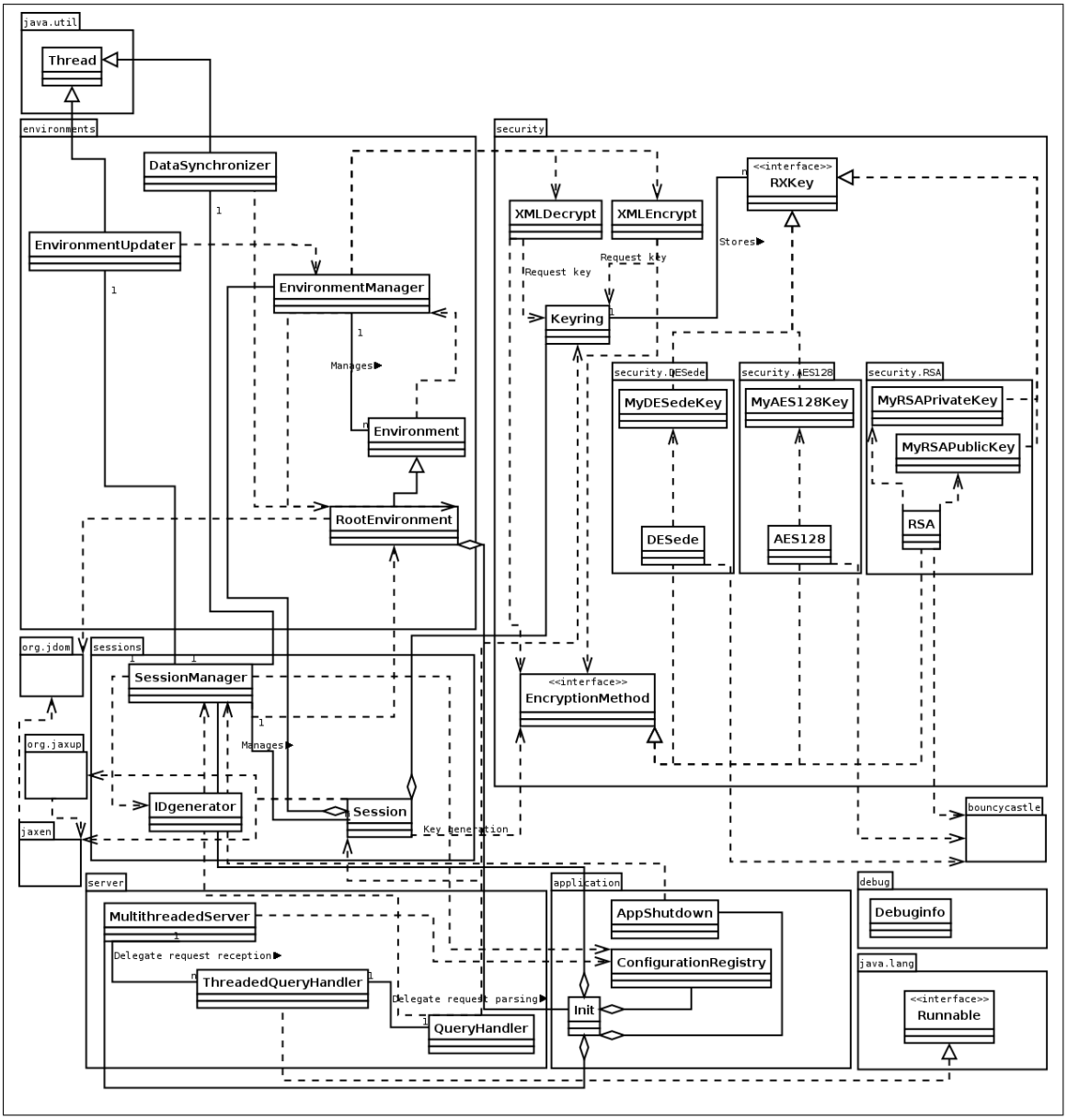


Figure 4.1: x.core component class overview

The JDOM library is used to load data from disk and to manage it in memory, as well as save it back to disk. The fact that everything is done in memory means that all data is available at any moment, without requiring access to permanent storage, but also imposes a limit on the size of manageable data. If the memory footprint of the server application grows over the available memory, the underlying operating system will start swapping on disk, slowing considerably operation. This limitation can be made less noticeable if the server application is run on a dedicated machine, with sufficent main memory (RAM) available.

The JDOM implementation is very clean, and consider every XML object in a similar manner: objects that can contain content implement the interface, and objects that can be contained must extend the `Content` class. By making this simple distinction it is possible to abstract every object, from elements to documents, to attributes, either to be a `Parent` implementation or a `Content` sub-class or both.

Attributes and content for each element are stored inside dynamic lists that extends the `java.util.AbstractList` class: for the first the `AttributeList` class is used, for the latter the `ContentList` class is used. The `AbstractList` class provides well defined methods to manage the elements they link; the basic method are `get` and `set`, which respectively return an element from the list and set an element into the list. Figure 4.2 shows an element and its content and attributes, organized in the respective lists.

To support some features required by the roXanne Framework, the JDOM library has been modified and extended; some of these changes are found in the ContentList class and in the Element class, and have been required, for example, to distinguish between encrypted or plain nodes, or to add a protection mechanism needed to ensure that some data be only readable.

A detailed analysis of these changes is given in the next section.

## 4.5 Transparent cryptography

One of the most important goals of this project was to hide to the user the fact that some data could have been encrypted. Being able to threat data as if encryption does not exist allows the user to perform, for example, XPath queries or XUpdate modifications easily and faster inside encrypted blocks. This framework performs transparent encryption and decryption of data by introducing the concept of environment. Basically, an environment is an object that contains an XML tree fragment, like an envelope contains a letter. An important fact about environments is that they are built on top of the JDOM data structure, or, better said, they integrate well with the existing structure and, most important, they are hidden to the user.

How the environments model helps hiding encryption (and decryption) of data? Consider the Example 4.5.1, which shows a simple XML document.

**Example 4.5.1** *Sample XML*

```
<mydata>
This is some unencrypted data
<substructure> ... </substructure>
<paragraph> ... </paragraph>
<EncryptedData ...>
...
</EncryptedData>
<example id="roxanne">
</mydata>
```

In figure 4.3 the corresponding ideal representation of the main environment object is depicted (in reality each environment only manages pointers to fragments of a JDOM tree and do not contain the XML document code).

The main environment object, called **root environment**, always exist, and contain links to the whole XML document (the JDOM data structure of it); the root environment is different from other environments: it is created as soon as the base XML document is loaded from disk and parsed into a DOM tree, and it cannot directly represent encrypted data (which is the purpose of other environment objects instead). Having only one root element instanced means that it is only possible to have an XML document loaded at time inside the server; if there's need to operate on more XML documents at the same time, (for the moment) the only solution is to execute multiple server applications that would listen each one on a different port.

Figure 4.4 shows the tree representation of the given XML code, so that parenthood relations are easier to understand: `mydata` element contains five children, the `EncryptedData` one is the fourth one. The `EncryptedData` element must follow the syntax given by the XML-Encryption format introduced in the previous chapter (with some limitations that will be illustrated later).

Encryption hiding comes into play when the application tries to access this data, namely as soon as the `EncryptedData` node is referenced inside the DOM tree. This request is trapped by methods added to the JDOM implementation, and, if needed decryption keys are available (i.e. loaded to the server), the corresponding decrypted node is returned instead of the encrypted one. What is internally done is that a new environment containing the decrypted data is created and from now every access to the `EncryptedData` node redirects to the root element of the XML fragment inside that environment. Figure 4.5 shows the new environment that links to the decrypted data.

From the user's point of view, the new tree representation of data becomes a little different, because the `EncryptedData` node becomes invisible, and the `people` element is now seen as a new child of `mydata` (see Figure 4.6). In the situation where the user has no right to decrypt data, the perceived data tree remains the same as in figure 4.4.

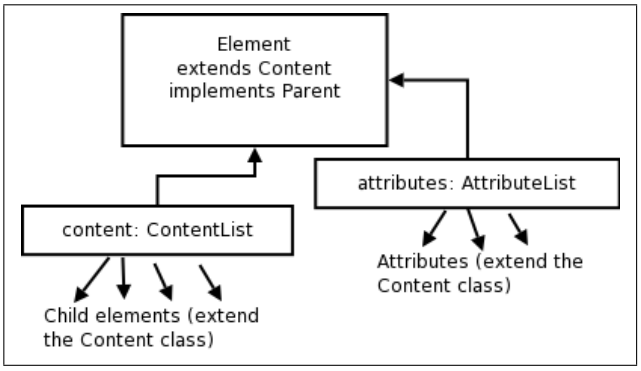


Figure 4.2: JDOM element model

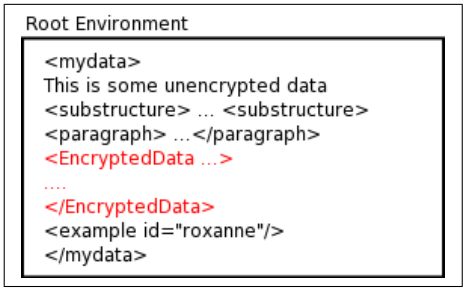


Figure 4.3: Root environment representation

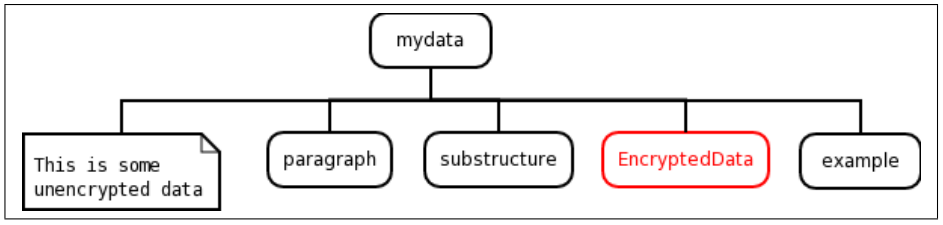


Figure 4.4: DOM schematic view

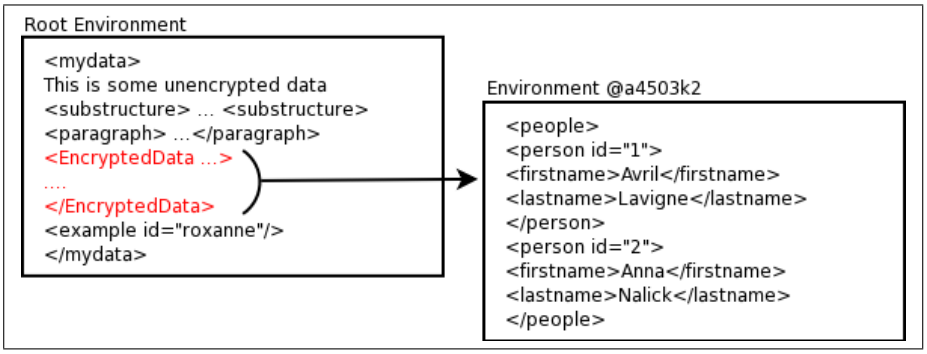


Figure 4.5: New environment for encrypted data

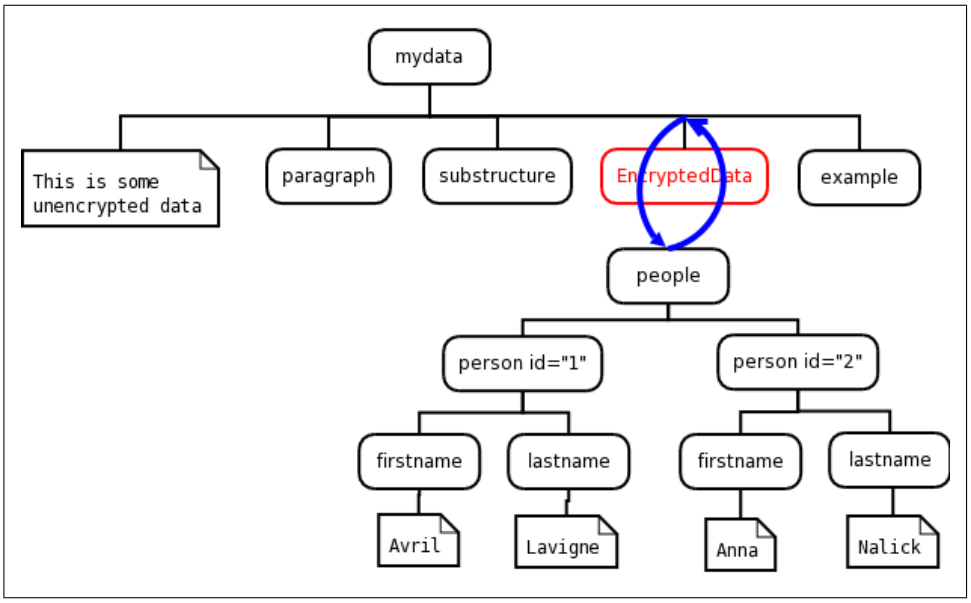


Figure 4.6: Perceived DOM structure

It should be noted that it is only possible to encrypt (and decrypt) whole single nodes (commonly referred as `http://www.w3.org/2001/04/xmlenc#Element` by the XML ENC syntax), not single attributes or multiple nodes at once. This limitation has been imposed because allowing multiple node encryption at once cannot be easily hidden (as one node becomes more nodes) and could interfere with XPath references (if an encrypted node expands to multiple node, references such as "the third child" could become invalid if used to reference right siblings of the encrypted node). Additionally encryption of attributes is not supported by the current specification of the XML ENC format (and it will presumably never be added, as it has little usefulness).

To keep track of created environments inside a session (and to be able to retrieve existing ones) a special object called `EnvironmentManager` is created; environments alone are organized hierarchically, with every environment containing a link to its parent. Environments pointers are stored inside an hashtable and referenced using the `EncryptedData` object; using this pointer as a key inside the hashtable, makes the process of checking if an environment already exist faster and does not require adding special attributes to JDOM nodes.

### 4.5.1 Parenthood references

The environment model introduces a number of problems regarding element parenthood. As the DOM tree can be "navigated" not only top-down (using child references) but also bottom-up (using the element parent reference) or over an horizontal axis (by referencing node siblings), we must ensure that the right nodes are returned. Consider, for example, the situation where the parent of the people node is requested; in reality this node has no parent, because it is directly linked inside the environment object. What is done is that in situations where the parent of a node is requested, and this node is the root element of an environment (that is, it is the root node of the decrypted XML data fragment referenced by the environment), the parent node of the corresponding `EncryptedData` node is returned.

Concerning siblings the problem is also solved, because they are accessed by first getting the parent element and then descending to the selected children.

### 4.5.2 Updating encrypted data

As everything that's accessing the JDOM (for example the XPath or the XUpdate engines) will get "fooled" by this trick and get the decrypted data instead of the encrypted one, every modification that takes place inside an environment should then be reflected back to the `EncryptedData` node.

First of all, it should be noted that modifying data inside an environment is only allowed if an encryption key that can be used to re-generate the encrypted information is loaded into the server; if only the decryption key exist, the corresponding environment gets a read-only flag that forbids every modification to data inside it (and because of the tree structure of environments, every sub-environment inherit this read-only flag).

If the environment data is modified as the client request terminates, the server must ensure that the encrypted data counterpart also contains these changes, so that further queries (from either the same client or others) will get consistent information; to do this, the server application simply has to re-create the `EncryptedData` elements from the new content. In such a situation, when the decrypted data reflects the encrypted one, the corresponding environment is said to be updated.

As environments are organized hierarchically (because of the fact that the decrypted data can also contain other `EncryptedData` elements and so on), the order by which environments get updated is extremely important: deeper environments must be updated before, because as the server updates an environment, it is supposed to get also updated and consistent `EncryptedData` elements if present within the content. Supposing that environments are organized like in example Figure 4.7, environments 3 and 5 must be updated before environment 2, and environments 4 and 2 must be updated before environment 1. Note that while updating, the "encryption hiding trick" described above is disabled, meaning that encrypted data is returned as it is. Not following a strict updating order would mean that some encrypted data would be generated from old content.

### 4.5.3 Element removal

Another problem that arise with the environment model refers to element removal: environments make the removal of elements from the JDOM tree more difficult because of a number of possible memory leaks the application should take care of. Once again these problems can be better understand by mean of a simple example.

Figure 4.8 shows a possible scenario of open environments. Suppose that four environments have been instanced (**Env1**, **Env2**, **Env3** and **Env4**; **Env1** could have been the root environment). Each environment contains a JDOM tree fragment, specifically **Env1** references a tree fragment consisting of seven elements; let upper-case letters (in this case **G** and **F**) denote `EncryptedData` elements, which then have an associated environment. Let dotted lines denote environment parenthood relations, and arcs the "virtual" link between `EncryptedData` nodes and the root of the XML data fragment inside each environment.

#### Deletion of an encrypted data element

In case of deletion of an `EncryptedData` element, two cases are possible: either the `EncryptedData` element has an associated environment or not. In the second case, the application should simply delete remove the node and its contents (the Java's garbage collection will do the rest).

In the first case, things are a little more complicated, as in reality the user never has a real access to `EncryptedData` nodes themeselves (**F** and **G** in the example), but to the root of the corresponding environments instead. So, what happens is that only the deletion of these root nodes could have been requested.

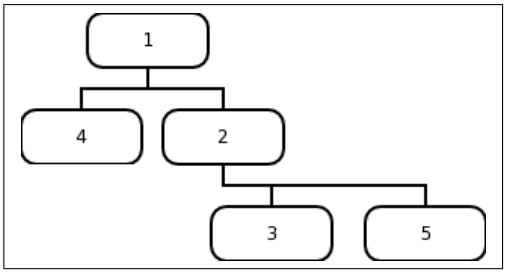


Figure 4.7: Example of environment hierachy

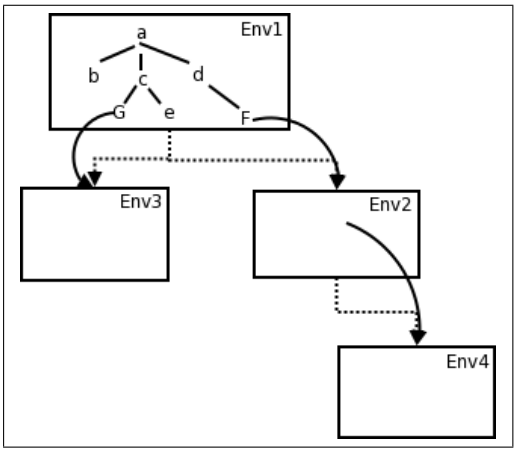


Figure 4.8: Deletion problem

What is done in case of deletion of an element is simply checking if the node to be removed is the root of an environment, and, if it is the case, the corresponding `EncryptedData` element gets deleted. Consequently the environment should also be deleted (in order to free memory), and, because of the hierachical structure also every descendant environment (because it becomes automatically unreachable). So, if for example the user is requesting deletion of the root element of environment **Env2**, node **F** gets removed along with environments **Env2** and **Env4**.

#### Deletion of plain elements

Things become much more difficult when normal elements are to be removed. Suppose that the user wants to remove element **d**. Node **d** is removed and then it cannot be accessed anymore, but memory used by it and by its children as well as memory used by **Env2** and **Env4** is not freed at all. This happens because objects representing node **d** and node **F**, as well as the whole environments **Env2** and **Env4** are still "kept alive" by the link to these two environments that exists from within the `EnvironmentManager` object and that the application is not directly aware of (remember that Java has an automatic garbage collection mechanism that removes objects from memory only if the are no more references to them).

There are two solutions to this problem: either the system checks if within the descendants of the element that should be removed there are `EncryptedData` elements with associated environments, and then remove these environments, either it marks all child environments of environment **Env2** as possibly unreachable and just gets over it (for the moment).

The solution adopted is the second one, and the reason is that checking the DOM tree downside, as the first solution requires, has an exponential complexity, where with the other one the complexity is linear (because, as it will be clear, the tree is walked bottom-up).

By the way, by forgetting the memory leak that is "pending" on the data structure, the element to be removed has been already unlinked from the tree, so from the user's point of view it does not exist anymore; only the "non-freed memory" problem remains, and this issue is solved as soon as environments are updated.

As said before, even environments that are no more accessible are still in the environment list, and that's what causes problems (they are using memory and they can be wrongly updated, requiring un-needed computation and time). To correct this situation the following solution has been adopted: for each environment that has been marked as possibly unreachable, the application follows a reverse path in the DOM tree (by getting the parent of each node encountered) starting from the root node of that environment. If at some point the root of the XML document (i.e. the root element of the root environment) is reached, it would mean that the environment is still reachable, if not (i.e. somewhere in our path we reach a node whose parent is null) it would mean that the environment is no more reachable, so the system marks it as a dead environment. Marking an environment as dead clearly affects also its descendant environments, which are also marked as dead (because there

cannot be more than one path from the root to any node).

The next step that remains to be done is removing all references to dead environments, so that Java's garbage collection can free free the occupied memory.

### 4.5.4 The environment model and JDOM

To make things work, the JDOM implementation should be made aware of environments. As stated in the previous section, the JDOM library essentially distinguishes between two type of objects: content and parent. Content are for example Element, Text or Attribute objects, and must inherit from the Content class, wheras classes implementing the Parent interface are meant to include content (Document and Element classes are examples that implement the Parent interface).

Classes implementing the Parent interface must maintain a list of their children using some kind of dynamic lists: attributes are stored in an `AttributeList` object and other content (`Element` and `Text` children) are stored using a `ContentList` object.

To support the environment model, beside some additions to the `Element` class (to correct the parenthood problem stated before), only the `ContentList` and `AttributeList` classes have been modified. An advantage of these classes is that they inherit from `java.util.AbstractList`, so that accessing and dealing with elements is always done by `get`, **set** and `remove` methods.

The `get` method has been changed so that, if an EncryptedData element is about to be returned, the framework first checks if decryption is possible; if so a new environment object is created (or the existing one is used, if applicable), and the root element of this environment is returned. Note 4.5.1 illustrates the pseudo-code executed when the application is trying to access an element (either encrypted or not), by mean of the `get` method of the `ContentList` object of the parent element. Note that the environment model can be disabled (but not by the user) so that no problems arise in situations where the real encrypted node is required: for example when updating data from an open environment the application expects that `EncryptedData` elements inside it, if present, do not get decrypted, because this would mean that encryption of sub-environments would get lost.

**Note 4.5.1** *Get method simplified pseudo-code*

```
if environment model is enabled {
if the element is encrypted {
   if can decrypt {
     if exist environment {
       return root element of existing environment
  } else {
    create environment
    return root element of environment
  }
```

```
}}}
else return element
```

If, for some reason, the decryption fails, the `EncryptedData` element is returned.

Beside the `get` method, also the `remove` method has been modified so that it behaves correctly in respect of the deletion problems introduced in the previous paragraph. Note 4.5.2 shows the pseudo-code related to the remove method (for further information please browse the source code).

**Note 4.5.2** *Remove method simplified pseudo-code*

```
if environment model is enabled {
  if element is root of an environment {
    remove EncryptedData element
    remove associated environment and sub environments
}} else {
  remove element
  mark sub environments as possibly unreachable
}
```

Additional modifications relate to modification of data inside the tree: before each modification (either content or attribute alteration) a test to ensure that data is not contained inside a read-only environment must be performed. If the user requests alteration of write protected data, no modification will be done and an error will be returned (this is done by performing a check inside every method that is supposed to alter data).

Finally, the environment model sets a limit to elements that can be deleted: the root element of the XML document cannot be deleted, nor replaced by another element (this is not a real limitation, as the root element name can be freely chosen by providing a custom default XML file or by setting the name inside the configuration file).

## 4.6 Cryptographic infrastructure

In previous sections, cryptograhy has been cited without giving any detail on how data is effectively ciphered and deciphered. When designing the cryptographic part of this framework, it was clear that this one would have been as general as possible, or not too much tied to the encryption methods implemented. This means that plugging in new encryption methods should have been as easy as possible and should not have required changing "vital" parts of the framework. For this reason, encryption algorithms and encryption keys have been abstracted to common interfaces that generalize them. Concerning the encryption algorithms, their interface is located in file `EncryptionMethod.java`, in

package `security`. Note 4.6.1 shows the methods required by this interface (exceptions have been omitted here).

**Note 4.6.1** *Encryption Method interface*

```
public interface EncryptionMethod {
  public String encryptData(String data, Object encKey);
  public byte[] decryptData(String data, Object decKey);
  public int getAlgorithmType();
  public String getAlgorithmName();
  public void generateKeyPair(Hashtable parameters);
  public void generateKey(Hashtable parameters);
  public String getPrivateKey();
  public String getPublicKey();
  public String getKey();
}
```

Each algorithm must provide these methods and then operate like a black-box. Some methods refer to public key encryption only, so they have little use for symmetric encryption algorithms (for this reason their return value is **null**), and vice-versa. Generally speaking, an encryption method should be able to provide the following features:

(i) Generate and return valid keys (using the `generateKey` or `generateKeyPair` methods, depending on the type of encryption algorithm). If some parameters are needed to generate a key (for example a password string), they can be passed by mean of a dictionary.

(ii) Encrypt data without requiring access to external data structures (the required key is given as an argument to the relative method).

(iii) Decrypt data.

This abstraction allows to deal with symmetric and public key algorithms in a very general way.

### 4.6.1 Encryption and decryption procedures

In the first chapter, the XML Encryption format has been introduced, and various aspects of encryption and decryption have also been explained. In this section the actual procedures used to generate an `EncryptedData` element from plain XML data as well as the successive decryption to create a corresponding environment will be discussed.

As said before, this framework only deals with enveloped encryption of whole elements; keys are always referenced with a `ds:KeyName` element, that contains an identifier of a key that is supposed

to be found in the keyring object (see the section about sessions that follows for more information). The limitation not to use external keys or external data is due to the fact that transparent accessing encrypted data would have been much more difficult (this doesn't mean that it is impossible to implement that, but just that it has not been done yet).

#### Decryption of data

As an EncryptedData element is about to be traversed inside the JDOM tree, the system tries to set up a new environment to store the deciphered XML fragment. The Environment Manager make use of an `XMLDecrypt` object (defined in `XMLDecrypt.java`, in package `security`) to do this. The first steps involved in this process include the parsing of the `EncryptedData` element to determine the encryption algorithm used and the encryption key needed. If the key is not available in the keyring of the current session the procedure aborts, and the `EncryptedData` node is then returned as it is.

If keys are available, data is deciphered and then parsed into a DOM tree; the root node of this document is linked as root node of the newly created environment, and additional information (such as the key name and algorithm) are saved in the environment object. Finally a pointer to the root node is returned to the JDOM object that requested the `EncryptedData` element.

If an environment object is already available, its root node is simply returned. Figure 4.9 shows an example of access of an EncryptedData node. If the user want to permanently decrypt a ciphered node, the EncryptedData node is simply replaced with the root node of the plain document, and the environment object is discarded.

#### Encryption of data

There are two situations where encryption of data is required: if the user asks for a plain node to be encrypted and when an environment is modified and changes must be reflected back to the encrypted data. Encryption is simpler that decryption, because it simply involves the creation of a new `EncryptedData` element, and the replacement of the old `EncryptedData` node with the new one. As said before, it is important that environments get updated in a reversed creation order, to ensure that the right data is put into encrypted nodes. Additionally creation of new environments is disabled during this operation.

### 4.6.2 Key management

Another thing that makes encryption hiding possible, is by having all needed keys available anytime without having to ask the user for them when an encrypted node is reached. As it will better explained later in this document, the server offers a way to keep some user information by mean of sessions. Sessions are also used to store encryption and decryption keys, without requiring that the user uploads them before every request; the object that is responsible for key management

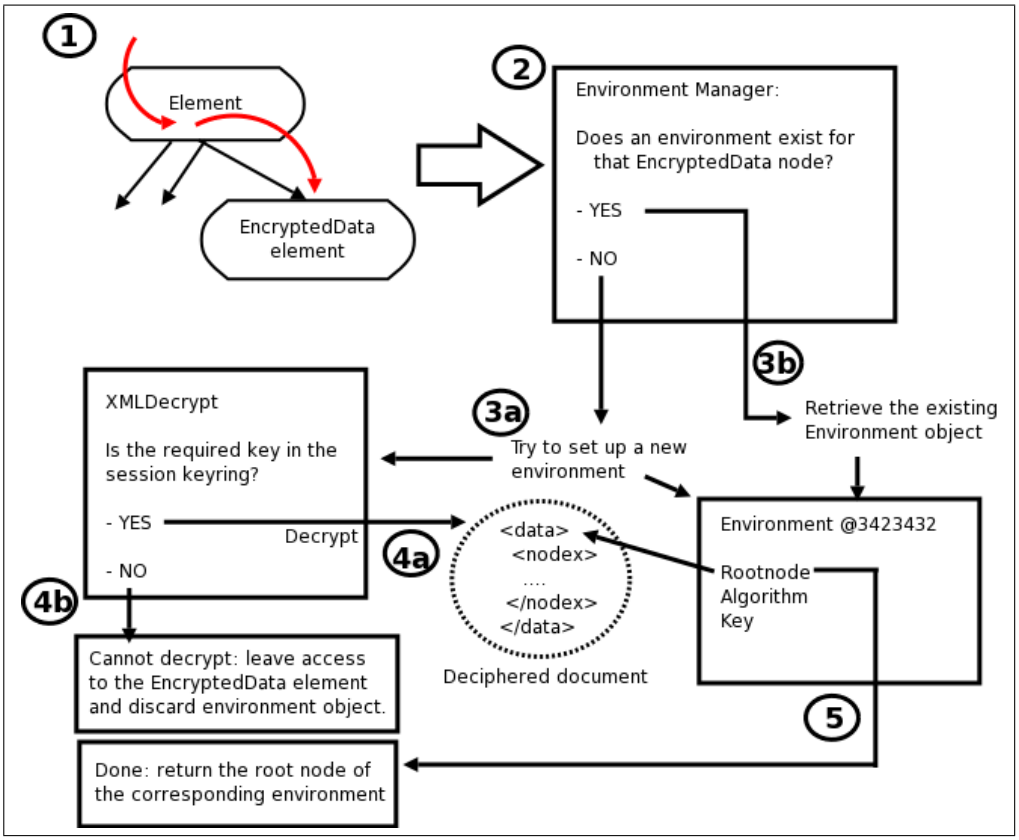


Figure 4.9: Encrypted node access

is instanced from the Keyring class (defined in file `Keyring.java`, in package `security`). Key management is really simple: as a key is uploaded by the user to the current session, the associated key object is registered within the keyring. Each key object is referenced by an user provided **key id** and the encryption method name to which they are referred; this identifier is also used to refer keys inside `EncryptedData` elements in the JDOM tree, so that, if encryption or decryption is required, the corresponding key object can be easily retrieved. The keyring itself does not make any difference between keys generated by different algorithms, because key objects have been abstracted to a common interface valid for every possible encryption method (either for symmetric or asymmetric encryption). Note 4.6.2 shows the interface devised to manage keys objects.

**Note 4.6.2** *RXKey interface*

```
public interface RXKey {
  public void decode(String data) throws InvalidKeyException;
  public String encode();
  public Object getKey();
  public void update();
}
```

As keys are uploaded to server, an object implementing the `RXKey` interface and corresponding to the selected cipher is created; received key parameters are decoded inside it by the `decode` method. The corresponding encryption algorithm can then get a valid key object by calling the `getKey` method.

On the other side, when a new key is generated, an `RXKey` implementation is used to encode it to a format suitable to be returned to the client application, by mean of the `encode` method. The format chosen to distribute keys to clients is XML, encoded with Base64: each method saves data needed to re-create key objects inside an XML document, which then is encoded using a Base64 algorithm to return a string that can be easily transmitted inside the XML reply to the user.

The encryption methods currently implemented and available inside this framework are AES 128, DESede (both symmetric algorithms) and RSA 1.5 (asymmetric algorithm). Asymmetric algorithms, such as RSA are slower that symmetric algorithms, so they are only advisable to protect a small amount of data. The RSA method cannot perform stream encryption, meaning that data must be processed in blocks: as every block is encrypted with the same key, encrypting much data, thus generating many blocks, can represent a security hole. For this reason, it is suggested to encrypt large data with an symmetric algorithm and only small parts with an asymmetric one.

### AES 128 cipher

The AES cipher (acronym of Advanced Encryption Standard, and also known as Rijndael) is a block cipher adopted as an encryption standard by the US government. It is the successor of

another cipher, Data Encryption Standard (DES), which is also somewhat available (as TripleDES, DESede) in this framework. AES was adopted by National Institute of Standards and Technology (NIST) as US FIPS PUB 197 in November 2001 after a 5-year standardisation process. This cipher was developed by two Belgian cryptographers, Joan Daemen and Vincent Rijmen, and submitted to the AES selection process under the name "Rijndael", a word made from the names of their creators.

AES is a solid cipher algorithm: until now, no successful attack against AES has been recognised. When the National Security Agency (NSA) reviewed all the AES finalists, including Rijndael, it stated that all of them were secure enough for US Government non-classified data. In June 2003, the US Government announced that AES may be also used for classified information:

> "The design and strength of all key lengths of the AES algorithm (i.e., 128, 192 and 256) are sufficient to protect classified information up to the SECRET level. TOP SECRET information will require use of either the 192 or 256 key lengths. The implementation of AES in products intended to protect national security systems and/or information must be reviewed and certified by NSA prior to their acquisition and use."[7]

This framework make use of 128 bit keys (from that, the name AES 128) but 192 or 256 bit extensions can be easily implemented. A problem with some strong encryption algorithms developed in the United States is a law that forbid their export to some countries.

The AES 128 algorithm key is generated from a character string, that can be considered as a password (as this algorithm performs symmetric encryption). The XML DTD for this kind of key is shown in 4.6.3. The AES 128 cipher is referred, within the framework, with the identifier `AES_128`.

**Note 4.6.3** *AES 128 Key structure*

```
<!ELEMENT RoXanneAES128Key (EMPTY) #REQUIRED>
<!ATTLIST RoXanneAES128Key password CDATA #REQUIRED>
```

What needs to be stored is only the password required to generate the key but not the key itself, which is recreated every time on the server. **As this algorithm uses a 128bit key, a 16 character password is required to achieve maximum security**. The generated XML document containing this parameter is then encoded and returned to the user. To re-generate a valid key object, the key data is decoded, parsed into an XML document and required parameters are extracted so that a key can be reconstructed.

### DESede cipher

DESede, also known as TripleDES or 3DES, is a block cipher based on the Data Encryption Standard (DES) cipher. The encryption procedure is composed of three steps: first a DES encryption is performed, then a DES decryption, and finally another DES encryption.

The DES cipher was selected as an official Federal Information Processing Standard (FIPS) for the United States in 1976, and was then used internationally. DES is now considered to be insecure for many applications. The main reason is the size of its keys (56 bits) being too small, meaning that the algorithm can be "cracked" easily by mean of a brute-force attack. Today it has been replaced by the AES cipher, but the evolved version of DES, DESede is also considered to be secure[8].

A DESede key has an effective size of 112 bits, which grows up to 156 and 192 bits with additional required data. As for the AES 128 key, the only parameter required to generate a DESede key is a password string (of at least 19 characters for a secure password), so its format is very similar to the previous one. The DESede cipher is referenced as `DESede`.

**Note 4.6.4** *DESede Key structure*

```
<!ELEMENT RoXanneDESedeKey (EMPTY) #REQUIRED>
<!ATTLIST RoXanneDESedeKey password CDATA #REQUIRED>
```

### RSA 1.5

RSA is an algorithm for public key encryption. It was the first algorithm known to be suitable for signing as well as encryption, and one of the first great advances in public key cryptography. Despite being developed in 1977 by Ron Rivest, Adi Shamir and Len Adleman at MIT (the letters RSA being the initials of their surnames), this algorithm is still widely used in electronic commerce protocols, and is believed to be secure given sufficiently long keys [9].

Public key cryptography requires two keys: a public one and a private one. Public keys can be genenerated from private key data, but the inverse must be very hard.

The format used to store RSA private and public keys by this framework is shown in Note 4.6.5. This cipher is identified under the name `RSA_V1_5`.

**Note 4.6.5** *RSA private and public keys DTDs*

*RSA Private key:*

```
<!ELEMENT RoXanneRSAPrivateKey (EMPTY) #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSApubExp CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAp CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAdq CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAqInv CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAdp CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAq CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAprivExp CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPrivateKey RSAmod CDATA #REQUIRED>
```

*RSA Public Key:*

```
<!ELEMENT RoXanneRSAPublicKey (EMPTY) #REQUIRED>
<!ATTLIST RoXanneRSAPublicKey RSApubExp CDATA #REQUIRED>
<!ATTLIST RoXanneRSAPublicKey RSAmod CDATA #REQUIRED>
```

Each attribute is a required parameter for the RSA algorithm. For each keypair, `RSApubExp` and `RSAmod` attributes value must be the same for either the private and the public key. Note that if keys are generated by the server, random parameter values are used.

## 4.7 Server implementation

In order to support concurrent access by multiple clients or fast consecutive access by the same client, a multi-threaded server approach has been chosen. Clients can communicate with the server by mean of a TCP connection to the server host. The connection is itself not encrypted and highly insecure, which means that it is only meant for local usage or to use in combination with, for example, a secure shell (SSH) tunnel. As the server application is loaded, a `MultithreadedServer` object is instantiated along with some other support thread; the job of this object is to accept incoming connection on a predefined port (default 8351) and then fire up a `ThreadedQueryHandler` thread to continue work (i.e. reading and processing the request), allowing the server to return listening for new connections.

By default, the `MultithreadedServer` object mantains some handler threads allocated, so that in case of multiple incoming connections, the delay that will be required to create new handler thread instances is minimized. These threads are organized in a list, and the server simply picks one of these when needed; as soon as a request terminates, the corresponding handler thread will be freed and returns available into the list. If threads are not sufficient to process incoming requests, new ones are automatically instanced.

### 4.7.1 Why not SOAP or XML-RPC ?

This framework deals almost everywhere with XML data. Altough SOAP [37] or XML-RPC [42] protocols are very well designed, they pose a big performance problem when remote methods to be called deal with large sized parameters as it is the case with queries sent to this server.

The bottleneck is due to the fact that, as both protocols are themselves based on XML, each parameter is processed as a string and each possible invalid XML character is escaped. This means that an unacceptable overhead is put in both client and server application to send and receive data. By designing a simplified client/server architecture and protocol it is possible to eliminate this problem by avoiding unneeded data conversions.

As an example, some performance tests performed during the test phase of this project have shown that a simple conversion from XML to a byte array or vice-versa will double the time required to receive and parse an incoming query.

### 4.7.2 Request handling

As the server receives a new incoming request, it will delegate its handling to a `ThreadedQueryHandler` thread, that will then continue reading from the connection until a complete request has been received (freeing the incoming port on the server). Each of these threads has a limited read buffer, meaning that received requests have a length limit; the choice of a limited size buffer has been made because of performance and security issues. First, by using a dynamic buffer, reading and storing data into it becomes orders of magnitude slower that with a fixed size array; second, by allowing virtually unlimited size queries means that a malicious client can just send garbage data that will keep the connection open for an undefined amount of time and, in the worse case, will exhaust memory on the server (as the buffer will continue growing to accept all new incoming data).

As there is no character or symbol used to mark the end of a query, as soon as a valid XML code is read, the query is considered to be completed (this is done by looking at the open and closed XML tags, up to the point where all opened tags are correctly closed). When a request is completely read, a new `QueryHandler` object is instanced, and the XML code is passed to the `execute` method so that the request is parsed. As soon as this call returns, the `ThreadedQueryHandler` will close the connection and the corresponding thread will announce itself into the waiting queue of the `MultithreadedServer` object.

#### Request parsing

An incoming XML request is passed to a `QueryHandler` object and then the `execute` method is called. This method creates a new `Session` object or retrieves and existing one, then parses the XML request and executes commands inside that session, such as adding or removing keys or performing XPath or XUpdate queries. As the request is also in XML format, it parsed into a JDOM tree (this is a good example that shows that the environment extension has no influence on "normal" XML handling). Finally an XML answer to be sent to the user is built and returned to the caller (the `ThreadedQueryHandler` object). Figure 4.10 shows a simplification of request processing.

### 4.7.3 Sessions

As already stated, users can upload encryption and decryption keys to the server and/or (as will be discussed later) define name-space bindings in order to access data. Requiring to perform these tasks for every query issued is nonsense, as it wastes bandwidth and slows operations. For this reason the concept of session has been introduced: a session is an object that is created by the server to acquire

and maintain user data across multiple requests. A sessions is created when a query is received and exists as long as the user does not request its termination or the session itself becomes "too old" and gets purged automatically. On the server side, each session is identified by an unique identifier composed of the client's host address (IP), the client login name and a random character string label (generated automatically by the server); on the client side each session is characterized by the same random string label only (as the host address and login name are of not much interest for the client itself). As soon as the first query gets answered, the client application will get, along with the reply, the ID assigned to the newly created session. In order to benefit from persistent key storage and permanent (session wide) name-space bindings, the client has to provide the same session ID in every successive request, as well as marking the session as "persistent" (by mean of a special attribute): persistent sessions get not deleted after the query has completed.

Each session has some associated objects which are not shared with other sessions: a `Keyring` object, an `EnvironmentManager` object (which is responsible of creation and management of environments) and a list of current namespace bindings (which are essential to perform XPath queries). This means that uploaded keys as well as created environments are isolated from other sessions and are no way accessible by others.

#### Namespaces

Because of the fact that the framework uses an XPath engine conform to the XPath 1.0 specification, it is required that every traversed namespace during a search to be declared beforehand. If namespaces contained in the XML data are not correctly declared, it may be possible that some result from an XPath search will be missing. For example, suppose to have the following XML document, consisting in only one element:

```
<example xmlns="http://www.unifr.ch/#examplens"/>
```

To reference this node using XPath (used by both XPath queries and XUpdate selections) it is not sufficient to select the node `example`. Instead, it is required that the namespace declared within this element to be explicitly registered inside the current session, by giving it a bogus prefix. Supposing that the namespace URI `http://www.unifr.ch/#examplens` has been registered and associated with prefix `default`, to access the previous node the following syntax should be used:

```
\slashdefault:example
```

This limitation is not a bug nor an error of the application but is defined within the XPath 1.0 semantic, which this framework implements. The same way, to access a node using a prefixed namespace, requires the latter to be registered explicitly. For example, to reference a node defined as:

```
<other xmlns:world="http://www.unifr.ch/#worldns"/>
```

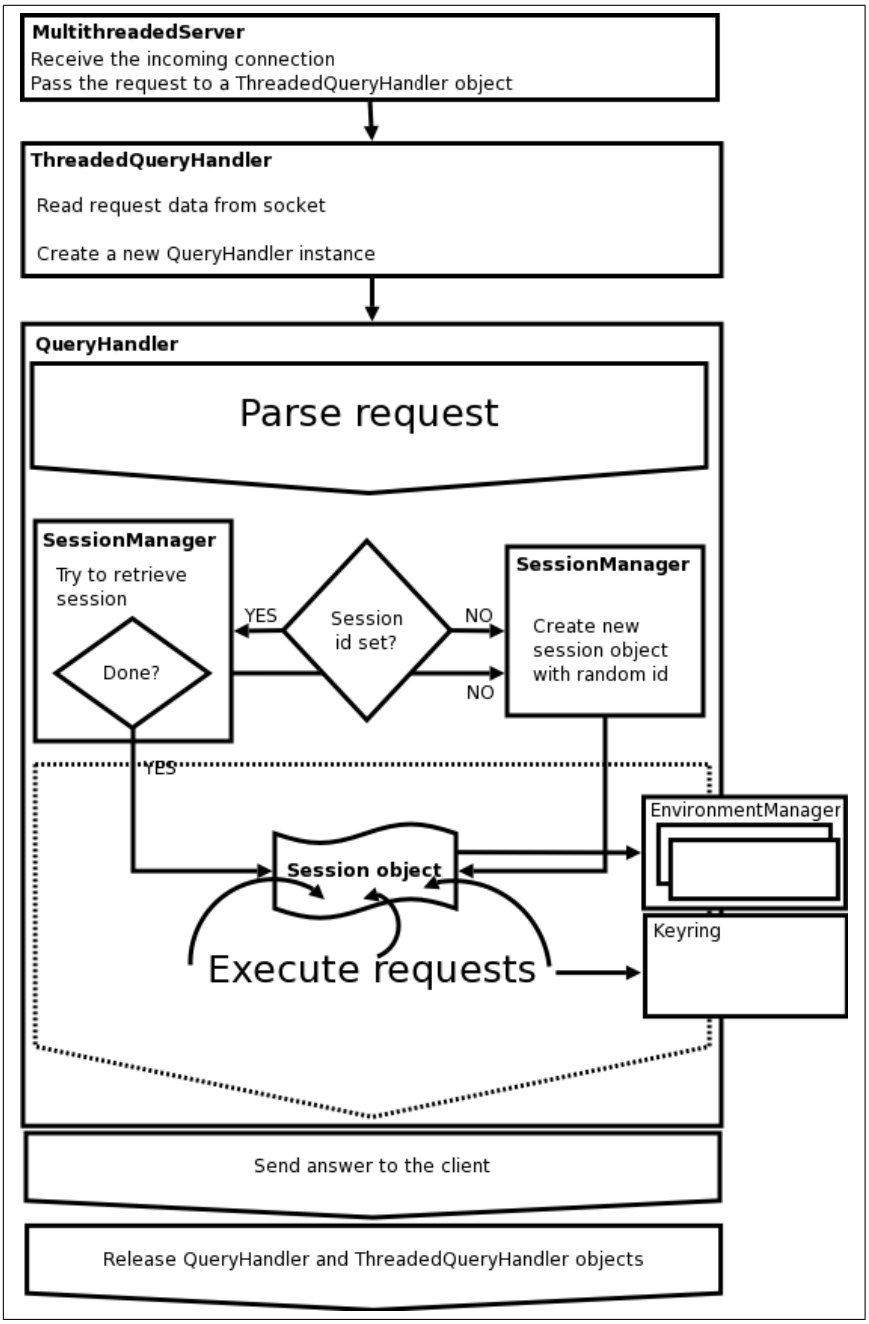


Figure 4.10: Request parsing and execution

the following syntax should be used:

```
/world:other
```

Omitting to register a namespace and/or to declare the namespace inside the XPath query will result in no element found and generally in wrong results for both XPath and XQuery commands. The XPath 2.0 [39] specification (which is currently only a draft, under active work by the W3C committee) should be more namespace's aware, and allow querying without need to define every namespace.

### 4.7.4 Concurrency

Being a client/server architecture, it is clear that some concurrency problems could arise when two or more clients try to access the data structure at the same time [13].

Pessimistic concurrency control can be successfully exploited by mean of exclusive locks; in contrast, use of shared locks is difficult and provide very little performance improvement. In an XML database data is not organized in closed and seamlessly independent entities such as tables, as it happens in traditional relational databases; therefore, assuring the ACID properties common to a transactional model while still allowing a high concurrency degree is not evident. The intrinsic hierarchical data organization and multi-dimensional nature of tree structures such as DOM, makes shared locking mostly ineffective, because a locking of every traversed node is required. Also, granularity of the locking mechanism must be lowered down to the node level; in contrast, common transactional databases can ensure mutual exclusion by locking whole tables, as tuple objects cannot extend over their two-dimensional structure.

An optimistic approach is better suited to address the concurrency problem regarding access to a DOM tree. Shadow tree fragments can be used during transaction execution and changes can be applied to the database only after a successful validation against concurrent modifications. By the way, this approach is trickier to implement, and requires deep changes in the DOM implementation in order to speed up the write process.

Therefore, data locking in the current implementation is achieved using exclusive locks [46] on the data structure: this pessimistic approach, although not the best solution for performance, gives acceptable results also under heavy load, and, as we will see, provides a good concurrency management solution also in our environment based data structure. Please note that actually there is also no concept of transaction, which means that if request's execution is at some point interrupted, data corruption could happen.

There exist only one global lock, implemented as a `java.util.concurrent.Semaphore`, declared in `SessionManager.java`:

```
public final Semaphore domAccess;
```

that is used to control access to the JDOM tree; this lock can be acquired by tree kind of objects: `Session` objects, the `Environment updater` thread and the `DataSynchronizer` thread. `Session` objects can acquire the exclusive lock on the main resource by mean of an associated query handler, which, when requested by the current query, tries to acquire the lock or block until it's available (causing the caller thread, a `ThreadedQueryHandler` object, to sleep). This lock is acquired before executing every request and before making changes to the key chain; in the latter case, locking is needed because of the lazy update of encrypted environments introduced for performance optimization (refer to next section). As soon as the request as been fullfilled, the lock is released and can be acquired by another thread. After each request all open and modified environments need to be updated so that encrypted data reflects changes; environment objects will also get discarded if the next session accessing the data tree is different from the last one.

Figure 4.11 shows a possible situation. Client A and Client B send their queries to the server; note that these two queries could also have been issued at the same time, but, because of the exclusive lock request are executed sequentially. At point 5, before the query handler manages Client B request, all environments instanced by Client A must be updated and closed, so that data on the server is consistent.

## 4.8 Performance optimization

A number of performance tricks have been added to minimize the delay caused by data encryption. Most of these enhancements have been already introduced in the previous sections, as, for example the lazy updating of encrypted environments. In this section these optimization will be discussed in detail.

### 4.8.1 Environment updater thread

Updating an environment after every modification of data is both time consuming and useless because it can be supposed that the next request has good chances of being issued by the same client, meaning that the same data could probably get modified another time, thus requiring another decryption and re-encryption procedure. Knowing that long delays due to data encryption are not well accepted by users and that there are surely many "death slots" of time between requests, it has been decided to introduce and implement a lazy updating concept. By lazy updating we mean that environment updating (that is re-generation of an encrypted data structure) is not performed immediately after data modification (namely as soon as the query terminates), but sometimes in the future, when the system has nothing better to do. Environment updating is done by a thread, called EnvironmentUpdater, that is implemented in file `EnvironmentUpdater.java` inside package `Environments` . Only one updater object is instanced by the server application, and lives up to the termination of the whole program; if there are no environments to update the thread is simply put

asleep, and it is then waken up as soon as the next query has been processed.

This thread alone does not help optimizing the encryption and decryption delays, because if the environment updater really starts doing its work after every request, the mean delay would not be minimized at all. What it is done instead, is to use the lock on the data tree as a method to decide if the thread can proceed or not; when the thread is waken up, it will first get a list of all open environments (from the last session) on what it is supposed to "work" on; for every environment to update, it then tries to access the data tree (by trying to acquire the corresponding lock): if the acquisition succeeds, environment update takes place; if not the thread clears its environment list and puts itself asleep.

In a real situation, as the updater tries to acquire the lock, it can happen that the latter is already owned by another session/query handler; in that case, there is no need that the updater thread continues its work (because environments could have been forced to close or just left open, depending on the session that required the lock).

To optimize further the benefits provided by lazy updating, a time delay between end of request and awakening of this thread has been introduced: this way a time slot for incoming connections is left open. This delay can be changed by modifying the corresponding value in the configuration file (refer to section on configuration management).

Lazy updating also introduces some problematic situations, for example when an encryption key is to be removed from the server. If the user has modified some data in an encrypted part of the XML document bound to this key, it is important that data is immediately ciphered, before the key is removed. For this reason, a lock on the DOM tree is also acquired when removing a key from the session's keyring.

### 4.8.2 Data synchronizer thread

Saving data to disk after every modification is another source of delays, which becomes noticeable when XML data becomes large. Flushing changes to disk after every modification is not very efficient, because also for alteration of just some bytes (it's better use the term characters, as the system is dealing with text XML data), the whole data structure would have been written on disk.

For this reason, the job of saving data on disk is performed by another thread, called DataSynchronizer (implemented in `DataSynchronizer.java`, in package `environments`) that saves periodically data on disk; also this threads has to wait that the lock on the DOM tree is available before accessing it.

The delay that determines when data should be saved to disk is set by mean of a global property (that can be changed by modifying the configuration file) named `SyncTime`, which defaults to 10000ms, or 10 seconds.

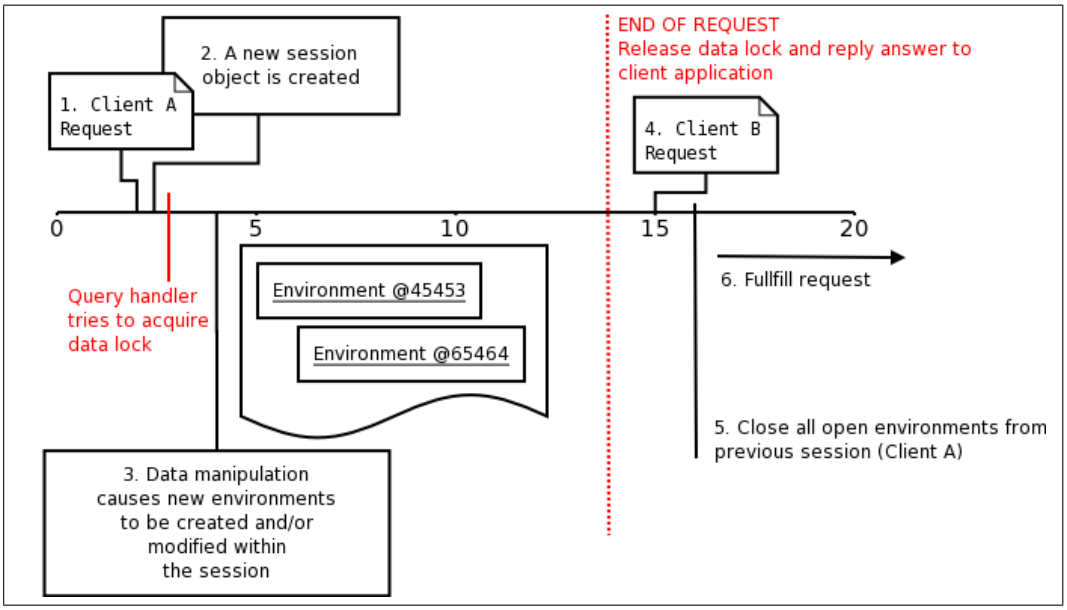


Figure 4.11: Concurrent access solved by exclusive lock

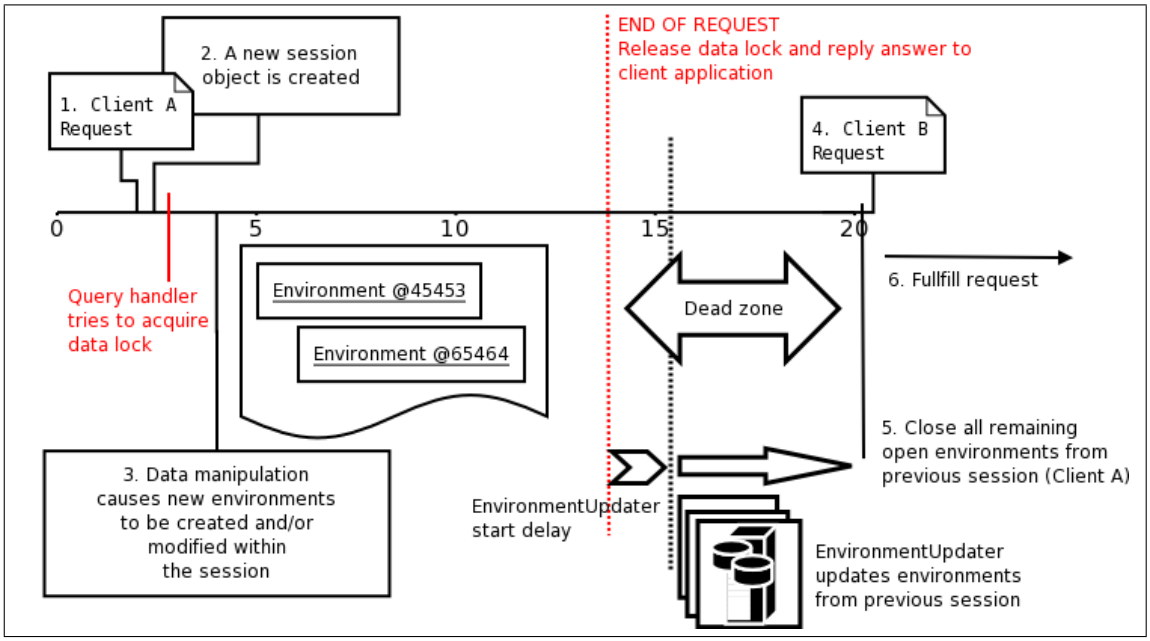


Figure 4.12: Lazy environment updating

## 4.9 Communication interface and protocol

Clients can communicate with the server over TCP, by connecting to a specified port (by default this port is 8351, but this value can be modified by mean of a key in the configuration file). Commands are issued by sending XML requests through this connection and then wait for server answer. Connections to the server are not persistent (sessions, as explained before, are), so the client application has to create a new connection for every series of requests. An advantage of not having persistent connections is that the server does not have to mantain data structures allocated, a disadvantage is that there is some little overhead involved in setting up the connection; this choice also comes from the fact that clients can issue multiple requests within a single query, and these requests are not processed in realtime, so a continous server-client interaction is not really necessary.

Using a standard TCP interface, means that clients can be implemented in whatever language supports TCP sockets (Java, C/C++, Python, Perl,...), because no special library is needed to make a custom application interact with the server.

### 4.9.1 Client to server protocol

As said before, queries are written in XML, which is simply sent over a TCP connection to the server; the syntax of RoXanne queries sent to the server must follow the syntax shown in Note 4.9.1. If syntax errors are detected in the query, the client application is not assured to receive an answer from the server (so the connection will simply timeout); a common source of errors is forgetting to send the required XML header (`<?xml version="1.0" encoding="UTF-8"?>`) before the query itself. The protocol syntax is described by mean of the DTD language.

**Note 4.9.1** *roXanne Query syntax*

```
<!ELEMENT roxannequery (keychain?, requests?)>
<!ATTLIST roxannequery login CDATA #REQUIRED
                        password CDATA #REQUIRED
                        control CDATA #IMPLIED
                        sessionId CDATA #IMPLIED
                        version CDATA #FIXED "1.0"
                        persistent (true | false) #IMPLIED>
<!ELEMENT keychain (addkey | removekey)*>
<!ELEMENT requests action*>
<!ATTLIST action type (xpath | xupdate | encrypt | genkey | decrypt | hash | status |
                        nodeinfo | nsbind | nsunbind | nsunbindall ) #REQUIRED
                        encoded (true | false) #REQUIRED
                        method CDATA #IMPLIED
```

```
                          keyID CDATA #IMPLIED
                          prefix CDATA #IMPLIED
                          uri CDATA #IMPLIED
                          depth CDATA #IMPLIED
                          password CDATA #IMPLIED>
<!ELEMENT addkey (encryption* , decryption*)>
<!ELEMENT removekey (EMPTY)>
<!ATTLIST addkey id CDATA #REQUIRED
                           method CDATA #REQUIRED>
<!ATTLIST removekey id CDATA #REQUIRED
                           method CDATA #REQUIRED
                           flag (encryption | decryption | both) #REQUIRED>
<!ELEMENT encryption (#PCDATA) #REQUIRED>
<!ELEMENT decryption (#PCDATA) #REQUIRED>
```

### Login control

Access control is managed using login names and passwords; new users can be added by modifying the server configuration file. Login name and password must be specified with the `login` and `password` attributes of the `roxannequery` element. If invalid login data is sent, the server will return nothing. Login names and password are case-sensitive, and are sent in plain format with every query. Please refer to section 4.10 for a description of how to add new users by modifying the configuration file.

**Example 4.9.1** *Setting login information*

```
<roxannequery login="foo" password="bar"/>
```

### Server control

The `control` attribute can be used by administrators to execute some special commands: setting this attribute to `shutdown` will cause the server to shutdown after a certain amount of time (the exact time is set by the `ShutdownWaitTime` configuration key), where using the value `reloadconfig` will force the server to reload its config. As execution of these special commands is reserved to users with administrator role, their use by a "normal" user have no effect.

**Example 4.9.2** *Shutting down the server or forcing reloading*

```
<roxannequery login="foo" password="bar" control="shutdown"/>
<roxannequery login="foo" password="bar" control="reloadconfig"/>
```

#### Session ID

The `sessionId` attribute can be used to supply a valid session identifier to the server in order to login to an existing session. If an invalid session identifier is provided, the server will generate a new session with a new sessionId. As sessions are automatically removed from server after a certain amount of time, if you're getting errors about unbound namespaces or invalid keys, always check that you're currently using the right session.

**Example 4.9.3** *Setting a session identifier*

```
<roxannequery login="foo" password="bar" sessionId="gjfsgbvjsfsgg"/>
```

#### Session persistency

By setting the `persistent` attribute either true or false it is possible to define if the current (or new) session should not be deleted after the end of the query (persistent set to `true`) or not. This attribute can also be used to force the end of a session by sending to the server an empty query (`roxannequery` element only) with no persistent attribute set (which then defaults to `false`).

**Example 4.9.4** *Setting session persistency*

*A persistent session:*

```
<roxannequery login="foo" password="bar" persistent="true"/>
```

*A non-persistent session:*

```
<roxannequery login="foo" password="bar"/>
```

### 4.9.2 Keychain operations

Within the `keychain` element it is possible to request modifications to the session's keychain, meaning that the user can add or remove keys. To add a new key (that should have been generated by roXanne Framework using the genkey command), an `addkey` element is used whereas to remove an existing key a `removekey` element is to be used.

#### Adding keys

To upload a new key into the session (so that it can be used by the server), append an `addkey` element to the keychain element. The `addkey` node accepts an `id` attribute which is used to give a name to the key (this name will be used to reference the key) and a `method` attribute, which specifies the encryption algorithm this key refers to. The corresponding encryption key data is to be added as text inside an `encryption` element, wheras the decryption key data has to be added as text inside

a `decryption` element; the server will only accept keys generated by this framework, and does not require that encryption and decryption keys are added at the same time. If keys with the same id already exists in the current session they are simply replaced by the new provided keys; keys having the same identifier but referring to different methods are considered as not equal and therefore can co-exist. If a read-only environment exist on the server and the corresponding encryption key gets loaded, the read-only property will be removed.

#### Removing keys

To remove a key from the current session a `removekey` element to keychain has to be appended. This element should have the following attributes: the `id` of the key to be removed, and a flag attribute specifying if only the `encryption` key or the `decryption` key or `both` keys should be removed. If there are open environments that depend on the removed key they will be either closed (if the decryption key gets unloaded) or set read-only (if the encryption key is removed).

**Example 4.9.5** *Adding and removing keys*

```
<roxannequery
login="foo" password="bar" sessionId="fjsdfjfdsj" persistent="true">
<keychain>
 <addkey id="mypersonalkey" method="DESede">
  <encryption>...</encryption>
  <decryption>...</decryption>
 </addkey>
 <removekey id="anotherkey" method="AES_128" flag="both"/>
 </keychain>
</roxannequery>
```

### 4.9.3 Performing actions

Within the request element it is possible to define an unlimited number of `action` children. It is up to the user to only send a reasonable number of requests per query. Actions are executed sequentially on the server, and for every action a result is returned (see next sub-paragraph); if an action causes a fatal error if may be possible that subsequent actions are not valid anymore and also cause errors upon execution.

#### XPath search

To execute an XPath search, an action element with the `type` attribute set to `action` must be used. The XPath location is to be specified as the content of the `node`; it is also possible to set a maximum

`depth` attribute: by setting a depth of 0, if an Element node is to be returned as result from the query, it will be returned alone (without any children attached), with a depth of 1 the node and its children will be returned, and so on. The `depth` attribute is useful if the user want to limit the length of result data, for example in case where the complete structure resulting from elements would be too large. Setting the `depth` to `-1` means infinite depth. The XPath action also accepts a boolean value `encoded` attribute, which is used (if set to `true`) to specify that the XPath location is encoded in Base64 format. Example 4.9.6 shows a sample XPath search.

**Example 4.9.6** *Sample XPath search*

```
<action type="xpath" depth="0">/documents/pictures/picture[3]/@id</action>
```

### XUpdate modification

An XUpdate modification is used to alter data on the database, and can be requested by setting the `type` attribute of an action element to `xupdate`. There are some conditions that need to be satisfied in order for an XUpdate request to be successfully executed:

- The target node (determined by the `select` attribute) need to point to a valid node inside the tree.
- The target node must not be contained inside a read-only environment.
- The target node cannot be the root node of the database.

XUpdate modifications require that the XUpdate namespace and version are specified within the action element. This means that a query will look like:

**Example 4.9.7** *XUpdate query*

```
<action type="xupdate" version="1.0"
        xmlns:xupdate="http://www.xmldb.org/xupdate">
     <!-- XUpdate modification here -->
</action>
```

### Node encryption

To encrypt a node, a `type` attribute set to `encrypt` must be used; the key to be used must be already loaded on the server, and the node to be encrypted is to be selected by mean of an XPath location. For security purposes it is only possible to encrypt a single element at time: if the XPath location points to more than one node, encryption fails. The root node of the XML document cannot be encrypted because current implementation requires that the entry point of the DOM tree to be in

plain XML. The key to be used is selected by mean of the `keyId` attribute, and the cipher by mean of the `method` attribute (available ciphers are `DESede`, `AES_128` and `RSA_V1_5`).

**Example 4.9.8** *Encryption of a node*

```
<action type="encrypt" keyId="mykey" method="RSA_V1_5">/path/to/node</action>
```

#### Node decryption

To decrypt a node, only its XPath is necessary. As for encryption, it is only possible to decrypt a single element at time, and required decryption keys must be loaded on the server: if these requirements are not met or if the specified node is not encrypted, a warning message will be returned.

**Example 4.9.9** *Decryption of a node*

```
<action type="decrypt">/path/to/encrypted/node</action>
```

#### Key generation

To generate a new key, an action of type `genkey` is used. The cipher is selected by mean of a `method` attribute. Additional parameters, such as a `password`, required by some methods (DESede and AES_128 at this time) are set by mean of a corresponding attribute of the `action` element. No key identifier is necessary, because it is only related to keys being uploaded to the server.

**Example 4.9.10** *Key generation examples*

```
<action type="genkey" method="DESede" password="mypasswordaslongaspossible"/>

<action type="genkey" method="AES_128" password="anotherlongpassword"/>

<action type="genkey" method="RSA_V1_5"/>
```

For the RSA cipher, the key parameters are automatically random generated by the server.

#### Node hashing

Using the `hash` action, it is possible to ask the server for a hash of a selected node. If descendant of the specified node include EncryptedData elements, the ciphered data is updated before hash calculation, and the hash value is computed on the encrypted content. The hash function is specified by mean of the `method` attribute; currently, available hash functions are:

- `MD2`: The MD2 message digest algorithm as defined in RFC 1319.
- `MD5`: The MD5 message digest algorithm as defined in RFC 1321.
- `SHA-1`: The Secure Hash Algorithm, as defined in Secure Hash Standard
- `SHA-256`, `SHA-384`, and `SHA-512`

The target is specified by setting the corresponding XPath expression as text content of the `action` node. If the XPath location resolves to none or multiple elements the request fails with an error message (see next section for more information about result messages from the server).

**Example 4.9.11** *Hash action*

```
<action type="hash" method="MD5">/path/to/target/node</action>
```

#### Session status

With the status action, the client can retrieve information about the current session. These information concerns available keys and bound namespaces.

**Example 4.9.12** *Requesting session status*

```
<action type="status"/>
```

#### Node information

With the `nodeinfo` action, various information about a specified node is returned to the user. This action only works if the specified XPath (to be set as text content of the `action` element) points to a single node.

**Example 4.9.13** *Node information*

```
<action type="nodeinfo">/path/to/a/node</action>
```

#### Namespace binding

As already mentioned, if the XML document makes use of namespace declarations, it is necessary to bind them into the session in order to access data correctly. Namespace bindings remain available until the session ends or namespaces are unbound by hand. To bind a namespace the `action` with `type` set to `nsbind` is to be used; additional required attributes are a `prefix` value for the namespace to be bound and a relative `uri` attribute.

**Example 4.9.14** *Binding example*

```
<action type="nsbind" prefix="sun" uri="http://www.sunblade.com/#ns2005"/>
```

#### Namespace unbinding

To unbind a bound namespace, the `nsunbind` action is used. Only the prefix of the namespace to be unbound is to be specified by mean of the `prefix` attribute.

**Example 4.9.15** *Unbind example*

```
<action type="nsunbind" prefix="sun"/>
```

#### Complete namespace unbinding

If unbinding of all bound namespaces is required, the `unbindall` action shortcut can be used (instead of unbinding namespaces one by one).

### 4.9.4 Server to client protocol

For every request submitted to the server, a corresponding reply is sent to the client. This reply contains either the information the user asked for or the return status of a requested action, and is also coded as an XML document. Complete XML answers are only sent if a valid XML query was sent to the server: if non XML or invalid login data is sent, no answer is granted. Note 4.9.2 shows the DTD of replies sent to clients.

**Note 4.9.2** *Server reply syntax*

```
<!ELEMENT roxannequeryresult (keychain?, requests?)>
<!ATTLIST roxannequeryresult sessionId CDATA #IMPLIED
                             persistent (true | false) #IMPLIED>
<!ELEMENT keychain (addkey | removekey)*>
<!ELEMENT requests action*>
```

Child elements `action` as well as `addkey` and `removekey`, are returned in the same order as they were issued in the query sent to the server. The exact syntax of action children is determined by the type of the action requested, and will be described in the next sub-sections.

### 4.9.5 Replies to keychain operations

If keychain operations where requested in the client's query, a `keychain` child will be added to the `roxannequeryresult` element. Children of this node will refer to key addition or removal, in the same order as they were requested.

### Key addition reply

An `addkey` request can be either successful or not. Failure can happen if invalid key data is sent (for example key data related to a wrong algorithm), if a non valid cipher was specified or if an invalid key identifier was chosen.

**Note 4.9.3** *Key addition reply syntax*

```
<!ELEMENT addkey (EMPTY) #REQUIRED>
<!ATTLIST addkey id CDATA #REQUIRED>
<!ATTLIST addkey method CDATA #IMPLIED>
<!ATTLIST addkey encryption (added | failed) #IMPLIED>
<!ATTLIST addkey decryption (added | failed) #IMPLIED>
```

The `id` attribute value will be equal to the one sent in the `addkey` request. If a null or invalid string identifier was used the corresponding reply will only contain the `id` attribute set to `"!invalid!"`. Depending of the key type that was sent to the server for addition, the addkey element will also define an `encryption` and/or `decryption` attribute, whose value is either set to `added` (if the key was successfully uploaed) or `failed` (if there was an error).

### Key removal reply

The reply for a key removal operation is similar to the one described for key addition. Failure can happen if an invalid key identifier was chosen, in this case the reply will only contain the `id` attribute set to `"!invalid!"`.

**Note 4.9.4** *Key removal reply syntax*

```
<!ELEMENT removekey (EMPTY) #REQUIRED>
<!ATTLIST removekey id CDATA #REQUIRED>
<!ATTLIST removekey method CDATA #IMPLIED>
<!ATTLIST removekey encryption (removed | failed) #IMPLIED>
<!ATTLIST removekey decryption (removed | failed) #IMPLIED>
```

The `method` attribute value will be the same as in the request. The `encryption` and `decryption` attributes will be set to `removed` or `failed` according to the result of the operation.

Performing actions with a key uploaded in the same query, can lead to problems: it is suggested that keys are uploaded separately from requests, so that it is always possible to check the result of each operation.

### 4.9.6 Replies to actions requests

For each requested action, a corresponding `action` child will be added to the `requests` element of the answer document. The base format of this element is listed in Note 4.9.5: the `type` attribute will correspond to the requested action type. If no actions where requested, there will be no `requests` element in the reply. If an action fails, a standard reply element, whose DTD is shown later in this document, will be added; if the action succeeds, descendants of this `action` element will depend on the action type.

**Note 4.9.5** *Action reply base DTD*

```
<!ELEMENT action #REQUIRED>
<!ATTLIST action type CDATA #REQUIRED>
```

Beside the `xpath`, `status`, `genkey` and `nodeinfo` requests, which will be described separately, other requests will return a very simple action element to inform on the status of the requested command. The reply format is defined in note 4.9.6.

**Note 4.9.6** *Action reply, extended DTD*

```
<!ELEMENT action result #REQUIRED>
<!ATTLIST action type CDATA #REQUIRED>
<!ELEMENT result (description, detail) #REQUIRED>
<!ATTLIST result exitCode CDATA #REQUIRED>
<!ELEMENT description (#PCDATA) #REQUIRED>
<!ELEMENT detail (#PCDATA) #REQUIRED>
```

The `exitCode` value can be either set to `200` if the request was fulfilled or another value if the request resulted in an error. The `description` and `detail` child will contain a description useful to determine the error that occurred. The possible exit codes and descriptions and details are listed in Appendix B.

#### XPath search result

An XPath expression will typically resolve to a list of nodes of various types, for this reason additional information must be returned for each kind of result: element nodes, attribute nodes, text nodes, etc. For each node retrieved by the XPath search, a `result` element is added inside the `action` node; this result node not only contains the actual result value but also some additional information, such as encryption references.

**Note 4.9.7** *XPath reply syntax*

```
<!ELEMENT action (result*) #REQUIRED>
<!ATTLIST action type CDATA #FIXED "xpath">
<!ATTLIST action depth CDATA #REQUIRED>
<!ATTLIST action context CDATA #REQUIRED>
<!ATTLIST action selector CDATA #REQUIRED>
<!ELEMENT result #REQUIRED>
<!ATTLIST result type (Attribute | Element | Text | Namespace) #REQUIRED>
<!ATTLIST result isInEncrypted (true | false) #REQUIRED>
<!ATTLIST result isSelfEncrypted (true | false) #IMPLIED>
<!ATTLIST result keyId CDATA #IMPLIED>
<!ATTLIST result method CDATA #IMPLIED>
<!ATTLIST result readonly (true | false) #IMPLIED>
<!ATTLIST result hasText (true | false) #IMPLIED>
<!ATTLIST result selector CDATA #REQUIRED>
<!ATTLIST result prefix CDATA #IMPLIED>
<!ATTLIST result uri CDATA #IMPLIED>
```

The `action` element will contain some additional information about the expression that has been evaluated: the maximum depth (equal to the one requested by the client), the context path and the selector. Values of the context and selector attributes, if combined, result in the given XPath expression. For example:

`/some/nodes/@*`

will retrieve all attributes of the `nodes` child of node `some`. In this case the `context` attribute will be set to `/some/nodes` and the `selector` to `@*`. Regarding each result node, whose type is defined by the `type` attribute (either to `Attribute`, `Element`, `Text` or `Namespace`), some of the described attributes (in Note 4.9.7) only refer to a certain type of result:

- Each type of result element define the `isInEncrypted` attribute to inform if the node is inside an encrypted part of the document. The value is a string representing a boolean value either `"true"` or `"false"`.
- Element results also define the `isSelfEncrypted` attribute to inform if the node is itself encrypted (namely if the node is the root of an encrypted environment).
- If an Element is encrypted, the `keyId` and `method` attributes are set correspondingly.
- If a node is inside an encrypted environment, the `readonly` attribute is set to either true or false depending on the its writability status.

- If an Element has at least a Text child, the `hasText` attribute will be set to `"true"`.
- The `selector` attribute will define a relative XPath of the result from the current context.
- The `prefix` and `uri` attributes refers to Namespace results.

### Node information reply

Information about a node is very similar to an XPath result for a single element. The only difference is that no node content is returned, and that every information is added directly inside the `action` node.

**Note 4.9.8** *XPath reply syntax*

```
<!ELEMENT action (result*) #REQUIRED>
<!ATTLIST action type CDATA #FIXED "nodeinfo">
<!ATTLIST action location CDATA #REQUIRED>
<!ATTLIST action type (Attribute | Element | Text | Namespace) #REQUIRED>
<!ATTLIST action isInEncrypted (true | false) #REQUIRED>
<!ATTLIST action isSelfEncrypted (true | false) #IMPLIED>
<!ATTLIST action keyId CDATA #IMPLIED>
<!ATTLIST action method CDATA #IMPLIED>
<!ATTLIST action readonly (true | false) #IMPLIED>
<!ATTLIST action hasText (true | false) #IMPLIED>
<!ATTLIST action prefix CDATA #IMPLIED>
<!ATTLIST action uri CDATA #IMPLIED>
```

The node location provided to the nodeinfo action request is replied as value of the location attribute. Other attributes meaning reflect what has been said for XPath queryies results.

### Status reply

A status request returns interesting information about the current session, such as loaded keys and bound namespaces. The DTD associated to replies to status requests is shown in Note 4.9.9.

**Note 4.9.9** *Status reply syntax*

```
<!ELEMENT action (keychain,nsbindings) #REQUIRED>
<!ATTLIST action type CDATA #FIXED "status">
<!ELEMENT keychain (key*) #REQUIRED>
<!ELEMENT nsbindings (nsbind*) #REQUIRED>
<!ELEMENT key (EMPTY)>
```

```
<!ELEMENT nsbind (EMPTY)>
<!ATTLIST key id CDATA #REQUIRED>
<!ATTLIST key method CDATA #REQUIRED>
<!ATTLIST key hasEncryption (true | false) #REQUIRED>
<!ATTLIST key hasDecryption (true | false) #REQUIRED>
<!ATTLIST nsbind prefix CDATA #REQUIRED>
<!ATTLIST nsbind uri CDATA #REQUIRED>
```

For every key loaded on the server, the `keychain` element will contain a `key` child that specifies the key identifier as the `id` attribute, a `method` attribute that will refer to the cipher name and two boolean attributes (`hasEncryption` and `hasDecryption`) to inform about which keys are available. For namespace bindings a `nsbind` child is added to the `nsbindings` element, and contain a `prefix` and `uri` reference.

#### Key generation reply

Reply to key generation requests will depend on the type of key, namely generated for a symmetric or asymmetric algorithm. The basic format for the associated action child is shown in Note 4.9.10.

**Note 4.9.10** *Key generation action reply syntax*

```
<!ELEMENT action (key?,encryption?,decryption?) #REQUIRED>
<!ATTLIST action type CDATA #FIXED "genkey">
<!ATTLIST action method CDATA #REQUIRED>
```

The `method` attribute will correspond to the name of the cipher the key was generated for. Children nodes attached to this element will depend on the kind of key; for symmetric keys, a single `key` child will be created, whereas for asymmetric keys both an `encryption` and a `decryption` child will be used to return the public and the private key.

**Note 4.9.11** *Symmetric and asymmetric key results syntax*

```
<!ELEMENT key (#PCDATA) #REQUIRED>
<!ELEMENT encryption (#PCDATA) #REQUIRED>
<!ELEMENT decryption (#PCDATA) #REQUIRED>
```

Data of the corresponding keys (whose format was described before in this document) will be set as content of either `key`, `encryption` or `decryption` nodes.

## 4.10 Configuration management

Certain aspects of the server application are customizable by the user, and are stored in an XML configuration file on disk. This file follows the XML syntax for Java Properties [47] objects; the associated DTD is shown in 4.10.1.

**Note 4.10.1** *java.util.Properties DTD*

```
<!ELEMENT properties (comment?, entry*)>
<!ATTLIST properties version CDATA #FIXED "1.0">
<!ELEMENT comment (#PCDATA)>
<!ELEMENT entry (#PCDATA)>
<!ATTLIST entry key CDATA #REQUIRED>
```

By default, this configuration is read from file `roxanne.conf` in the current path; to override this setting the `configfile` command line switch should be used. The format of this configuration file is pairs of key and value strings, as shown in Example 4.10.1.

**Example 4.10.1** *A configuration entry*

```
<entry key="ServerRequestTimeout">1500</entry>
```

The keys can be either String or Numerical values, and their type is automatically determined by the application. The configuration is loaded inside a single ConfigurationRegistry object (see `ConfigurationRegistry.java` in package `application`), where various methods for getting and setting configuration values are implemented, as shown in note 4.10.2.

**Note 4.10.2** *Some configuration registry methods*

```
public boolean getBooleanValue(String key, boolean defvalue)
public double getDoubleValue(String key, double defvalue)
public int getIntValue(String key, int defvalue)
public long getLongValue(String key, long defvalue)
```

Each method requires a default value to be used when no key with the given name is defined inside the configuration. This default value is not saved to disk. Table 4.1 list the available configuration keys and their default value.

| **Key** | **Type** | **Default** | **Description** |
|---|---|---|---|
| XMLdbfile | String | root.xml | File to be used as root document for the database |
| defaultRootNodeName | String | roxannedb | The default name of the root node, if no root |
| | | | document is specified |
| ServerRequestTimeout | Int | 1500 | Timeout delay for connections (in ms) |
| SyncTime | Int | 10000 | Sync on disk delay (in ms) |
| UpdaterDelay | Int | 500 | Delay between end of request and updater start |
| ServerPort | Int | 8351 | Server port for incoming connections |
| ServerMaxThreads | Int | 3 | Number of working threads pre-allocated. |
| BufferSize | Int | 16384000 | Requests buffer size (in bytes) |
| ShutdownWaitTime | Int | 10 | Seconds to wait before shutting down server |
| SessionMaxTime | Int | 1800000 | Maximum time to live for a session (in ms) |
| login(name) | String | n/a | Sets the password for username 'name' |
| role(name) | String | n/a | Sets the role of user 'name', ex. administrator |

Table 4.1: Configuration keys and default values

# Chapter 5

# User manual

## 5.1 Chapter overview

This chapter explains how to execute and use both the server and client applications. Beside the client graphical interface provided with the framework (x.click) a simple adapter library written in Python is also presented.

## 5.2 Server application

The server application is text based and can be run independently from the graphical user interface. To allow incoming connections, the server opens by defaul the TCP port 8351: this port number can be changed by modifying the configuration file. If the **x.core** component is executed on a separated host than the client, it is important that the firewall settings on the server machine allow incoming connections on the cited port.

#### Launching the server application

To execute the server application, the following conditions are needed:

- a Java 1.5 runtime that must be available and correctly configured (meaning that the Java VM can be executed by mean of the `java` command)
- some JAR and native libraries have to be found in the current class path

On an Unix like operating system (such as Linux or FreeBSD) access a shell (either in text mode or by launching a terminal application from inside an X Session), then move to the directory where the x.core application is located:

```
cd /path/to/xcore
```

A startup shell script that will set the correct class path and library path is already provided, so to launch the application simply invoke it:

```
./xcore.sh
```

To launch the application from another path in the filesystem, use a script to ensure that the path is first changed to the right directory, or execution will fail. The server component also accepts some command line arguments, which are:

```
./xcore.sh [--help]
           [--debug <debuglevel>]
           [--configfile <configfile>]
           [--xmlfile <xmlconfiguration>]
```

The `--debug` or `-d` command line option allow the user to select the debug threshold for messages shown by the server: a lower value means textttose output; default value is 1. The `--configfile` or `-c` option allows to set the configuration file to read from. The `--xmlfile` or `-x` option allows to set the base XML file used as root document in the database. The `--help` switch will show the above help message.

### Increasing JVM memory

Since all data is managed in memory, it may be possible that the server application runs out of memory during execution. To avoid that it is encouraged to add the following command line parameters to the Java executable [41].

```
-XX:NewSize=128m -XX:MaxNewSize=128m -XX:SurvivorRatio=8 -Xms512m -Xmx512m
```

This will increase the available memory inside the virtual machine to 512 MBytes. Please refer to the Java documentation (for example [43]) for additional help.

### Security concerns

As the server currently does not provide a way to use a secure connection, to ensure confidentiality of data transmission one of the following schemes must be used. Either the server application and the client reside on the same host or an SSH tunnel between the client and server machine must be used.

### Managing multiple documents

To manage multiple XML documents it is necessary to run different instances of the x.core component. Ensure that separate copies of each required file are available at different locations for each instance runned, as the XML file used to store the database is always written on the same file in the current path (named `root.xml`).

## 5.3 Client application

As a proof-of-concept, beside the server application, a client called **x.click** was developed. Accessing the server is, as we have seen, extremely easy provided that the programming language or the client application have support for TCP sockets. Several design have been considered, and what has been developed is a working application that can be used to perform almost every possible task offered by the server application. This client application has been also developed with the Java programming language; for the graphical interface the **SWT/JFace** [40] library has been chosen, because it provides consistent look-and-feel on every major platform (**Linux/GTK**, **Windows** and **MacOS/X**).

#### Launching the client application

To execute the client application, the following conditions are to be met:

- a Java 1.5 runtime must be available and correctly configured (meaning that the Java VM can be executed by mean of the `java` command)
- some JAR and native libraries have to be found in the current class path
- a running roXanne Framework server (**x.core**) must be running locally on a remote computer so that the client can connect to it

On an Unix like operating system (such as Linux), execution of the client application can be done from a terminal from inside an X Session: the user is required to move to the directory where the x.click application is located:

```
cd /path/to/xclick
```

A startup shell script that will set the correct class path and library path is already provided, so to launch the application it is necessary to simply invoke it:

```
./xclick.sh
```

To launch the application from another path in the filesystem, it is suggested to use a script to ensure that the path is first changed to the right directory, or execution will fail.

### 5.3.1 The main window

The client application is designed to operate like a terminal emulator, presenting the user a prompt where some commands can be inserted. Figure 5.1 shows how the main window appears as the program is started.

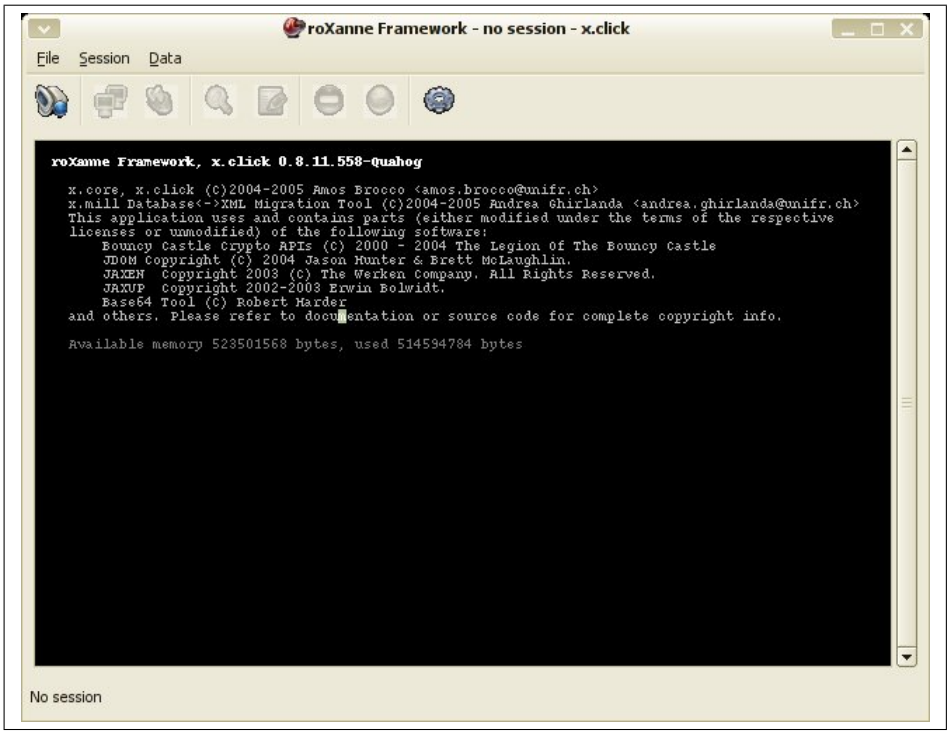


Figure 5.1: x.click main window

On the top of the window there is a menu bar to access all functions available:

- **File** menu:
  - **Setup new session** (shortcut Ctrl+Shift+N): this command is used to configure a new session, by setting some parameters needed to connect to the server.
  - **Connect** (shortcut Ctrl+Shift+C): as a session has been configured, this option will be enabled, and is used to connect to the selected server.
  - **Logout** (shortcut Ctrl+Shift+L): to properly terminate a session, this option should be used. Note that log out is only available after the user has connected to the server.
- **Session** menu:
  - **Manage keys** : this will show the key management dialog. This option is only available as the application is connected to the server.
  - **Manage namespaces** : this option will open the namespace management dialog. This option is only available if connected.
- **Data** menu:
  - **Perform XPath search** (Ctrl+F): used to enter XPath expressions.
  - **Execute XUpdate query** (Ctrl+U): launches a wizard to execute XUpdate modifications.
  - **Encrypt a node** (Ctrl+E): used to encrypt a node.
  - **Decrypt a node** (Ctrl+D): used to decrypt an encrypted node.
  - **Import** submenu:
    - **Import from database** : launch an **x.mill** session to import data from an external database.
  - **Export** submenu:
    - **Export data to XML** : used to export data from the database to an XML file.
    - **Export to database** : **x.mill** session to export data to an external database.

**The toolbar**

Frequently used operations have also an icon on the toolbar for faster access. As for menu options, some items are disabled unless the client application is connected to the server.

### 5.3.2 Creating a new session

To access data on a server it is first necessary to set up a session on the server. To do this the **Setup new session** option in the **File** menu must be selected, or the corresponding icon on the toolbar clicked. Then a dialog to set connection parameters (shown in Figure 5.3) will be displayed.

In the **Server address** text box the qualified network address or IP of the server must be specified; in the **Server port** text box the port to connect to can be changed (the default value, 8351, is shown). Finally a **Login name** and **Password** must be specified (the login name must have a corresponding **login(login name)** entry in the server configuration file).

If every required value has been set, the **Finish** button becomes clickable and allows to store the configuration and close the dialog. It is also possible to close the configuration dialog without changing previous entered values by simply pressing the **Cancel** button.

### 5.3.3 Connecting to server

As soon as the session parameters have been configured as described in the previous section, the **Connect** option in the **File** menu as well as the corresponding toolbar item become available: select one to connect to the server[1].

On success, a prompt will be shown on the window; if connection fails, an error message will appear instead, as shown in Figure 5.4.

### 5.3.4 Terminal emulator

The client application is built around a terminal emulator widget that is used to browse XML data on a server by mean of XPath expressions. To improve the user experience, a number of shortcuts have been implemented, for example auto-completion or history. The terminal emulator takes the full application window, and, as soon as the client application is connected to the server, it will show a command prompt, as shown in Figure 5.5.

After the `$>` prompt it is possible to enter an XPath expression. Pressing the enter or return key will cause the current command line to be evaluated on the server, and a result to be shown in the terminal emulator. When evaluating expressions the terminal becomes locked, and it is not possible to enter commands.

#### Results

Results from evaluations of XPath expressions are also shown in the terminal emulator. Along with each result, some information about the returned node are given. Each evaluation result will contain an header, that shows the `Depth` of the current XPath, the `Context` and the `Selector`, for example:

[1] As the client and the server do not mantain a permanent connection, the result of the **Connect** action is to create a valid session on the server where keys can be uploaded and namespaces can be defined.

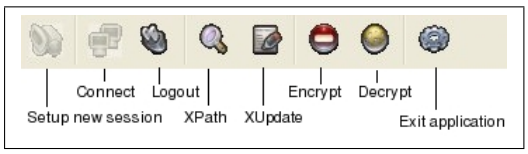


Figure 5.2: The toolbar

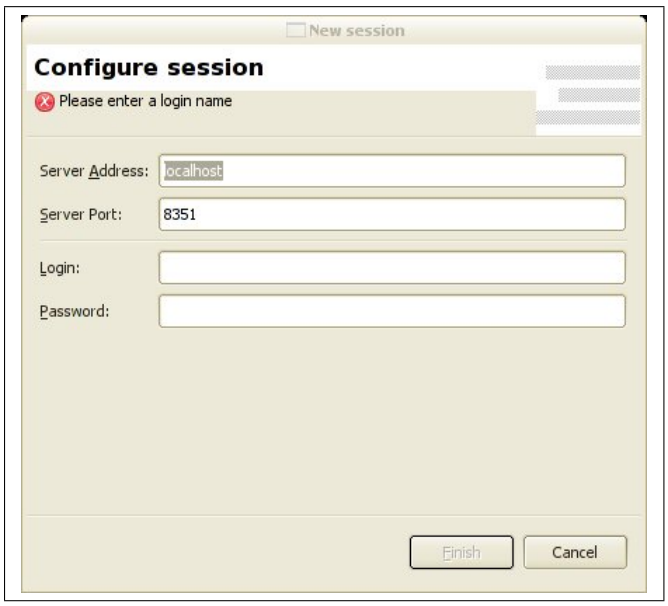


Figure 5.3: Session parameters dialog


Figure 5.4: Message shown on failed connection


Figure 5.5: Terminal emulator prompt

```
XPath
Depth:                                  0
Context:                                /
Selector:                               *
```

After this header, individual results are listed; their format depend on the type of the node returned.

**Element node result** The `Relative` XPath string will allow to refer to that Element: simply append the Context value from the result header before it to get an univoque location. The relative XPath location is also added to the auto-completion word list. If the element is located inside an encrypted part of the document, the `Is in encrypted` property will be set to `true`. The element itself is shown after these properties.

**Example 5.3.1** *An Element result*

```
XPath::Result
Result type:                            Element
Is in encrypted:                        false
Is self encrypted:                      false
Has text:                               false
Relative XPath:                         back[32]
Readonly:                               false
```

**Encrypted element node result** If decryption keys are available, `EncryptedData` elements gets expanded and replaced by the deciphered content. The shown key name refers to the identifier used to encrypt the element.

**Example 5.3.2** *An Encrypted Element result*

```
XPath::Result
Result type:                            Element
Is in encrypted:                        false
Is self encrypted:                      true
Has text:                               false
Relative XPath:                         back[32]
Readonly:                               false
Key:                                    mykey
Method:                                 RSA_V1_5
```

**Attribute node result** The relative XPath for and attributes is given using the abbreviated syntax. An attribute is set to read-only if the parent element is also read-only. The attribute value is shown after these properties.

**Example 5.3.3** *An Attribute result*

```
XPath::Result
Result type:                    Attribute
Is in encrypted:                false
Relative XPath:                 @href
Readonly:                       false
```

**Text node result** A text node has no relative XPath as it lists all available text concatenated.

**Example 5.3.4** *A Text result*

```
XPath::Result
Result type:                    Text
Is in encrypted:                false
Readonly:                       false
```

### Setting the depth

It is possible to set the depth value for XPath results by setting the global variable depth by entering the `!set` command at the prompt, for example:

```
!set depth 5
```

To retrieve the current depth value (if set) the `!get` command is used:

```
!get depth
```

The `depth` variable affects results by going deeper in the XML document; this means that, for example, setting a depth of 1 an element is returned along with its children, wheras a depth of 2 will also list children of each child. A depth of -1 means infinite recursion into children. **Warning: setting a high depth value or infinite depth can cause an high memory usage, both on the server and the client!**

#### Auto completion

To speed-up writing XPath expressions, an auto-completion feature has been integrated in the terminal; to use it, simply press the **Tab** key and the current expression will be auto-completed as good as possible. By pressing the **Tab** key on a empty line, the current context will be inserted. By pressing two times rapidily the **Tab** key a slash followed by an asterisk (the XPath short expression to get all children of the current node) will be added to the current command line. The auto-completion word list is based on the relative XPath of child nodes of the current context, meaning that it depends on the last evaluated expression.

#### History

The terminal mantains an history of the last evaluated XPath expressions: to browse them simply press the **Up** and **Down** keys at the command prompt. **Warning: browsing history entries will cause the current command line to be replaced**.

### 5.3.5 Managing namespaces

To manage namespace bindings for the current session, the **Manage namespaces** entry in the **Session** menu is to be selected. A dialog similar to the one shown in Figure 5.6 will be then displayed.

It is important that all namespaces that are likely to be traversed during XPath searches are correctly bound into the current session.

To bind a new namespace the user must click on the **New binding** button: a new line will be added to the list of namespaces, and by clicking click on it, it is possible to edit both the prefix and the URI.

To remove a binding the corresponding line in the list must be selected, then click the **Remove selected** button, on the top of the window. A confirmation request will be shown: click **Yes** to confirm deletion or **No** to cancel.

It can be useful to save the list of bound namespaces to a file, so that they can be easily re-inserted in another session: to do this, click on the **Save to file** button. A file selection dialog will appear allowing to insert the filename for the file that will hold the bindings list.

To later reload a binding list click on the **Load from file** button and select the previously saved file.

Once needed modifications have been done, click on the **Commit changes** button to reflect changes to the current session on the server, or click **Cancel** to abort them. If two or more enctries exist with the same prefix, an error message will appear when committing changes: return to the bindings list and resolve duplicates, then commit again.

### 5.3.6 Key management

In order to take advantage of transparent access to encrypted XML data on the server, as well as encrypt new plain data, keys must be loaded into the current session. First, open the key management dialog by selecting the **Manage keys** entry from the **Session** menu, or click on the corresponding icon on the toolbar: a new window will appear (see Figure 5.7).

It is possible to perform every key related operation from there: generate new keys, add them to current session or remove them. The **Manage Keys** dialog will show the current loaded keys, their identifier, the cipher they relate to, if they are encryption or decryption keys and the key data source on disk. Unlike for the namespace management dialog, operations on keys are directly executed on the server.

#### Generating a new key

To generate a new key, simply click on the Generate Key button. This will start a wizard that will guide you through the key generation process.

Select the encryption method in the corresponding combo box. Depending on method (cipher name) selected, the **Password** text box is enabled or not: actually the `DESede` and `AES_128` ciphers require a password string, whereas `RSA_V1_5` does not. If required enter a suitable password, then click **Next** to launch key generation.

As keys have been correctly generated, the **Save key** (or **Save encryption key** and **Save decryption key** for asymmetric algorithms) buttons will be enabled, click on them to save the key data in a file on disk. If an error occurs during key generation, a message will be shown in the **Operation log** box. When finished, click the **Finish** button to terminate the wizard.

#### Adding a key

To upload a key to the server, the corresponding key data file must be available on disk; click the **Add key** button in the key management dialog to show the key addition wizard (see Figure 5.10).

By mean on the **Method** list, the cipher of the key to be uploaded must be specified: if unsure, it is also possible to choose <**Autodetect**> to ask the application to auto detect the method. Next, select the **key type**: for symmetric ciphers it is possible to upload at the same time both encryption and decryption keys (which are in fact the same), by choosing **Both**.

A **key identifier** must be given to the key to be uploaded: this label is used to identify the key both in the client application and on the server.

Finally, enter the **key file path** or click on the corresponding button to open a file selection dialog to browse the filesystem and locate the required file.

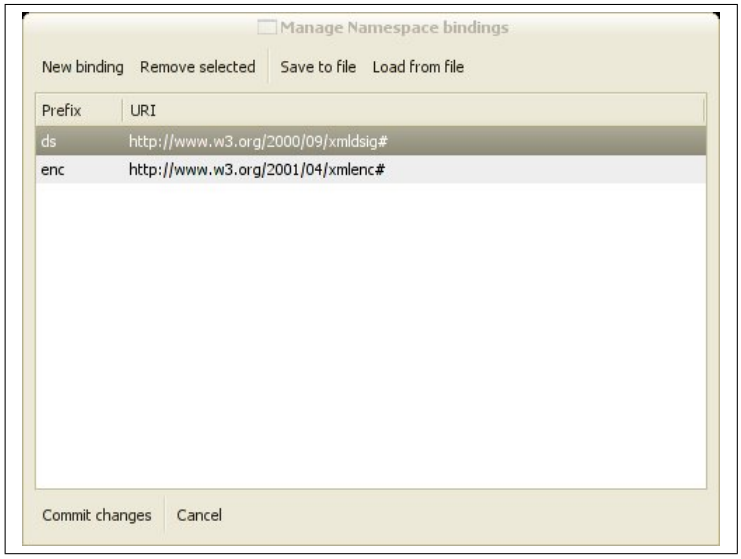


Figure 5.6: Namespaces management dialog

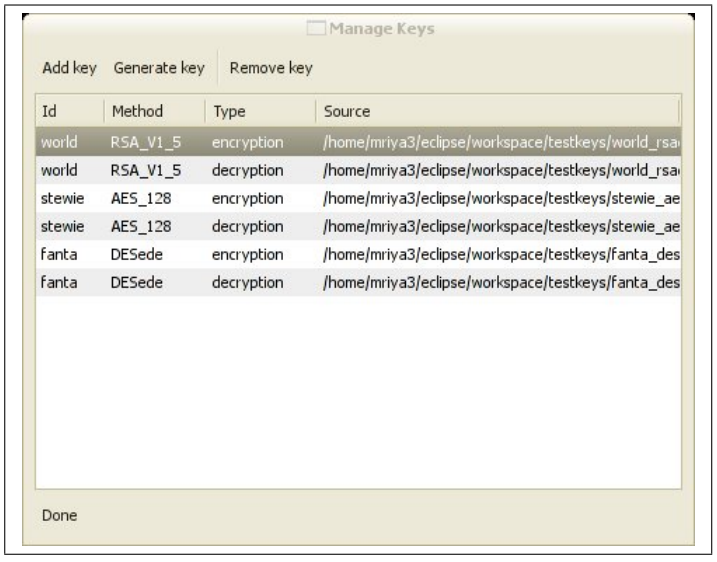


Figure 5.7: Key management dialog

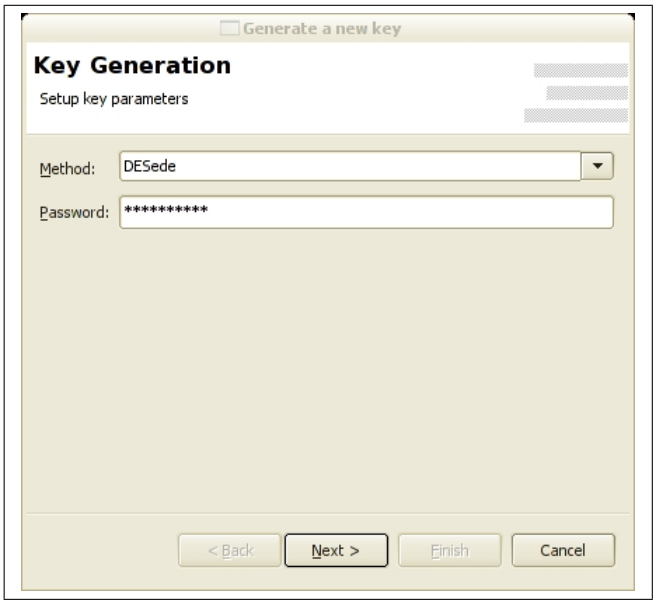


Figure 5.8: Key generation wizard (step 1)


Figure 5.9: Key generation wizard (step 2)

**Warning! Currently no check is performed to test if the decryption key works against the encryption one and vice-versa, so problems can occur (including loose of data during re-encryption) if two non matching encryption and decryption keys are using the same identifier.**

When finished configuring key parameters it is possible to click on the **Next** button to proceed with key uploading. The log text box in the next page will show information about the process. Figure 5.11 shows an example of the final page of the Add Key wizard.

#### Removing a key

To remove a key, select the corresponding entry in the key management window, then click on Remove key; a confirmation message will be shown: click **Yes** to delete the key from the server, or **No** to abort deletion.

### 5.3.7 Updating data

Modifying and updating data on the server is done by mean of an XUpdate query; to simplify the process of inserting and executing these queryies, a wizard to assist the user has been created: to access it, the **Execute XUpdate query** option from the **Data** menu must be selected.

An XUpdate query first requires a **Modification** command, that is to be selected from the corresponding list in the first page of the wizard; available modifications are:

- **xupdate:insert-before** : to insert content before the specified node
- **xupdate:insert-after** : to insert content after the specified node
- **xupdate:append** : to append new content inside an element
- **xupdate:remove** : to remove some content
- **xupdate:update** : to change the value of the specified content

The target node is selected by mean of an XPath query that points to it, that is to be entered in the **Selection** box. By default this box already contains the XPath expression pointing to the current context. The actual content to be inserted or modified on the server is to be specified in the **Content** box.

When finished, by pressing the **Next** button: the XUpdate query will be exectued, and the log of this operation shown in the last page of the wizard.

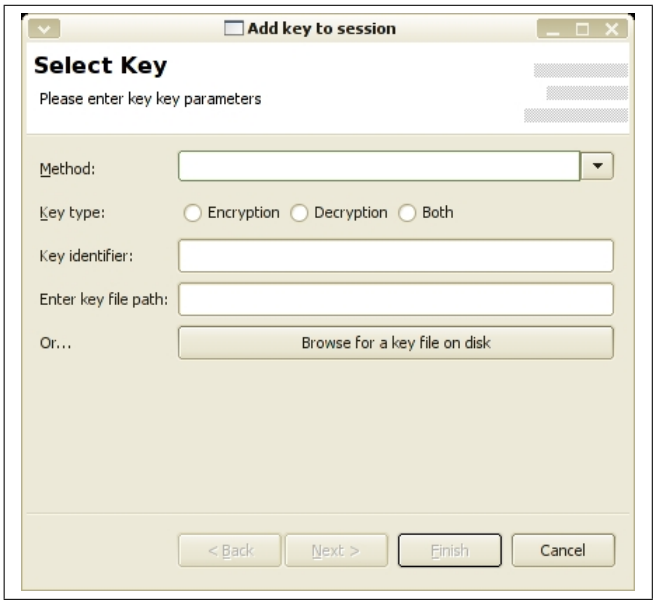


Figure 5.10: Adding a key (step 1)

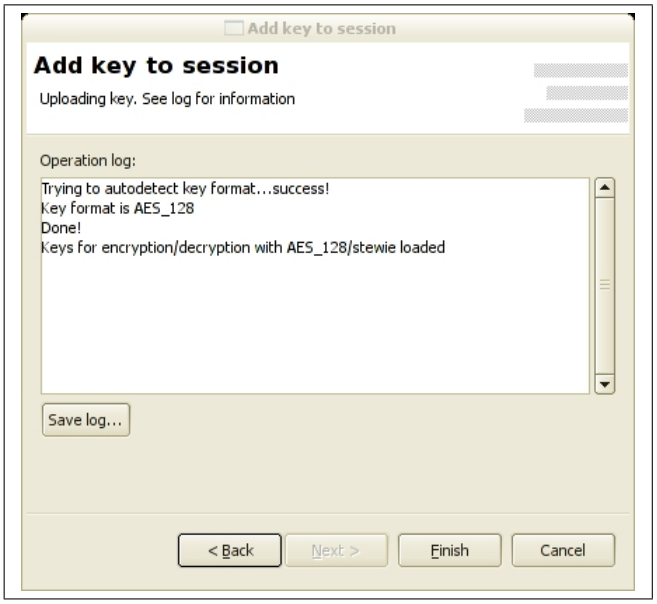


Figure 5.11: Adding a key (step 2)

### 5.3.8 Encryption

By having at least an encryption key loaded on the server, it is possible to encrypt a selected node to protect its content. To do this, select the **Encrypt a node** option from the Data menu. A dialog similar to the one shown in Figure 5.13 will appear.

By selecting the desired cipher in the **Method** list, the corresponding available keys will be shown in the **Key** list. The **Location** of the node to be encrypted must point to exactly one node (encryption of multiple nodes at the same time is disabled for security reasons): if this condition is not met, encryption will fail. It is also not possible to encrypt the root element of the database nor text or attributes. To continue and encrypt the selected element, press **Next**.

If an error occurs it will be displayed in the final page of the wizard.

### 5.3.9 Decryption

To decrypt an encrypted node, the required decryption keys must be already available on the server. Then select the **Decrypt a node** item from the **Data** menu, to display a dialog similar to the one shown in Figure 5.14.

Decryption only requires an XPath expression pointing to the encrypted node to be deciphered, then, by pressing the **Next** button, operations take place. If the node is not found or if the specified one is not encrypted, an error message will appear in the final page of this wizard.

### 5.3.10 Database interaction

Interaction with other databases is provided by the **x.mill** component, developed by Andrea Ghirlanda as his Master Project at the University of Fribourg. **x.mill** allows to export data from traditional relational databases (such as MySQL and PostgreSQL) to XML and back. Using the import and export wizard provided by the client application, it is possible to insert data coming from an external database inside the existing XML document on the **x.core** server, encrypting it if necessary, and, as soon as needed, exporting it back to an external database.

This section will only describe the x.mill component from the user point of view: please refer to the x.mill documentation to discover how the import and export features are implemented and how data is converted to the XML format.

#### Importing data

Importing data from an external database is done by selecting the **Import from database** option in the **Import** submenu of the **Data** menu. A dialog like the one in Figure 5.15 will be shown.

The required parameters to connect to the source database must be provided: server address, port, login username and password. Currently, two databases can be chosen as source: MySQL and PostgreSQL. By selecting a database name from the **Provider** list, the default port will be changed

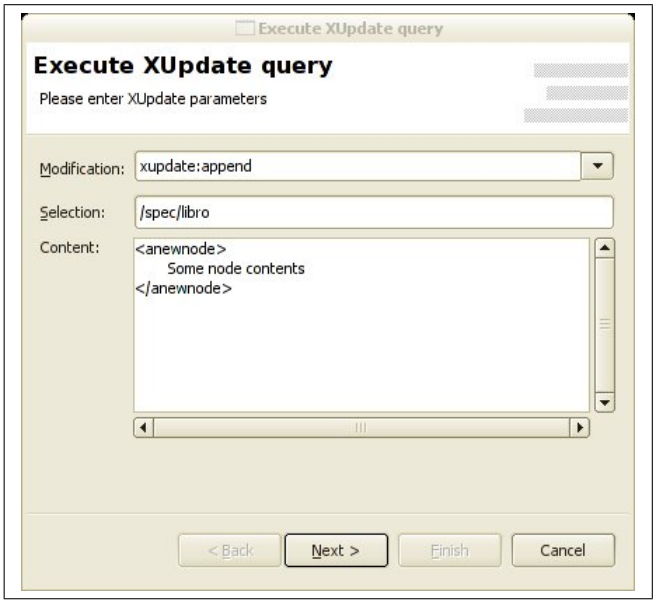


Figure 5.12: XUpdate wizard


Figure 5.13: Encryption wizard

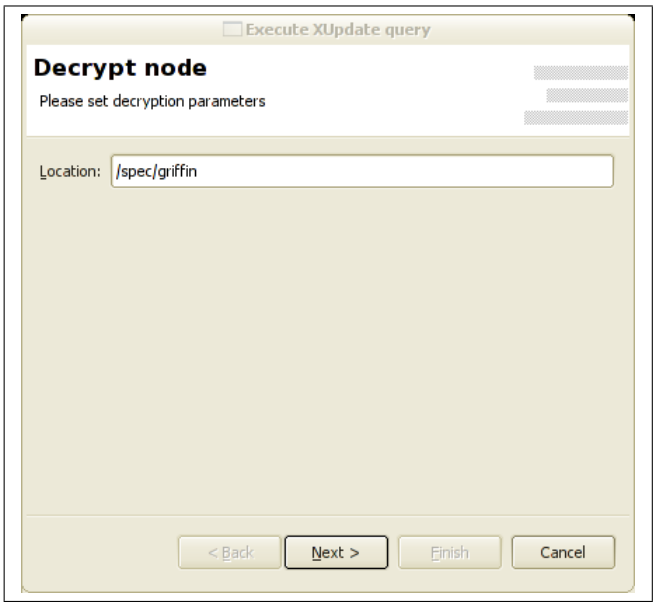


Figure 5.14: Decryption wizard

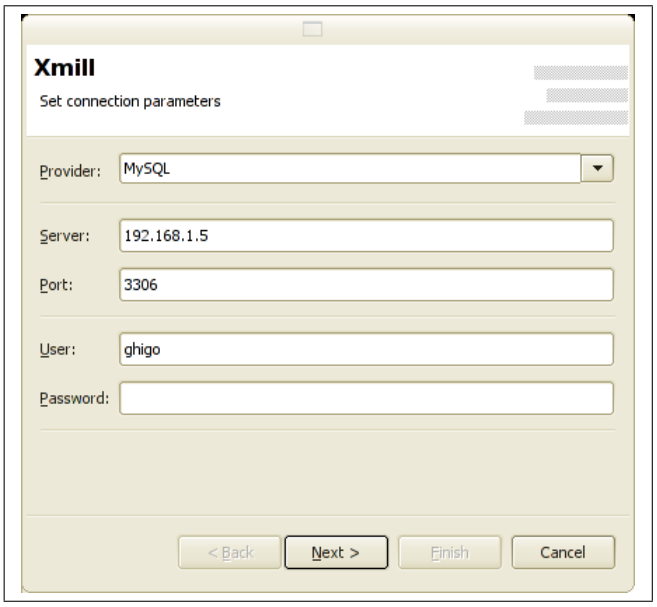


Figure 5.15: x.mill wizard (step 1)

accordingly. As soon as the wizard has collected enough information, the **Next** button will become enabled: click it to continue.

Using the specified parameters, the x.mill component connects to the provider and retrieves a list of available databases (see Figure 5.16). Only one database at time can be processed, and to continue one must be selected from the list. In this page it is also possible to set some parameters that affect the behavior of the import engine:

- **Import database information** : by enabling this, additional information (regarding supported types, precision, size, etc.) is also included in the imported data. This information can be useful for long term storage as future database systems are likely to have different data types.
- **Continue if a table is not found** : forces the export also if a table is not found on the database.
- **Continue if database is not found** : forces the import also if the database is not found.
- **Force continue on error** : tries to ignore every error and continues importing

Click **Next** to access the next page of the import wizard (see Figure 5.17).

A database can contain multiple tables: it is now possible to select which tables would get imported. Click on entries to select, press the Ctrl key and click to perform multiple choices. Then press **Next** to continue.

In the next page (Figure 5.18) it is required to decide the import policy to use: from the tables selected in the previous page it is possible to export only the structure or only the data or both. The user, as needed, must choose tables from both lists (to perform multiple choices press the Ctrl key and click on list entries).

The import process will then start; when finished, it is possible to save the extracted data to a file, by pressing on the **Save data** button (see Figure 5.19).

The destination node to which the imported data will be append is to be specified by mean of an XPath expression. To proceed with import click on the **Next** button.

Figure 5.20 shows the last page of the wizard, as data has been successfully imported into the XML document. If an error occurs, it will be displayed in the log text box.

### Exporting data

Data extracted with the x.mill tool can be exported back to an external database. This operation is also performed with the help of a graphical wizard, launched by selecting the **Export to database** item, from the **Export** submenu in the **Data** menu. A window similar to the one shown in Figure 5.21.


Figure 5.16: x.mill wizard (step 2)

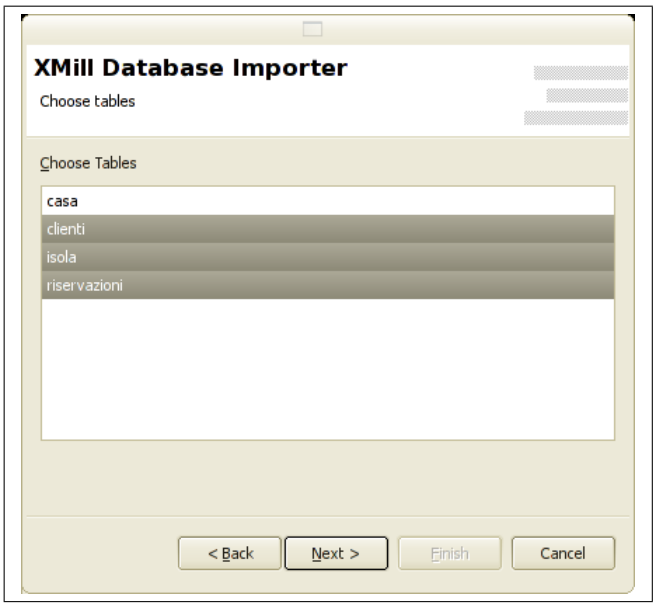


Figure 5.17: x.mill wizard (step 3)


Figure 5.18: x.mill wizard (step 4)

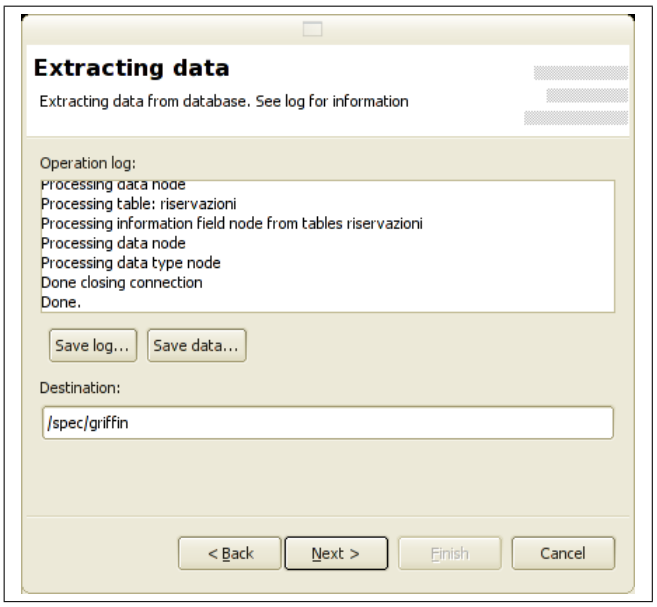


Figure 5.19: x.mill wizard (step 5)

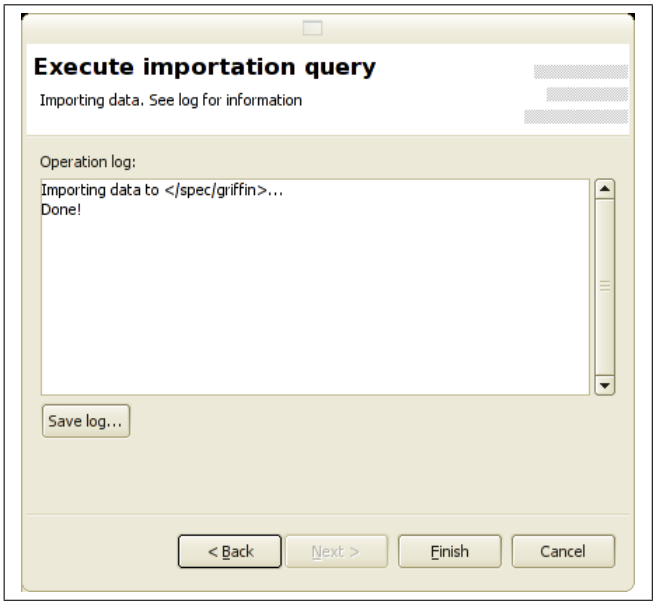


Figure 5.20: x.mill wizard (step 6)

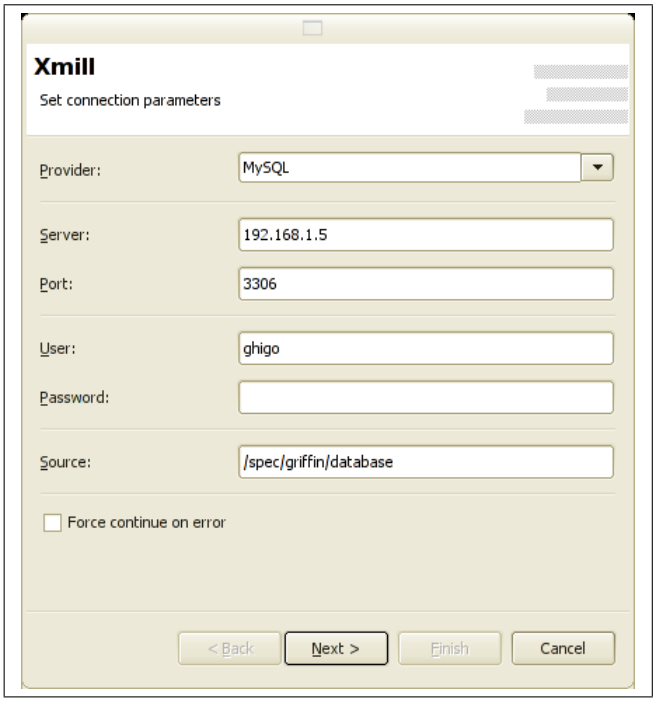


Figure 5.21: Data export wizard

Beside parameters needed to connect to the target database, an XPath expression pointing to the source **database** node inside the XML document is needed. By clicking the **Next** button the extraction process will take place, and data will be retrieved from roXanne server.

It is possible to save this data to a file by clicking on the **Save XML data to file** button (see Figure 5.22). If errors occur a message is displayed in the operation log box.

Clicking the **Next** button will start exporting data to the target database. If error occurs, and the **Force continue on error** checkbox was disabled, exporting will stop. As data migration is completed, the **Finish** button in the final page (see Figure 5.23) will be enabled: click it to close the wizard dialog.

### 5.3.11 Exporting data to file

Beside exporting XML data to an external database, it is also possible to export it to a file, by selecting the **Export data to XML** from the **Export** submenu in the **Data** menu. A dialog asking for the source element to be exported is displayed; it is also possible to set the depth of the export: a depth of -1 means that all data is exported.

## 5.4 Example client adapter in Python

Beside the client application written in Java presented in the previous section, a simple Python [1] example that can be used to connect to a server will be described in this section.

In this example a class named Pyrx is implemented: it wraps all important functions available on the **x.core** component, such as retrieving and modifying data, generating and managing keys, etc. At the end of the class definition, a test is also provided. Source code of the Pyrx class can be found in Appendix C.

Connecting and interacting with the server is extremely simple, provided that a socket library is available. Additionally a XML DOM parsing library is suggested (and required by the previous example), to further extract information from provided results. As the described class can be practically used from within a Python application, its methods deserve a more detailed description.

**createSession(serverAddress, serverPort, loginName, password)**

This method is used to create a new session on the specified server. The **serverPort** parameter can be omitted and will default to 8351.

**isConnected()**

This method will return True if the Pyrx object has already created a session on the server.

#### getSessionId()

This method returns the session identifier string associated to the open session.

#### closeSession()

This method is to be used to cleanly terminate a session on the server. If a session is not closed, it will be automatically deleted by the server after a specified time.

#### generateKey(methodName, password)

Generates and returns a new key for the specified method (cipher). If a method requires a password (like `AES_128` or `DESede`) it can be specified with the **password** parameter. The key is returned inside a list, so that keypairs are returned as a two elements list, wheras symmetric keys as a one element list.

#### addKey(methodName, keyId, keyType, keyData)

This method adds a previously generated key to the current session on the server. It is necessary to specify the method to which the key refers. Each key is identified by a string identifier that can be freely chosen by the user. The **keyType** parameter can be either "encryption" or "decryption". The **keyData** parameter contains the data returned by the generateKey method.

#### removeKey(methodName, keyId, keyType)

This will remove the specified key from the server's session. The **keyType** parameter specifies which key (encryption or decryption) to remove.

#### doXPath(xpathExpression, depth)

This performs an XPath search on the server with the provided XPath expression. The depth of results can be selected by mean of the corresponding parameter. Results are returned in form of a xml.minidom document.

#### doXUpdate(xupdateQuery)

This method executes an XUpdate query on the server and returns **True** on successful completion, or raises an exception on error.

#### encrypt(targetNode, methodName, keyId)

This encrypts the node identified by the XPath expression provided as **targetNode**, with the cipher **methodName** and the key **keyId**.

**decrypt(targetNode)**

This method decrypts the target node specified by the provided XPath expression.

**bindNS(prefix, uri)**

Use this method to bind a namespace in the current session.

**unbindNS(prefix)**

This method unbinds a namespace.

**unbindAllNS()**

This method unbinds all namespaces in the current session.

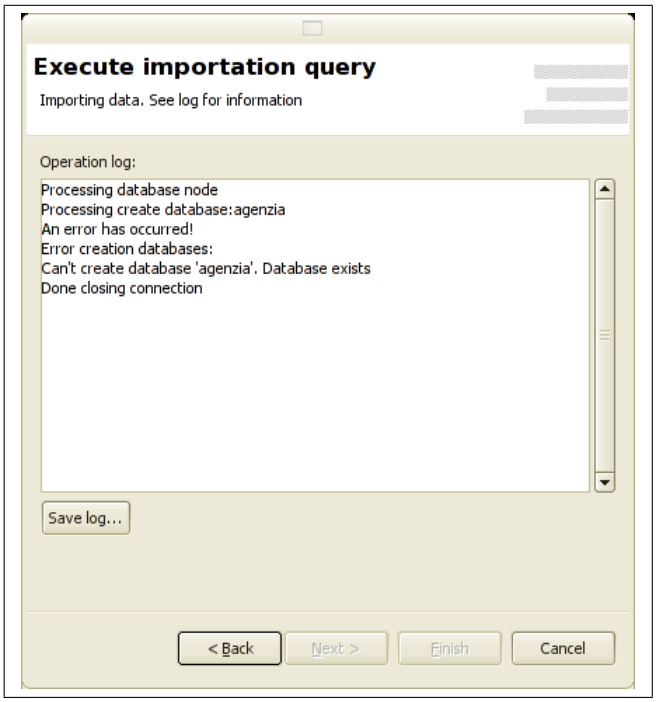


Figure 5.22: Data export wizard

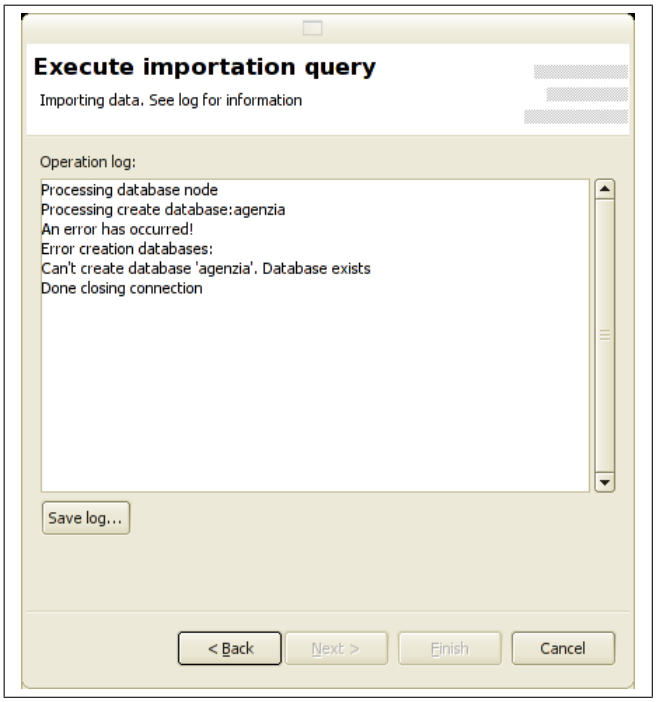


Figure 5.23: Data export wizard

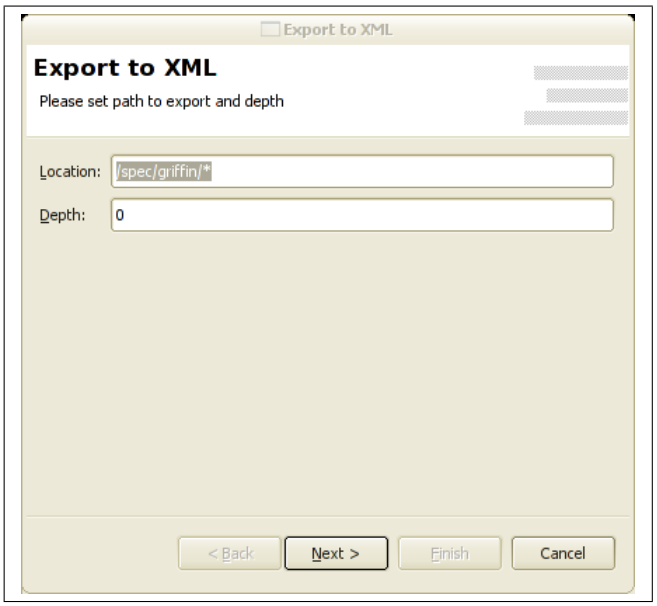


Figure 5.24: Export to file wizard

# Chapter 6

# Conclusion and Outlook

## 6.1 Chapter overview

In this final chapter, some notes about the development steps and problems encountered will be presented as well as a conclusion on the work done. Also, possible further development directions are discussed.

## 6.2 Conclusion

The goal of this thesis was to design and develop a solution for the long term data storage problem. The result of this research work, RoXanne Framework, is a viable solution because it exploits common and standardized technologies to provide a solid application.

The main contributions of this project include an approach to the long term storage problem using standard data formats and an approach to user transparent and simpler cryptography usage.

Use of a data format that can assure longevity while maintaining an high exploitability factor in the present is extremely important, because it means that data is not only safe stored for the future but also still accessible and manageable today. The XML format seems to provide sufficient guarantees in this sense.

The transparent encryption concept offers a simpler and faster way to access protected data and motivates users to encrypt their sensible information. In fact, an important factor is that the easier a technology is, more often it will be employed.

Finally graphical client interface, implemented as the **x.click** component, offers an easy to use environment to manage XML data on the server and included wizards help the user performing tasks without requiring client-server protocol knowledge.

## 6.3 Known issues

Nothing is perfect, and this application makes no exception. There are many things that have not been implemented due to time or technologies constraints.

### 6.3.1 x.core component

At this time it is not yet possible to generate an XML signature of nodes, although this part can be implemented in “client space“ and it is not necessarely to have it inside the server; one of the main problems is that JDOM currently does not have support for XML canonicalization, so adding signature generation was simply non-sense (at the moment). If a canonicalization engine will be added to JDOM in the future, implementing node signature will be extremely easy, as the cryptography infrastructure already provides asymmetric encryption support as well as hashing of XML data.

Another limitation is the fact that concurrent access is managed with a pessimistic approach by mean of an exclusive lock on the data. This solution is the simplest way to deal with parallel request execution but offers worse performances than an optimistic approach.

### 6.3.2 x.click component

Currently the interface lacks some way to save the keyring entries to disk as it can be done with namespace definitions. Also it is not possible to save session parameters, which must be entered each time the program is executed.

## 6.4 Further directions

There are many possible enhancements that can be made to the current design of the framework. Some of them have not been included in the first release of the framework as their specification would have required too much time or would have gone beyond the scope of this master thesis.

### 6.4.1 Better access control

Currently sever's access control is limited to a username and password check. This method is not very secure, because it does not enforces further security after a successful login. A better way to control access to the database would be to implement a privilege system: it would be then possible to protect some paths inside the DOM tree from undesired access, by giving read and/or write permissions only to a restricted number of people.

An additional security enhancement would be to implement a ”per-user” key system and non-transferable keys (or keys signed by the generating system).

### 6.4.2 Multiple document support

Currently the server is only able to deal with one XML document at time. Support for managing multiple documents within the same server is desirable.

### 6.4.3 XML Canonicalization and digital signature

Canonicalization of XML [45] data is essential for implementing a digital signature system for nodes. Unfortunately the current JDOM library does not support it. As XML canonicalization and digital signing (through XML DIGSIG [19]) are be available, security of the data storage framework would be greatly improved because it would allow detecting unauthorized changes in information contained in the document.

### 6.4.4 Better XML-ENC support

Currently the framework only supports a subset of all XML-ENC features. It would be interesting to add support for external key references or external cipher data references. For this to work the framework also needs to support the XPointer specification [16, 17].

### 6.4.5 Alternatives to JDOM

The JDOM library is well designed, but it shows some drawbacks when dealing with large XML files because everything is done in memory. An improved version of the framework should either enhance the JDOM library to support paging of DOM fragments and swapping from and to disk or include a custom designed DOM implementation featuring on disk data management. Cleary either solution should be designed to be (or to remain) compatible with existing support libraries such as Jaxen and Jaxup.

### 6.4.6 Modular design

Currently encryption methods are hardcoded in the framework. A better design is to implement them as pluggable modules that can be loaded at runtime. The same idea could be applied to the server package (so that additional interfaces beside the TCP server can be plugged in) and the for authentication system (Pluggable Authentication System).

### 6.4.7 Administration console

An administration console for the **x.core** component should be made available to manage permissions and multiple XML documents. Currently it is possible to manage data from the **x.click** interface, but other administrative tasks, such as access control, require changes in the configuration file. An

extension of the client-server protocol to support administration commands and operations will also ease remote server management.

## 6.5 Development roadmap

As this project started almost from scratch an initial study phase was needed to get in touch with some of the technologies involved: database design, encryption mechanism,...

**First part (October 2004 - December 2004)**

In the first months, the basic database development techniques were studied. Additionally the requirements for each component of the framework were defined, already available libraries have been chosen and needed documentation on the topic has been read.

**Second part (January 2005 - June 2005)**

In the second part of the project, the development took place: although the design model was already defined in the first part, some unattended problems arose. In the final months, as the communication protocol between the server and clients had already been defined, also the graphical user interface has been developed. Finally the x.mill component, developed by Andrea Ghirlanda, was integrated in the client application.

**Third part (June 2005 - August 2005)**

While performin some test sessions on the server and the client, this documentation was written. Also some little problems and bugs were fixed.

## 6.6 Final words

As said before this project was not very easy to implement, as much of the work was done starting from my personal ideas. Additionally, in such a long work it is easy to "lose the path", or make hard mistakes that require much time to recover from. The fact that this framework is divided in three components also required to define good interfaces: for example the x.mill component had to be designed to integrate with the x.click graphical interface.

By the way, the fact that the work was divided between two persons allowed better problem discussion and solving.

# Appendix A

# XUpdate syntax definition

```
<!ENTITY % commands \"
     xupdate:variable
   | xupdate:insert-before
   | xupdate:insert-after
   | xupdate:append
   | xupdate:update
   | xupdate:remove
   | xupdate:rename
 \">

 <!ENTITY % instructions \"
     xupdate:element
   | xupdate:attribute
   | xupdate:text
   | xupdate:processing-instruction
   | xupdate:comment
 \">

 <!ENTITY % qname \"NMTOKEN\">

 <!ENTITY % template \"
  (#PCDATA
   | %instructions;)*
 \">
```

```
<!ELEMENT xupdate:modifications (%commands;)*>
<!ATTLIST xupdate:modifications
   id          ID      #IMPLIED
   version     NMTOKEN #REQUIRED
   xmlns:xupdate CDATA   #FIXED \"http://www.xmldb.org/xupdate\"
>

<!ELEMENT xupdate:insert-before (%instructions;)*>
<!ATTLIST xupdate:insert
   select      CDATA   #REQUIRED
>

<!ELEMENT xupdate:insert-after (%instructions;)*>
<!ATTLIST xupdate:insert
   select      CDATA   #REQUIRED
>

<!ELEMENT xupdate:append (%instructions;)*>
<!ATTLIST xupdate:insert
   select      CDATA   #REQUIRED
   child       CDATA   #IMPLIED
>

<!ELEMENT xupdate:element %template;>
<!ATTLIST xupdate:element
   name        %qname; #REQUIRED
   namespace   CDATA   #IMPLIED
>

<!ELEMENT xupdate:attribute (#PCDATA)>
<!ATTLIST xupdate:attribute
   name        %qname; #REQUIRED
   namespace   CDATA   #IMPLIED
>

<!ELEMENT xupdate:text (#PCDATA)>
```

```
<!ELEMENT xupdate:processing-instruction (#PCDATA)>
<!ATTLIST xupdate:processing-instruction
   name        NMTOKEN #REQUIRED
>

<!ELEMENT xupdate:update (#PCDATA)>
<!ATTLIST xupdate:update
   select      CDATA   #REQUIRED
>

<!ELEMENT xupdate:remove EMPTY>
<!ATTLIST xupdate:remove
   select      CDATA   #REQUIRED
>

<!ELEMENT xupdate:rename (#PCDATA)>
<!ATTLIST xupdate:rename
   select      CDATA   #REQUIRED
>

<!ELEMENT xupdate:variable (#PCDATA)*>
<!ATTLIST xupdate:variable
   name        NMTOKEN #REQUIRED
   select      CDATA   #IMPLIED
>

<!ELEMENT xupdate:value-of EMPTY>
<!ATTLIST xupdate:value-of
   select      CDATA   #REQUIRED
>

<!ELEMENT xupdate:if %template;>
<!ATTLIST xupdate:if
   test        CDATA   #REQUIRED
>
```

# Appendix B

# Server's replies exit codes

This appendix lists the possible exit code values returned by the server (see also Subsection 4.9.6).

| **exitCode** | **Description** | **Detail** | **Note** |
|---|---|---|---|
| 200 | Success | Action performed successfully | If no error occurred |
| 110 | Cannot remove namespace binding | Invalid prefix | nsunbind |
| 120 | Invalid XPath query | ¡reason¿ | xupdate |
| 121 | SAXPath exception | ¡reason¿ | xupdate |
| 122 | Cannot perform XUpdate Query | ¡reason¿ | xupdate |
| 130 | Cannot set namespace binding | Prefix and/or URI null | nsbind |
| 140 | Error | Algorithm not found | hash |
| 141 | Cannot locate data to calculate hash from | No node or multiple node found at the provided location | hash |
| 142 | Error | Cannot calculate hash | hash |
| 150 | Cannot locate data to decrypt | No node or multiple node found at the provided location | decrypt |
| 151 | Warning | Node is not encrypted or insufficient rights to decrypt it | decrypt |
| 152 | Cannot perform decryption | ¡stacktrace¿ | decrypt |

Table B.1: Server's replies exit code

| exitCode | Description | Detail | Note |
|---|---|---|---|
| 160 | Key not found | Requested key was not found in keyring | encrypt |
| 161 | Cannot locate data to encrypt | No node or multiple node found at the provided location | encrypt |
| 162 | Encryption of root node not allowed | It is not possible to encrypt the root node of the database | encrypt |
| 163 | Cannot perform node encryption | <reason> | encrypt |
| 170 | Error in key generation | Cannot generate | genkey |
| 171 | Cannot generate key | Unknown algorithm | genkey |
| 180 | Cannot locate data to get info | No node or multiple node found at the provided location | nodeinfo |
| 181 | Cannot get node info | <stacktrace> | nodeinfo |
| 190 | Invalid XPath query | <reason> | xpath |
| 191 | Cannot perform XPath | <reason> | xpath |

Table B.2: Server's replies exit code (continued)

# Appendix C

# Simple Python adapter

This appendix presents a simple example of adapter written in Python. For more information please refer to Section 5.4.

```
#!/usr/bin/python
"""
        roXanne Framework simple Python interface
        (c)2005 Amos Brocco <amos.brocco@unifr.ch>

        This program is free software; you can redistribute it and/or
        modify it under the terms of the GNU General Public License
        as published by the Free Software Foundation; either version 2
        of the License, or (at your option) any later version.

        This program is distributed in the hope that it will be useful,
        but WITHOUT ANY WARRANTY; without even the implied warranty of
        MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
        GNU General Public License for more details.

        You should have received a copy of the GNU General Public License
        along with this program; if not, write to the Free Software
        Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston,
        MA  02110-1301, USA.
"""
from socket import *
import xml.dom.minidom
```

```
class Pyrx:
        def __init__(self):
                self.__sessionId = None
                self.__loginName = None
                self.__password = None
                self.__connection = None
                self.__serverAddress = None
                self.__serverPort = None

        def createSession(self, serverAddress, serverPort=8351, loginName="", \
                password=""):
                """ Creates a new session on the server """
                if serverPort is None:
                        serverPort = 8351
                self.__serverAddress = serverAddress
                self.__serverPort = serverPort
                self.__loginName = loginName
                self.__password = password
                try:
                        s = socket(AF_INET, SOCK_STREAM)
                        s.connect((self.__serverAddress, self.__serverPort))
                        s.send("""<?xml version="1.0" encoding="UTF-8"?>
                                <roxannequery login="%s" password="%s"
                                persistent="true"/>""" % \
                                (self.__loginName, self.__password))
                        data = s.recv(8192)
                        s.close()
                        doc = xml.dom.minidom.parseString(data)
                        self.__sessionId = doc.firstChild.getAttribute("sessionId")
                except:
                        raise Exception, "Cannot connect to server %s , port %s" % \
                        (self.__serverAddress, self.__serverPort)

        def isConnected(self):
                """ Returns true if the session is set """
                return self.__sessionId is not None
```

```
def getSessionId(self):
        """ Returns the session Id """
        return self.__sessionId

def __execute(self, data, resultSize=8192, mantainSession="true"):
        """ Executes a given command within the current session """
        if self.__sessionId is None:
                raise Exception, "No active session"
        try:
                s = socket(AF_INET, SOCK_STREAM)
                s.connect((self.__serverAddress, self.__serverPort))
                header = """<?xml version="1.0" encoding="UTF-8"?>
                        <roxannequery login="%s" password="%s" sessionId="%s"
                        persistent="%s">""" % \
                        (self.__loginName, self.__password, \
                        self.__sessionId, mantainSession)
                s.send(header + data + """</roxannequery>""")
                result = s.recv(resultSize)
                s.close()
                return result
        except:
                raise Exception, "Cannot communicate with server"

def closeSession(self):
        """ Closes the session on the server """
        self.__execute(data="",mantainSession="false")
        self.__sessionId = None

def generateKey(self, methodName, password=None):
        """ Generates a new key """
        cmd = """<requests><action type="genkey" \
        method="%s" password="%s"/>
        </requests>""" % (methodName, password)
        result = self.__execute(data=cmd)
        doc = xml.dom.minidom.parseString(result)
        resultNode = doc.firstChild.firstChild.firstChild.firstChild
```

```
                exitCode = resultNode.getAttribute("exitCode")
                if not exitCode == "200":
                        raise Exception, "Error on key generation"
                if len(resultNode.childNodes) == 1:
                        return [resultNode.firstChild.firstChild.nodeValue]
                else:
                        return [resultNode.firstChild.firstChild.nodeValue, \
                        resultNode.lastChild.firstChild.nodeValue]

        def addKey(self, methodName, keyId, keyType, keyData):
                """ Adds a key to the server """
                cmd = """<keychain><addkey method="%s" id="%s"> \
                <%s>%s</%s></addkey>
                </keychain>""" % (methodName, keyId, keyType, keyData, keyType)
                result = self.__execute(data=cmd)
                doc = xml.dom.minidom.parseString(result)
                addKeyNode = doc.firstChild.firstChild.firstChild
                result = addKeyNode.getAttribute(keyType)
                if not result == "added":
                        raise Exception, "Error while adding key"
                return True

        def removeKey(self, methodName, keyId, keyType):
                """ Removes a key from the server """
                cmd = """<keychain><removekey method="%s" keyId="%s" flag="%s"/>
                </keychain>""" % (methodName, keyId, keyType)
                result = self.__execute(data=cmd)
                doc = xml.dom.minidom.parseString(result)
                addKeyNode = doc.firstChild.firstChild.firstChild
                result = resultNode.getAttribute("keyType")
                if not result == "removed":
                        raise Exception, "Error while removing key"
                else:
                        return True

        def doXPath(self, xpathExpression, depth="0"):
                """ Perform an XPath search """
```

```
                cmd = """<requests><action type="xpath" depth="%s">%s</action>
                </requests>""" % (depth, xpathExpression)
                result = self.__execute(data=cmd)
                print result
                doc = xml.dom.minidom.parseString(result)
                resultNode = doc.firstChild.firstChild.firstChild.firstChild
                return resultNode.childNodes

        def doXUpdate(self, xupdateQuery):
                """ Execute an XUpdate query """
                cmd = """<requests><action type="xupdate" version="1.0"
                xmlns:xupdate="http://www.xmldb.org/xupdate">%s</action>
                </requests>""" % (xupdateQuery)
                result = self.__execute(data=cmd)
                doc = xml.dom.minidom.parseString(result)
                resultNode = doc.firstChild.firstChild.firstChild.firstChild
                exitCode = resultNode.getAttribute("exitCode")
                if not exitCode == "200":
                        description = resultNode.firstChild.nodeValue
                        detail = resultNode.lastChild.nodeValue
                        raise Exception, "Error on XUpdate execution. %s : %s" \
                        %(description, detail)
                return resultNode.childNodes

        def encrypt(self, targetNode, methodName, keyId):
                """ Encrypts a node """
                cmd = """<requests><action type="encrypt" method="%s" keyId="%s">%s
                </action></requests>""" % (methodName, keyId, targetNode)
                result = self.__execute(data=cmd)
                doc = xml.dom.minidom.parseString(result)
                resultNode = doc.firstChild.firstChild.firstChild.firstChild
                exitCode = resultNode.getAttribute("exitCode")
                if not exitCode == "200":
                        description = resultNode.firstChild.nodeValue
                        detail = resultNode.lastChild.nodeValue
                        raise Exception, "Error on node encryption. \
                        %s : %s" %(description, detail)
```

```
        return True

    def decrypt(self, targetNode):
        """ Decrypts a node """
        cmd = """<requests><action type="decrypt">%s</action>
        </requests>""" % (targetNode)
        result = self.__execute(data=cmd)
        doc = xml.dom.minidom.parseString(result)
        resultNode = doc.firstChild.firstChild.firstChild.firstChild
        exitCode = resultNode.getAttribute("exitCode")
        if not exitCode == "200":
                description = resultNode.firstChild.nodeValue
                detail = resultNode.lastChild.nodeValue
                raise Exception, "Error on node decryption. \
                %s : %s" %(description, detail)
        return True

    def bindNS(self, prefix, uri):
        """ Binds a namespace """
        cmd = """<requests><action type="nsbind" prefix="%s" uri="%s"/>
        </requests>""" % (prefix, uri)
        result = self.__execute(data=cmd)
        doc = xml.dom.minidom.parseString(result)
        resultNode = doc.firstChild.firstChild.firstChild.firstChild
        exitCode = resultNode.getAttribute("exitCode")
        if not exitCode == "200":
                description = resultNode.firstChild.nodeValue
                detail = resultNode.lastChild.nodeValue
                raise Exception, "Error on namespace binding. %s : %s" \
                %(description, detail)
        return True

    def unbindNS(self, prefix):
        """ Unbinds a namespace """
        cmd = """<requests><action type="nsunbind" prefix="%s"/>
        </requests>""" % (prefix)
        result = self.__execute(data=cmd)
```

```
                doc = xml.dom.minidom.parseString(result)
                resultNode = doc.firstChild.firstChild.firstChild.firstChild
                exitCode = resultNode.getAttribute("exitCode")
                if not exitCode == "200":
                        description = resultNode.firstChild.nodeValue
                        detail = resultNode.lastChild.nodeValue
                        raise Exception, "Error on namespace unbinding. %s : %s" \
                        %(description, detail)
                return True

        def unbindAllNS(self):
                """ Unbinds all namespaces """
                cmd = """<requests><action type="nsunbindall"/></requests>"""
                result = self.__execute(data=cmd)
                doc = xml.dom.minidom.parseString(result)
                resultNode = doc.firstChild.firstChild.firstChild.firstChild
                exitCode = resultNode.getAttribute("exitCode")
                if not exitCode == "200":
                        description = resultNode.firstChild.nodeValue
                        detail = resultNode.lastChild.nodeValue
                        raise Exception, "Error on all namespaces unbinding. \
                        %s : %s" \
                        %(description, detail)
                return True



### TEST ROUTINE ###

if __name__ == '__main__':
        a = Pyrx()
        a.createSession(serverAddress="localhost", loginName="Amos", \
        password="test")
        a.isConnected()
        symkey = a.generateKey("AES_128","asimplepassword")
        asymkey = a.generateKey("RSA_V1_5")
        a.addKey("AES_128","myaeskey","encryption",symkey[0])
```

```
a.addKey("AES_128","myaeskey","decryption",symkey[0])
a.addKey("RSA_V1_5","myrsakey","decryption",asymkey[0])
a.addKey("RSA_V1_5","myrsakey","encryption",asymkey[1])
a.bindNS("default","http://www.sample.com/tips/#2005ns")
a.bindNS("test","http://www.sample.com/tips/#2004ns")
a.unbindNS("default")
a.unbindAllNS()
a.encrypt("/spec/griffin","AES_128","myaeskey")
a.doXUpdate("""<xupdate:append select="/spec/griffin"><examplenode/>
</xupdate:append>""")
a.decrypt("/spec/griffin")
a.closeSession()
a.isConnected()
```

# Bibliography


[1] Alex Martelli, Python in a Nutshell, Paula Ferguson and Laura Lewin editors, O'Reilly, 2003.

[2] MySQL AB, MySQL Reference, http://www.mysql.com, last visited October 21., 2005..

[3] Herbert Schildt, Java 2 The Complete Reference Fifth Edition, Osborne, 2002.

[4] Don Chamberlin, Denise Draper et al., XQuery from the Experts A Guide to theW3C XML Query Language, Howard Katz, aug 2003.

[5] U.S. Bureau of Industry and Security, Department of commerce, Commercial Encryption Export Controls, http://www.bis.doc.gov/encryption/default.htm, last visited October 21., 2005..

[6] Wikipedia.org, Criptography, http://www.wikipedia.org, last visited October 21., 2005..

[7] Wikipedia.org, AES, http://www.wikipedia.org/wiki/AES, last visited October 21., 2005..

[8] Wikipedia.org, DES, http://www.wikipedia.org/wiki/DES, last visited October 21., 2005..

[9] Wikipedia.org, RSA, http://www.wikipedia.org/wiki/RSA, last visited October 21., 2005..

[10] Wikipedia.org, ACID, http://en.wikipedia.org/wiki/ACID, last visited October 21., 2005..

[11] W3C Group, XML Path Language (XPath) Version 1.0 W3C Recommendation, http://www.w3.org/TR/xpath, nov. 1999, last visited October 21., 2005..

[12] Graeme Malcolm, Programmare Microsoft SQL Server 2000 con XML, Seconda Edizione, Mondadori Informatica 2002.

[13] Hector Garcia-Molina et al., Database Systems, The Complete Book, Prentice Hall Pearson Education International, Alan R.Apt, 2002.

[14] XML: DB Initiative, XML:DB Initiative for XML Databases, http://xmldb-org.sourceforge.net, last visited October 21., 2005..

[15] XML: DB Initiative, XUpdate - XML Update Language, http://xmldb-org.sourceforge.net/xupdate/, last visited October 21., 2005..

[16] Elliotte Rusty Harold, W. Scott Means, XML In a Nutshell, O'Reilly, 2002.

[17] W3C Group, XML Pointer Language (XPointer) Working Draft, http://www.w3.org/TR/xptr, aug. 2002, last visited October 21., 2005..

[18] PostgreSQL Global Development Group, PostgreSQL reference, http://www.postgresql.org, last visited October 21., 2005..

[19] Donald E. Eastlake III and Kitty Niles, Secure XML,The New Syntax for Signatures and Encryption, Addison-Wesley, jul 2002.

[20] Akmal B. Chaudhri et al., XML Data Management, Native XML and XML-Enabled Database Systems, Addison-Wesley, mar 2003.

[21] JDOM library, Jason Hunter and Brett McLaughlin, http://www.jdom.org, last visited October 21., 2005..

[22] Jaxen Universal Java XPath Engine, Codehaus, http://www.jaxen.org, last visited October 21., 2005..

[23] A Java XML Update engine, http://klomp.org/jaxup/, last visited October 21., 2005..

[24] Sun Microsystems, Java Cryptography Architecture, http://java.sun.com/j2se/1.5.0/docs/guide/security, last visited October 21., 2005.

[25] The Bouncy Castle Crypto APIs, http://www.bouncycastle.org/, last visited October 21., 2005..

[26] Sun Microsystems, Java Cryptography Extension, http://java.sun.com/products/jce, last visited October 21., 2005..

[27] Archives et législation, Délais et supports de conservation, http://www.archives.ch/docs/legislation.pdf, last visited October 21., 2005..

[28] JArgs command line option parsing suite for Java, http://jargs.sourceforge.net/, last visited October 21., 2005..

[29] Berth Bos, What is a good standard?, An essay on W3C's design principles, http://www.w3.org/People/Bos/DesignGuide/toc.html, last visited October 21., 2005..

[30] Frank Naudé Oracle XML FAQ, http://www.orafaq.com/faqxml.htm, last visited October 21., 2005..

[31] Paul Dubois, Using XML with MySQL, http://www.kitebird.com/articles/mysql-xml.html, last visited October 21., 2005..

[32] Carnegie Mellon University, Software Engineering Institute, Two Tier Software Architectures, http://www.sei.cmu.edu/str/descriptions/twotier.html, last visited October 21., 2005..

[33] DOM4J, The flexible XML framework for Java, www.dom4j.org, last visited October 21., 2005

[34] Ross Lee Graham, Locks, http://www.ida.liu.se/~TDDB38/2002/LectureOH/Le12.htm.

[35] Ivan Baldassi, XPath tutorial, www.ilgiovine.com.

[36] W3C Group, Document Object Model, http://www.w3.org/DOM/, last visited October 21., 2005.

[37] W3C Group, SOAP Specification, http://www.w3.org/TR/soap/, last visited October 21., 2005.

[38] W3C Group, XML Schema, http://www.w3.org/XML/Schema, last visited October 21., 2005.

[39] W3C Group, XPath 2.0 Working Draft, http://www.w3.org/TR/2005/WD-xpath20-20050915/, last visited October 21., 2005.

[40] Eclipse Foundation, SWT: The Standard Widget Toolkit, http://www.eclipse.org/swt/, last visited October 21., 2005.

[41] Netbeans.org, Tuning JVM switches for performance, http://performance.netbeans.org/howto/jvmswitches/ last visited October 21., 2005.

[42] UserLand Software, XML-RPC, http://www.xmlrpc.com/, last visited October 21., 2005.

[43] Sun Microsystems, JavaTM HotSpot VM Options, http://java.sun.com/docs/hotspot/VMOptions.html, last visited October 21., 2005.

[44] W3C School, XML Schema Tutorial, http://www.w3schools.com/schema/default.asp, last visited October 21., 2005.

[45] W3C Group, Canonical XML Version 1.0, W3C Recommendation 15 March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315, last visited October 21., 2005.

[46] Andreas Meier, Introduction pratique aux base de données relationelles, Springer, 2002.

[47] Sun Microsystems, Java 2 Platform 1.5.0 documentation, Properties, http://java.sun.com/j2se/1.5.0/docs/api/java/util/Properties.html, last visited October 21., 2005.