

Intentionally left blank

Summary

1 Introduction.....	12
1.1 The MediCoordination Architecture	12
1.1.1 Architecture overview	13
1.2 Buts of the current diploma work.....	13
2 The Storage Manager	14
2.1 Description	14
2.2 Architecture overview.....	14
2.3 Requirements.....	15
3 Security analyze.....	16
3.1 The security problems.....	16
3.1.1 The protection of files on physical supports.....	16
3.1.2 Use of file encryption.....	16
3.1.2.1 Benefits.....	16
3.1.2.2 Inconvenient.....	17
3.1.3 Credentials hacking.....	19
3.1.4 The double identification.....	19
3.1.4.1 Benefits.....	20
3.1.4.2 Inconvenient.....	20
3.2 Technologies	20
3.2.1 Encryption problem	20
3.2.1.1 Technology description	20
3.2.1.2 Impact in performances	21
3.2.1.3 Impact in costs.....	21
3.2.2 Credentials hacking.....	21
3.2.2.1 Technology description	21
3.2.2.2 Impact in performances	22

3.2.2.3 Impact in costs.....	22
3.3 Conclusion	22
4 Storage technologies analyze	23
4.1 Requirements.....	23
4.2 Proposed solutions.....	24
4.2.1 Jackrabbit repository embedded in a Web Service	24
4.2.1.1 Solution schema	25
4.2.1.2 Description	25
4.2.1.3 The Apache Jackrabbit API	26
4.2.1.4 Apache Jackrabbit architecture.....	27
4.2.1.4.1 Workspace	28
4.2.1.4.2 Session	28
4.2.1.4.3 Node	28
4.2.1.4.4 Performances and limitations	29
4.2.1.4.5 Security.....	30
4.2.1.5 Encryption/Decryption APIs	31
4.2.1.6 Encryption of medical data on a grid.....	31
4.2.1.6.1 Benefits.....	32
4.2.1.6.2 Disadvantages.....	32
4.2.1.7 Access to the web service methods	32
4.2.1.8 Conclusion	32
4.2.2. Web service with standard file system.....	32
4.2.2.1 Schema	33
4.2.2.2 Description	33
4.2.2.3 Benefits.....	33
4.2.2.4 Disadvantages.....	34
4.2.2.5 Conclusion	35
4.2.3 Web service with external data base connection.....	35
4.2.3.1 Schema	35
4.2.3.2 Description	35

4.2.3.3 Benefits.....	36
4.2.3.4 Disadvantages.....	36
4.2.3.5 Conclusion	36
4.3 The chosen solution	36
4.3.1 Points to take care	37
4.3.1.1 Limitations of the solution.....	37
4.4 Conclusion	37
5 The Storage implementation	38
5.1 Tools	38
5.1.1 NetBeans.....	38
5.1.2 Sun GlassFish version 2.1.....	38
5.1.3 Apache Jackrabbit 1.5.6	39
5.1.4 MySQL version 5.1	39
5.2 Installation and setup.....	39
5.2.1 Requirements.....	39
5.2.2 NetBeans 6.7	40
5.2.3 Sun GlassFish Enterprise Server 2.1.....	40
5.2.3.1 Install all Jackrabbit libraries	40
5.2.3.2 Installation of the MySQL driver.....	40
5.2.3.3 Install the java Cryptography Extension 1.2.2 (JCE) Unlimited Strength.....	41
5.2.3.4 First GlassFish server start.....	41
5.2.3.5 GlassFish configuration	41
5.2.3.5.1 Login into the GlassFish administration interface.....	42
5.2.3.5.2 Creation of the GlassFish user	43
5.2.3.5.3 Command line “Add new user”	45
5.2.4 MySQL 5.1 Community Server	46
5.2.4.1 Installation.....	46
5.2.4.2 Setup.....	46

5.2.4.3 StorageManager configuration changes	46
5.3 The solution implemented	47
5.3.1 Schema.....	47
5.3.1.1 The final architecture vs. projected architecture	48
5.3.1.2 Description	48
5.3.2 The security implementation end to end data protection	49
5.3.2.1 SSL secured connections	49
5.3.2.1.1 Step 1 – Define a secure Service	50
5.3.2.1.2 Step 2 - Define what resources may use SSL connection	51
5.3.2.1.3 Step 3 - Project deployment.....	52
5.3.2.2 File encryption	52
5.3.2.2.1 UML schemas of the cryptography packages.....	53
5.3.2.2.2 The cryptography package classes description	54
5.3.2.3 Web Methods access restrictions	55
5.3.2.3.1 The Java™ Security Manager class	56
5.3.2.3.2 The adopted solution	56
5.3.2.3.3 Conclusion	58
5.3.3 The storage interface implementation	58
5.3.3.1 Definition	58
5.3.3.2 Schematic view	59
5.3.3.2 StorageManager Project UML Schema (collapsed version)	59
5.3.3.2.1 org.medicoordination.db package	60
5.3.3.2.2 org.medicoordination.repository package.....	60
5.3.3.2.3 org.medicoordination.managers package.....	61
5.4 Project components	64
5.4.1 Commons library.....	64
5.4.1.1 The UML schema	64
5.4.1.2 SHA1Digest class.....	64
5.4.1.3 DocumentFormat enumeration	65
5.4.1.3.1 Possible solution to solve the DocumentFormat problem.....	65

5.4.1.4 XmlClassSerializer class	65
5.4.1.5 FileStructure class.....	65
5.4.1.6 DynamicClassLoader class	66
5.4.1.7 DebugMode class	66
5.4.1.8 ErrorMessageBean	66
5.4.2 SecurityManager Web Service.....	66
5.4.2.1 UML schema	66
5.4.2.2 The org.medicoordination.service package	67
5.4.2.2.1 The Authentication Web Service	67
5.4.2.2.2 The ResourcesAuthorization Web Service	67
5.4.2.3 The org.medicoordination.authorization package	68
5.4.2.3.1 The IAuthorization interface	68
5.4.2.3.2 The DefaultAuthorization class	69
5.4.2.4 The org.medicoordination.authentication package	69
5.4.2.4.1 Strategy pattern applied to the Authentication Web Service (diagram) .	69
5.4.2.4.2 The TestAuthentication class.....	70
5.4.2.5 The org.medicoordination.security and db packages	70
5.4.2.5.1 The DefaultSecurityToken class.....	70
5.4.3 TranslationManager Web Service.....	70
5.4.3.1 UML schema	71
5.4.4 StorageManager Web Service	71
5.4.4.1 Description of the functionalities	72
5.4.4.2 MySQLRepository	72
5.4.4.2.1 UML Diagram.....	73
5.4.5 MediCoordination front-end web application.....	74
5.4.5.1 Sitemap Diagram	74
5.4.5.2 Description of the functionalities	75
5.5 Implementation problems and solutions.....	75
5.5.1 Web Services.....	75
5.5.1.1 Security and authentication	75

5.5.1.1.1 Context	75
5.5.1.1.2 Problem encountered.....	75
5.5.1.1.3 Solution.....	75
5.5.1.2 SSL Implementation.....	79
5.5.1.2.1 Context	80
5.5.1.2.2 Problem encountered.....	80
5.5.1.2.3 Causes and solutions	81
5.5.2 Apache Jackrabbit.....	82
5.5.2.1 Configuration.....	82
5.5.2.1.1 Solution.....	83
5.5.2.2 Sessions and Locks (Concurrent accesses to the repository)	83
5.5.2.2.1 Solution.....	83
5.5.2.3 Node Versioning & Locking	84
5.5.2.3.1 Configuration of the repository.....	84
5.5.2.3.2 How to make a node versionable mix:versionable	85
5.5.2.3.3 mix:lockable.....	85
5.6 Improvements	85
5.6.1 Dynamic loading integration.....	85
5.6.2 Jackrabbit session management.....	86
6 Storage Manager Compatibility with IHE recommendations.....	87
6. 1 Storage compatibility	87
6. 2 File exchange (XDS) compatibility	87
6.3 Service security compatibility	87
6.3.1 User authentication (EUA) compatibility.....	87
7 Conclusion	88
8 Bibliography.....	89

List of figures

Figure 1 Architecture overview	13
Figure 2 The Storage Manager architecture overview	14
Figure 3 Graphical version of the table 1 data	18
Figure 4 Storage Management Schema using Apache Jackrabbit as a standard repository...	25
Figure 5 Apache Jackrabbit architecture model	27
Figure 6 Default repository storage structure.....	30
Figure 7 Proposed repository storage structure Default	30
Figure 8 Access procedure when shares of the decryption keys are stored on different key servers	31
Figure 9 Storage Management using standard file system	33
Figure 10 Storage Management using Apache Jackrabbit piloting an external database	35
Figure 11 GlassFish administration console	42
Figure 12 Realms and users.....	43
Figure 13 Users in GlassFish	44
Figure 14 New file realm user	45
Figure 15 Project final architecture	48
Figure 16 Projected solution architecture	48
Figure 17 SSL connection and visible signs.....	49
Figure 18 StorageManager Web Service attributes	50
Figure 19 StorageManager Security Mechanism definition.....	51
Figure 20 Add new Security Constraint	52
Figure 21 the UML diagram of the cryptography package in StorageManager Web Service .	53
Figure 22 the UML diagram of cryptography package in Commons.....	54
Figure 23 the security token UML schema	56
Figure 24 Web Methods access restriction system.....	57
Figure 25 a Web Method protection sample code	58
Figure 26 StorageManager internal representation	59
Figure 27 StorageManager UML diagram	60
Figure 28 the MySQLRepository class (UML diagram)	61
Figure 29 the strategy pattern UML diagram.....	62
Figure 30 the dynamic loading of the storage manager class.....	62
Figure 31 the core of the StorageManager Web Service (Strategy pattern diagram)	63
Figure 32 the commons library UML diagram.....	64
Figure 33 SecurityManager UML class	66
Figure 34 the methods provided by the ResourcesAuthorisation Web Service	68
Figure 35 The ResourcesAuthorisation strategy pattern refactoring UML diagram.....	69
Figure 36 Strategy pattern refactoring of the Authentication Web Service (UML diagram) ..	69
Figure 37 The TranslationManager project (UML diagram).....	71
Figure 38 The functionalities provided by the StorageManager Web Service.....	72
Figure 39 UML diagram of the dependencies of the MySQLRepository class	73

Figure 40 Sitemap of the MediCoordination front-end	74
Figure 41 Server authentication mechanism setup	76
Figure 42 Servlet login configuration	77
Figure 43 web.xml Servlets authentication configuration	78
Figure 44 Security Role Mappings	79
Figure 45 Sample code to display the connected principal.....	79
Figure 46 Web Service client creation.....	80
Figure 47 Exception thrown by NetBeans	80
Figure 48 Error report from NetBeans	81
Figure 49 Web Site Certificate viewer window	81
Figure 50 Logging in to Jackrabbit	82
Figure 51 Versioning entry in the repository.xml configuration file	84

List of tables

Table 1 between the most popular encrypting algorithms.....	17
Table 2 common file system limitations.....	34

Intentionally left blank

Intentionally left blank

1 Introduction

Nowadays, medical content exchange between medical actors is done primarily by paper. But the paper content exchange may generate extra costs and sometimes can
5 have important consequences on patient health.

The digitalization of medical content is seen as a priority for public authorities and by the same a chance to improve the quality of the Swiss health system.

In Switzerland, because of the particular political system, each State may have its own standards to electronic medical content storage and exchange. This diversity is more a
10 problem than a solution. Indeed, most of these systems are totally or partially incompatible with each other.

“The goal of the MediCoordination project is to propose concrete solutions to the interoperability of electronic records of patients between hospitals and the health business actors”¹

15 The MediCoordination project is a priority of the recent eHealth strategy of the Swiss Confederation and by same a chance to the HES-SO to be present in a multidisciplinary, very interesting and important field of action.

1.1 The MediCoordination Architecture

As seen in the Figure 1, the MediCoordination is totally service oriented. This strategy is
20 totally wanted by the working group. Indeed, the MediCoordination project does not want to replace existing electronic health document exchange standards in Swiss states, but rather, wants to be a federative layer that will allow all existing systems to communicate together and exchange electronic health documents together.

Because of the existence of 26 States in Switzerland and probably as many health
25 systems, data will not be centralized but rather distributed over many data repositories and Meta information servers.

The particularity of Swiss health system needs a particular architecture. Therefore, the MediCoordination project is a scalable and distributed system without a centralized service collecting patient personal data.

30 This architecture is also a consequence of the Swiss laws protecting citizens’ personal data and private life. Indeed, Swiss Confederation is very protective in matters that concern the personal data confidentiality and puts real efforts to avoid it diffusion to non authorized persons.

¹ <http://www.medicoordination.ch/>

1.1.1 Architecture overview

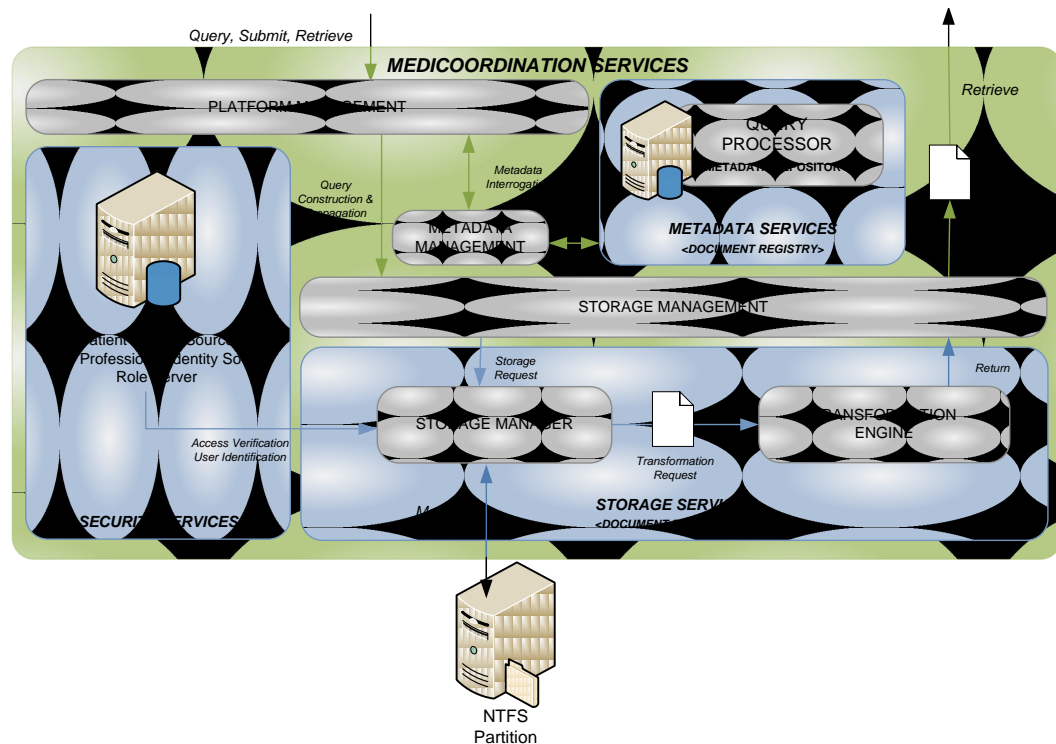


Figure 1 Architecture overview

The present project consists in a set of Web Services working together in a distributed manner. Succinctly the project consists in three parts:

1. The storage services, a set of services that are responsible to store and retrieve in a reliable manner the health documents.
2. The security services, a set of services that are responsible to authenticate and authorize all the actors and users of the system to interact with it.
3. The Metadata services, a set of services that are responsible to store information about patient documents in order to facilitate documents no matter where they are.

1.2 Buts of the current diploma work

The present diploma work is to analyze the context of the MediCoordination project, to understand the objectives and requirements related with it. This analyze will allow to better understand the requirements of the global solution in order to implement a prototype of the Storage Manager that fulfill these requirements.

The goal is to develop a prototype that will proof the feasibility of the concept, it means developing a prototype for electronic medical content storing respecting all the requirements formulated by the MediCoordination working team.

2 The Storage Manager

55 The Storage Manager is a master piece of the MediCoordination project and belongs to the Storage Services, which belongs itself to the MediCoordination Services layer. Figure 1 gives an overview of the MediCoordination services layer and the place taken by this manager.

2.1 Description

60 The Storage Manager consists in a secured Web Service that is connected to other secured services and has to respond to requests made by the storage management layer, Figure 1 above.

The use of standard technologies during implementation was a keyword, as well as data security and confidentiality. Then, during the entire project a special care has been
65 taken in order to fulfill these requirements.

The storage manager must store and retrieve electronic medical content from a data repository ensuring data integrity, security and confidentiality. By the way it must be totally integrated with the other service layers using, if possible health standards or recommendations made by the IHE consortium.

70 2.2 Architecture overview

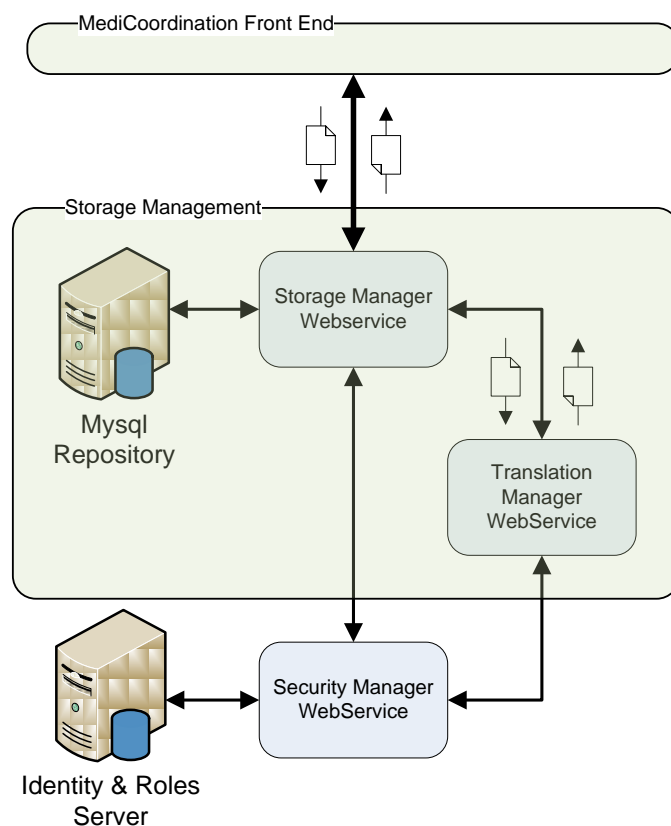


Figure 2 The Storage Manager architecture overview

75 The above schema represents the final implemented solution. As we can see, some services from the security services layer have been partially implemented for tests propose.

The implementation of these elements brought extra work, but without them, the diploma work would limit to the implementation of a simple repository and no tests could be made.

80 Indeed, the security in the Storage Manager is intrinsically linked with the security services layer and therefore both have to be implemented in order to make real tests possible.

2.3 Requirements

The Storage Manager project had to respect a certain number of requirements among them the most important are:

- 85
1. The storage must be totally integrable in the MediCoordination project.
 2. The security and data protection must be ensured at any moment.
 3. The storage reliability must be ensured as well as data integrity, confidentiality and security.
 4. The storage must, if such technologies exist, implement standard technologies.

90

 5. It must be totally hardware independent
 6. It must keep a track of all changes in a document (versioning)
 7. It must be able to store any kind of documents
 8. It must integrate access security to the web service methods
 9. Be able to keep a very important quantity of data, and backup it efficiently.

95

A last requirement, but this time not directly related with the Storage Manager itself, all the documentation had to be written in English in order to be readable by all the project participants.

3 Security analyze

100 In this chapter we will analyze the security in the MediCoordination project. The chapter is intended to bring a new view point to the security problem in the MediCoordination project and more precisely the security problems related to the storage manager. The current chapter is based on the deliverable *D3.1A: Interoperability Architecture* and will propose security improvements and implementable solutions.

105 We will not redo the work done in the D3.1A but rather improve it. Therefore, we will only mention, in the chapter, the new security lacks found and how to secure them.

3.1 The security problems

Since the project MediCoordination handles very sensitive data (patient medical information) a particular care must be taken in order to avoid any interception of those
110 files by non authorized persons. Starting with this Lemma, all the data transfers between clients and servers must be protected to avoid any kind of data hacking. By clients we mean any medical personnel with access to patient data

Actually the D3.1A document seems to respond quite well to this requirement, all transactions are carried by SSL ports², but some security points must be improved.

115 3.1.1 The protection of files on physical supports

The first improvement that must be performed consists in the encryption of the files stored on physical supports. In fact, non encrypted files could represent a potential security breach that would allow anybody having server access to copy, delete or modify any patient data stored inside the server. Such case is not allowed in such application
120 and must therefore be corrected.

Actually *D3.1A : Interoperability Architecture* do not take into account the encryption of patient files.

3.1.2 Use of file encryption

To avoid such security lack, we encourage the use of an encryption technology on the
125 files. Implementing such functionality will grant to patient that all files will be unreadable, by third person, from the medical practice to the physical storage.

3.1.2.1 Benefits

The main benefit of file encryption resides in the impossibility to see clearly what is stored inside the file and then consequently the impossibility to someone without
130 credentials to read, modify, the content of a file.

² D3.1A : Interoperability Architecture (§2.4.5)

File encryption will not solve the sabotage problem. Therefore, this case can be reasonably avoided by laying the storage system in a well secured and configured network with regular backups.

3.1.2.2 Inconvenient

- 135 File encryption will necessitate more processing time than simple file storage. Therefore the choice of the encryption/decryption algorithm may be critical to avoid server slowdown.

- 140 Because of the time needed to encrypt files, an investment on new hardware may be necessary. To help to avoid an important slowdown of the server when encrypting files, a separated thread could be in charge of the encryption and storage of the files. Those files would be temporarily stored in a queue and would only be encrypted and stored during processor idle phases.

- 145 The web page *Speed Comparison of Popular Crypto Algorithms*³ hosted on cryptopp.com can give an overview about the time needed by a computer to encrypt and decrypt a content. This table is particularly interesting because all tests have been made in same conditions and therefore it gives a very accurate comparison between algorithms.

Table 1 between the most popular encrypting algorithms⁴

Operation (over a small block of random data)	Milliseconds/Operation	Megacycles/Operation
RSA 1024 Encryption	0.08	0.14
RSA 1024 Decryption	1.46	2.68
LUC 1024 Encryption	0.08	0.14
LUC 1024 Decryption	2.49	4.55
DLIES 1024 Encryption	0.85	1.56
DLIES 1024 Encryption with precomputation	1.49	2.72
DLIES 1024 Decryption	1.18	2.17
LUCELG 512 Encryption	0.58	1.05
LUCELG 512 Encryption with precomputation	0.58	1.05
LUCELG 512 Decryption	0.65	1.18
RSA 2048 Encryption	0.16	0.29

³ <http://www.cryptopp.com/benchmarks.html>

⁴ <http://www.cryptopp.com/benchmarks.html>

RSA 2048 Decryption	6.08	11.12
LUC 2048 Encryption	0.18	0.33
LUC 2048 Decryption	9.89	18.1
DLIES 2048 Encryption	4.11	7.52
DLIES 2048 Encryption with precomputation	4.54	8.3
DLIES 2048 Decryption	3.86	7.07
LUCELG 1024 Encryption	1.89	3.45
LUCELG 1024 Encryption with precomputation	1.88	3.45
LUCELG 1024 Decryption	1.73	3.17

150 The results are however too high. This fact is due because of the length of the encryption key. For our purpose à 128 bits key will be enough. With such key length the operation time need will decrease.

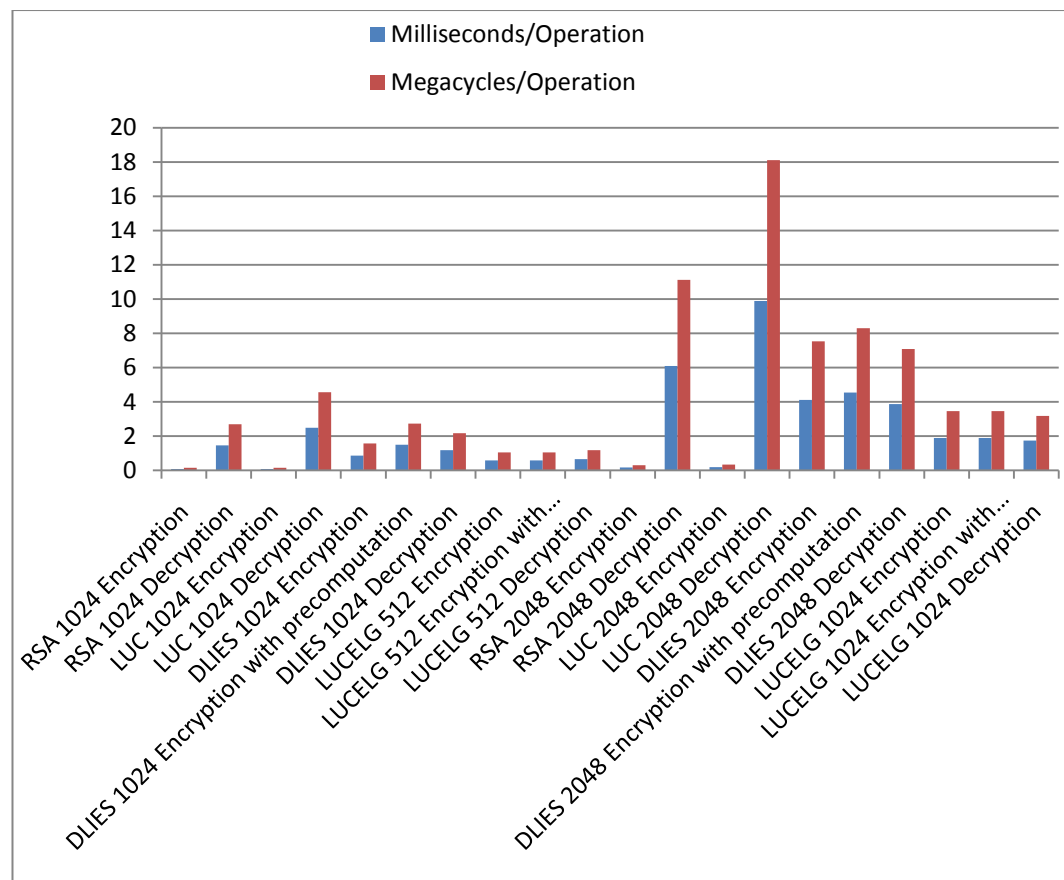


Figure 3 Graphical version of the table 1 data

155 If we look at the table above, for an encryption with a RSA 1024 bits, the operation will
take 0.08 milliseconds and for the same data chunk with RSA 2048 (key twice bigger) the
time to perform the same operation will be twice bigger. It suggests that the time to
encrypt the message is linearly dependant to the length of the encrypting key, thus, for
a 128 bits key, the operation will take only $\frac{1}{8}$ of the needed to encrypt the same data
chunk with a 1024 bits key. Contrariwise the decryption didn't follow the same rule.

160 3.1.3 Credentials hacking

The second security improvement will allow a better control over the patient files
access. Actually, as described in the *D3.1A : Interoperability Architecture* document⁵, the
only "firewall" that protects patients data from unwanted accesses are the file rights
stored in the roles server. Those accesses are granted by the patient, but once those
165 rights are granted a medical worker will be able to consult, copy or modify those files
every time he wants.

The visible problem is that, if someone inside the role is malicious, he will be able to
copy and sell information to third parties. The other problem more plausible is that
someone can hack the authentication data of a health worker, thus acquiring all
170 credentials to access patient files followed by this health worker. What happens if
someone can hack the authentication of a hospital? All the patients that gave their
consent to this hospital role will risk having their confidential data exposed to non
authorized third party.

3.1.4 The double identification

175 A double identification can solve the problem exposed above. It consists in the use of
both doctor identification and patient identification to access patient data. This solution
denies the possibility that third persons, belonging to the same role, can access, copy
and transmit the patient data to other persons or insurance companies. Every time a
health worker needs to consult the patient data he will need to provide the patient
180 identification and his own identification to access data.

This identification is mostly given by the patient through his patient card read by a card
reader or through his card number (the new thirteen digits AVS number) and a pin
number that acts as a password.

185 In the case of the patient didn't have the card or pin it will be possible to ask for
documents access to a global supervisor, but every request will be verified carefully and
the pin have to be changed as soon as the .

⁵ D3.1A : Interoperability Architecture (§3.4)

3.1.4.1 Benefits

Because the presence of the patient is required in order to read documents from the repository, confidential data will, normally, remain private to the medical worker after the departure of the patient, except the files loaded by the health worker.

3.1.4.2 Inconvenient

The biggest inconvenient is the costs that such a solution will generate. Every doctor will need to buy at least one card reader in order to access patient data. But as discussed in a previous chapter, this solution can work as well with only the AVS number that acts as a username plus a pin that acts as the password.

3.2 Technologies

We have discussed above about two security breaches and found solutions that solve the problem. This chapter will bring more detailed information about the existing solutions and if more than one solution exists, we will compare them.

3.2.1 Encryption problem

We saw above that the simple use of an encryption algorithm before storing the files will solve the problem. But we noted that encryption had a price in terms of server performance. We will propose some encryption solutions and try to propose solutions that will reduce the impact of files encryption on existing systems.

3.2.1.1 Technology description

Because the entire project turns around Java technologies, it will be possible to use the standard java cryptographic library, JCA and JCE, to encrypt / decrypt the files^{6,7}. This framework contains all classes needed to encrypt files using RSA, DES, DAS, etc. The package contains also all the classes needed to sign and verify a signature.

The second solution, easier to the programmers, consist using Operating System file encryption. By example Microsoft® Windows® Server 2003 provides an encryption service that wills encrypt all the files for the opened session. The Service is called EFS and uses, by default, a proprietary encrypting algorithm the DESX. But a stronger encryption algorithm can be used instead. One can change de DESX algorithm by the more secure algorithm 3DES^{8,9}.

⁶ <http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html>

⁷ <http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html>

⁸ <http://www.windowsecurity.com/articles/Implementing-EFS-Windows-Server-2003-Domain.html>

⁹ [http://technet.microsoft.com/fr-fr/library/cc700811\(en-us\).aspx](http://technet.microsoft.com/fr-fr/library/cc700811(en-us).aspx)

The problem with this solution is that the files remain visible and readable for someone who can access physically the server. EFS encrypts files, but for the windows session owner, all its files are visible as if they aren't encrypted.

220 A third solution consists using Apache Jackrabbit¹⁰ to store files. Jackrabbit is the complete implementation of the Java™ Content Repository well known as JCR. Jackrabbit provides a complete API to create a repository to store files. The repository is totally operating system independent and data can be stored in a wide range of storage supports such in a WebDAV directory, in a database or to a local directory on the hard disk. Jackrabbit allows data versioning.

225 *3.2.1.2 Impact in performances*

As already discussed above, encryption is an operation that needs much processor time than simple storage. But using SSL communication channels for data exchange without protecting it at the other end seems to be a totally non-sense. Encryption is needed in order to ensure a high degree of data confidentiality.

230 To avoid a too high overload of the system it will be possible to encrypt the files only when the processor is in the idle state or under a certain threshold, 30% by example. That solution avoids hard investments in new servers.

3.2.1.3 Impact in costs

There are no special extra costs implementing the encryption file system.

235 *3.2.2 Credentials hacking*

We will try to propose and discuss, in this chapter, solutions that will solve the credentials security breach.

3.2.2.1 Technology description

240 The first technology described that can be employed to solve this problem consists in the use of a card reader coupled with the health insurance cards. The system will only respond to queries if the medical worker and the patient are correctly authenticated with their respective health cards.

245 To perform that task, the medical practice or hospital, will need a computer with the card reader connected to it. To recover the patient medical files, both cards need to be presented to the system. To better understand how the system works, we can compare our system to a nuclear weapon system. To be able to launch a nuclear head, two keys are necessary. Without both keys the system will not respond to the requests. In our system the keys are replaced by the health insurance card and the launch of a nuclear head correspond to the read request sent by the medical worker.

¹⁰ <http://jackrabbit.apache.org/>

250 The second solution that we propose to solve the credentials problem is pretty similar to the one discussed in the previous paragraph excepted that we only need the thirteen digits AVS number that is written on the card and the pin of the card. The use of a pin instead of the card will reduce costs without reducing the security of the mechanism.

255 The patient has to provide his patient id (the thirteen digits AVS number), or any other information that will allow identifying him and the pin code in order to authenticate him as a user of the system. We can image that new authentication methods will access a Web Service that will provide the classic credentials as output. With that output the actual system don't need to be changed. Both methods can be applied to solve the problem; the last has the advantage to be cost free and easier to setup in hospitals or
260 medical practices because there is no need to install hardware.

3.2.2.2 Impact in performances

There is no special impact in performances.

3.2.2.3 Impact in costs

265 The extra costs depend on the solution chosen. The solution that consists in using the card reader will generated some hardware extra costs. At least a reader is needed per medical practice. Actually the price of a smart card reader turns between 7\$¹¹ and 100\$.

3.3 Conclusion

The MediCoordination project handles very sensitive data. To avoid those data to be hacked, some changes must be performed.

270 The most critical security breach, and at the same time the one that is really interesting to the implementation of the data manager module, is the encryption of the patient files. Without this patch, data could be accessed by hackers or by anyone that have an access to the storage server.

275 The second improvement proposition is less critical but will bring a plus to the final system rending it less vulnerable to data steel and giving a feeling of security to the end users.

¹¹ <http://www.teobyxiring.com/index.php?gclid=CLCjwMyS1poCFUgTzAodAVgOsg>

4 Storage technologies analyze

This document is intended to present some storage technologies or products from commercial or open source domains compatible with IHE¹² (Integrate the Healthcare Enterprise) recommendations. The goal is to have, at the end, a better overview of those technologies and then to be able to choose the best one to implement the Storage Module of the MediCoordination project.

The different solutions need to fulfil a certain number of criteria that have been fixed in the *D3.1A: Interoperability Architecture* and have to respect the IHE recommendations about Cross-Enterprise Document Sharing (XDS).

4.1 Requirements

The storage module has to respect a certain number of requirements edited by the IHE workgroup in order to be compatible with other healthcare sharing interfaces implementing those requirements.

The MediCoordination requirements for the storage module are the following:

- Be totally hardware independent
- Keep a track of all changes in a document (versioning)
- Be able to store any kind of documents
- Be able to store document fragments in a consistent manner
- Integrate encryption over the files
- Integrate access security to the repository
- Integrate access security to the web service methods
- Consume low resources
- Be sufficiently easy to setup in a medical office and sufficiently robust to support a consequent number of accesses (store/request) in a bigger medical structure, like a hospital.
- Be able to keep a very important quantity of data, and backup it efficiently.

The mandatory functionalities of the module as described in the IHE recommendations are:

- Retrieve request¹³,
 - A file must be read from the repository/db and send back to whom requested it
 - The module needs to ensure that the requester have the authorization of the patient to consult the document
- Submission request¹⁴

¹² www.ihe.net

¹³ ihe_iti_tf_5-0_vol1_ft_2008_12_12 (§ 10.1.2.5 page 69)

- A medical worker can submit a new file to the patient folder
- The file must be stored in a reliable manner
- Update request (replacement, addendum, transformation, transformation-replacement)¹⁵
 - A medical worker can submit a new version of a file
 - Old files must be maintained, a version number must be setup
 - Old files can be accessed if needed
 - The last version of a file is the one that is proposed, except if the requester asks a precise version, to a retrieve request

320 The storage module must be developed as a web service. This choice has been deeply discussed in the *Deliverable D3.1A: Interoperability Architecture*. It responds to the need of interoperability between storage module and other modules composing the MediCoordination solution and the need to create a scalable and totally distributed solution.

325 4.2 Proposed solutions

In this chapter we will discuss about three solutions, all based on Apache Jackrabbit API. We have chosen the Apache Jackrabbit solution for many reasons:

1. Apache Jackrabbit allows storing data on a large number of different supports.
2. Apache Jackrabbit is regularly updated and maintained.
- 330 3. It's relatively easy to integrate into a web service and to setup.
4. Apache Jackrabbit is totally operating system independent.
5. Apache Jackrabbit allows file versioning in a transparent manner.
6. Apache Jackrabbit allows clustering to setup very large repositories.
7. Apache Jackrabbit is open source and easily modifiable.
- 335 8. Apache Jackrabbit implements a Java™ standard the Java™ content repository known as JCR

Apache Jackrabbit is a good compromise between commercial solutions that don't fit most of requirements and are very expensive and a built from scratch solution that will take too long to implement. With Apache Jackrabbit we can create a solution that will

340 fit all the requirements without having to care about the manner the files will be stored and managed.

4.2.1 Jackrabbit repository embedded in a Web Service

The schema below shows the possible architecture of the storage manager service and dependencies services useful to it to make document translation by example. The first

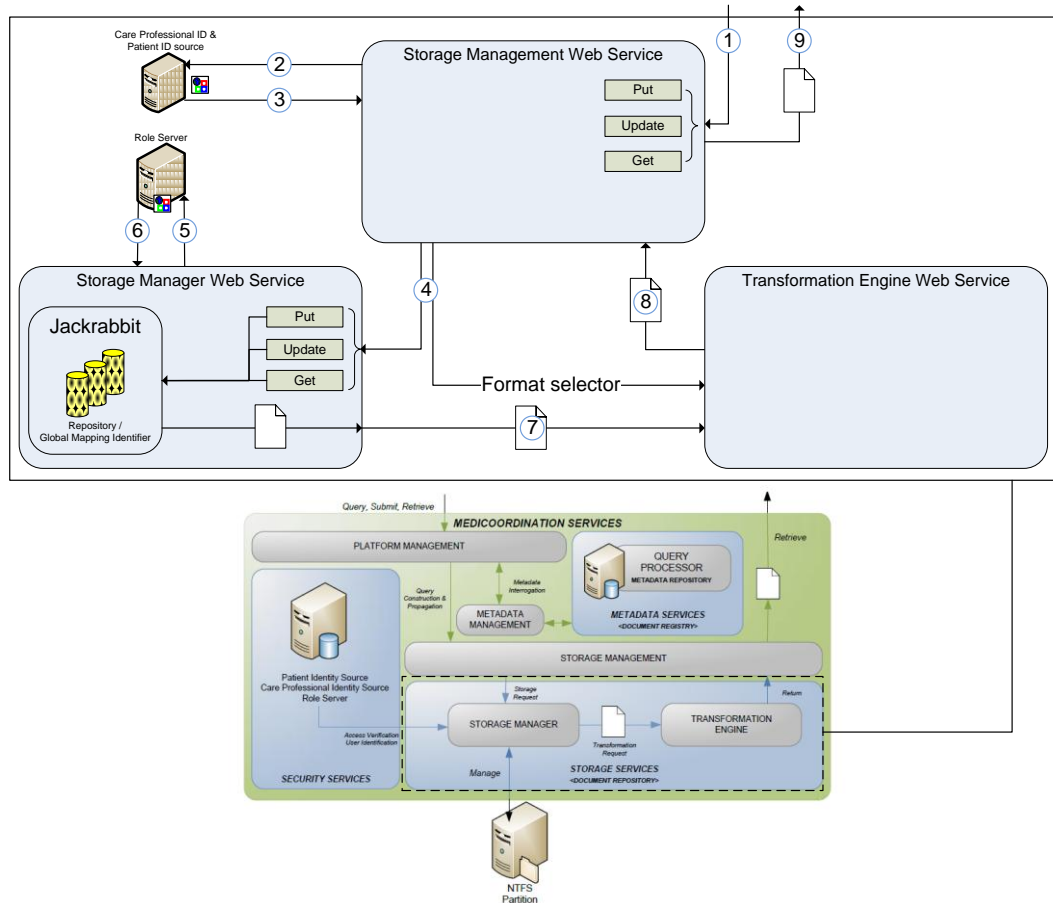
345 proposed solution uses Jackrabbit as a standard content repository. Files are stored

¹⁴ ihe_iti_tf_5-0_vol1_ft_2008_12_12 (§ 10.4.3 page 79)

¹⁵ ihe_iti_tf_5-0_vol1_ft_2008_12_12 (§ 10.4.10.2 page 87)

inside nodes. The resultant repository file structure looks like, if you are used to subversion, as a SVN repository.

4.2.1.1 Solution schema



350 **Figure 4 Storage Management Schema using Apache Jackrabbit as a standard repository**

4.2.1.2 Description

The first solution consists in a web service that will be installed locally in every medical office, health care centre or hospital that embeds a repository managed by the repository API Apache Jackrabbit.

355 We will explain the main steps during a client request.

- (1) The storage Management Web Service receives put/update/get request from a client
- (2) The Web Service controls that the session token provided by the user corresponds to a valid session
- 360 (3) If the session is invalid then the Web Service throws an *AccessDeniedException*
- (4) The request is relayed to the StorageManager Web Service
- (5) The service call the role server to know if the user can read/write/update the file

- (6) If the user is not authorized to do the action the service throws an *OperationNotAllowedException*
- 365 (7) If the user requested a file then it is transmitted to the TranslationManager Web Service to be translated.
- (8) The translated file is sent to the StorageManagement Web Service
- (9) The client receives the requested file translated into the selected document format.

370 This solution is particularly adapted to the actual problem, respects all the requirements and the main advantage is that it is built around an existing Repository API described by the Java™ Specification Request 170 and 283 (JSR 170, JSR 283) called Apache Jackrabbit. This API allows creating a repository that is operating system independent and multi support.

375 The fact that the API already exists implies a gain of time and money for the current project.

4.2.1.3 The Apache Jackrabbit API

The Apache Jackrabbit API implements the JSR 170 and JSR 283 that defines the recommendations to the creation of a Repository API system. Apache Jackrabbit can
380 create a repository on a wide range of operating systems and storage devices like hard disks, USB sticks, WebDav servers, data bases or network disks¹⁶.

The Apache Jackrabbit API allows versioning, backups and migration and allows storing any kind of content inside the repository.¹⁷

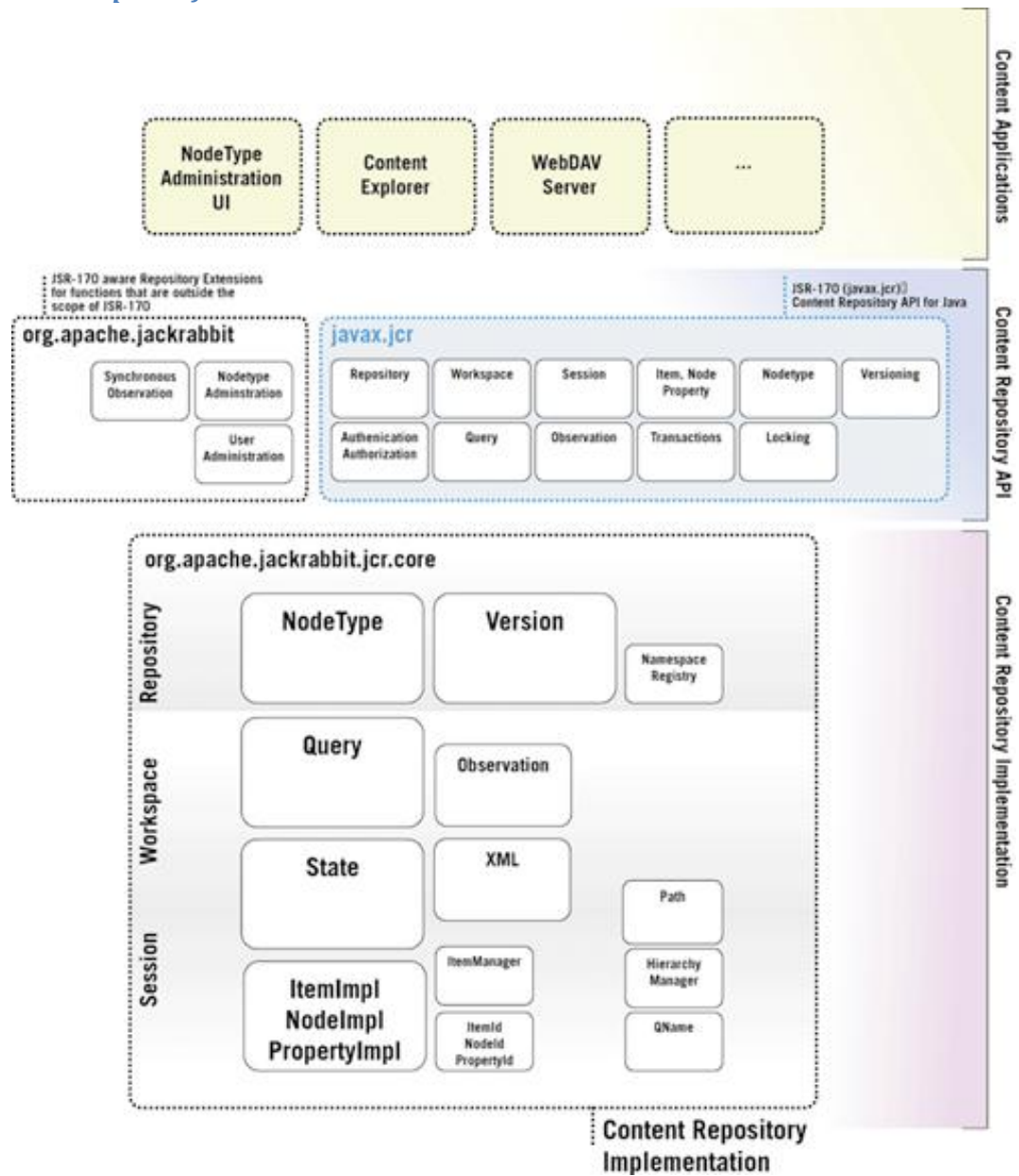
Nowadays it is used by a certain number of important companies creating content
385 management systems (CMS) or related products. One of the most famous is Alfresco¹⁸. Alfresco is an open source CMS, a Web content management (WCM), a Collaborative content management (CCM) at the same time using Jackrabbit as repository manager.

¹⁶ <http://jackrabbit.apache.org/frequently-asked-questions.html> (What is a Jackrabbit file system)

¹⁷ <http://wiki.apache.org/jackrabbit/BackupAndMigration>

¹⁸ <http://www.alfresco.com/>

4.2.1.4 Apache Jackrabbit architecture



390

Figure 5 Apache Jackrabbit architecture model¹⁹

The above model represents the Jackrabbit content repository implementation. Apache Jackrabbit works in terms of workspaces, sessions and nodes. The next chapters will bring more details about common objects of the Jackrabbit API.

¹⁹ <http://jackrabbit.apache.org/jackrabbit-architecture.html>

4.2.1.4.1 Workspace

395 The JCR workspace object represents a “view” of a repository workspace as seen through the authorization settings of its associated Session. A workspace belongs to a Session, and this session belongs to only one workspace. It is a one-to-one relation.

4.2.1.4.2 Session

400 The JCR session object provides authentication methods and read/write accesses to the content of a particular workspace. A repository can support concurrent sessions but a session is not thread safe. The Jackrabbit team advises to create a session for each thread trying to access the repository and to use the Node locks already implemented.

4.2.1.4.3 Node

405 A JCR node is an object that represents a node of the repository tree. A node is connected to a parent and can have zero to many children nodes and have zero or many properties attached to it. A JCR node can be of type **primary** or **mixin**. By default each newly created node has the **primary** node type. The second type is the **mixin** type. The **mixin** type can be set later and allows setting some special properties to a node. By example the creation date or to set a node versionable.

410 The primary node type is often used to define node structure while **mixin** usually specifies additional properties, *mix:versionable* to make the node versionable or *mix:lockable* to make possible to lock the node, or other child nodes.

Inside a node we have properties. Those properties contain the information we want to read or can act as data containers.

415 *A property definition (within a node type definition) contains the the following information:*

A property contains the information below:

- **Name** *The name of the property*
- **Required Type** *The required type of the property. Must be one of*
 - 420 ○ *STRING*
 - *BINARY*
 - *LONG*
 - *DOUBLE*
 - *BOOLEAN*
 - 425 ○ *DATE*
 - *PATH*
 - *NAME*
 - *REFERENCE*
 - *UNDEFINED* (the property can be of any type)
- 430 • **Value Constraints** *This field defines the value range that can be assigned to the property.*

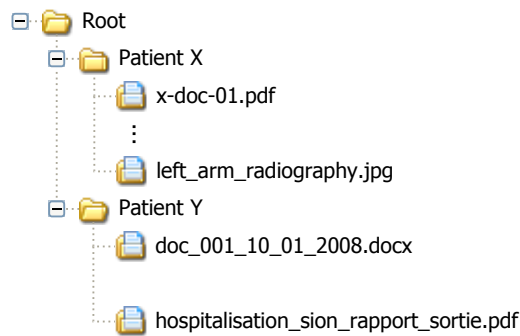
- **Default Value** The default value of the property in case it is auto-created.
- **Auto-create Status** Indicates that the property must be auto-created when its parent node is created. Only properties with a default value can be auto-created.
- **Mandatory Status** A mandatory property is one that must exist. If a node of a type that specifies a mandatory property is created then any attempt to save that node without adding the mandatory property will fail. It looks like the required field when creating an SQL Table with fields.
- **On-Parent-Version Status** The onParentVersion status of specifies what happens to this property if a new version of its parent node is created (i.e. a checked-in is done on it).
- **Protected Status** Indicates that the property cannot be modified. It means that we cannot add or remove children or properties and we cannot remove it from parent node through the JCR API.
- **Multiple Values Status** Whether this property can have multiple values, meaning that it stores an array of values, not just one. Note that this "multiple values" flag is special in that a given node type may have two property definitions that are identical in every respect except for their "multiple values" status. For example, a node type can specify two string properties both called X, one of which is multi-valued and the other that is not. An example of such a node type is nt:unstructured.²⁰

4.2.1.4.4 Performances and limitations

Apache Jackrabbit is optimized for small to medium sized child nodes set²¹. This means that the performances are still guaranteed up to ten thousand nodes per node. In our case it is not a problem for small medical offices, because in a whole career such number of patients is quite impossible to reach. But this observation is no longer true for big structures like the *CHUV* or even the *Hôpital Regional de Sion* and they quickly can reach such number of patients. To avoid that limitation, instead of storing the patient health records (PHR) inside a node directly attached to the root node (Figure 6), we can create a different structure by inserting an additional node between the root node and the patient node (Figure 7). In the below example we chose the year of file insertion, but we can imagine many other manners to store the files. That solution has a great advantage. In order to avoid having huge repositories we can transfer data from the main repository accessed often to a less visited server that will serve as archive server. The node export from one repository to another one is very easy and using such solution we avoid visiting all nodes in order to transfer them from the main server to the archive server. We only have to export one node with all the child nodes.

²⁰ <http://jackrabbit.apache.org/node-types.html>

²¹ <http://wiki.apache.org/jackrabbit/Performance>



470

Figure 6 Default repository storage structure

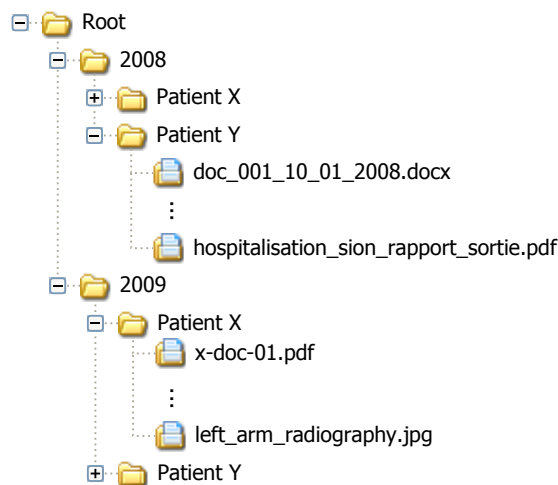


Figure 7 Proposed repository storage structure Default

4.2.1.4.5 Security

475 The security in Apache Jackrabbit is very basic when creating a transient repository. The accesses to the repository can be password protected but otherwise nothing more have been defined to render the solution more secure. To have a better security we have to define in configuration file of the repository other authentication and authorization modules.

480 Apache Jackrabbit uses JAAS (Java™ Authentication and Authorization Service) to authenticate users accessing it. JAAS is the standard way in java to authenticate someone through Kerberos, certificates, LDAP, or plain text login. Other authentication methods can be developed and used within it.

The second major problem with Apache Jackrabbit repository is that there is no encryption system that will protect files from being read by unauthorized persons.

485 To solve this problem, one solution consists in encrypting the files that we will store into the repository. The main problems raised are where to store the encryption key used. We might use a single key to encrypt all the files or, for a better security, we could use a different key for each file.

If we choose the first proposition, a single private key, we can use the data provided to log into the repository to create the encryption private key and then encrypt and decrypt the data stored. It is very important to keep in mind that if files are encrypted with a key, changing that key will make impossible to decrypt files that are already encrypted with another key.

If we choose the second proposition, we need then to find a solution that will grant a very high level of security to the encryption key storage and grant an access to those keys to everyone who has credentials to read the file content.

4.2.1.5 Encryption/Decryption APIs

For files encryption we shall use the Java™ Cryptography Architecture (JCA)²² and Java™ Cryptography Extension (JCE)²³ libraries that provide all the classes needed to encrypt and decrypt the files. This API provides a large number of encryption algorithms and key generators.

4.2.1.6 Encryption of medical data on a grid

A solution to grant a secure storage of the encryption keys is described in (Seitz, Pierson, & Brunie, Encrypted Storage of Medical Data on a Grid, 2005). They use a certain number of key servers where keys are partially stored using a secret sharing algorithm.

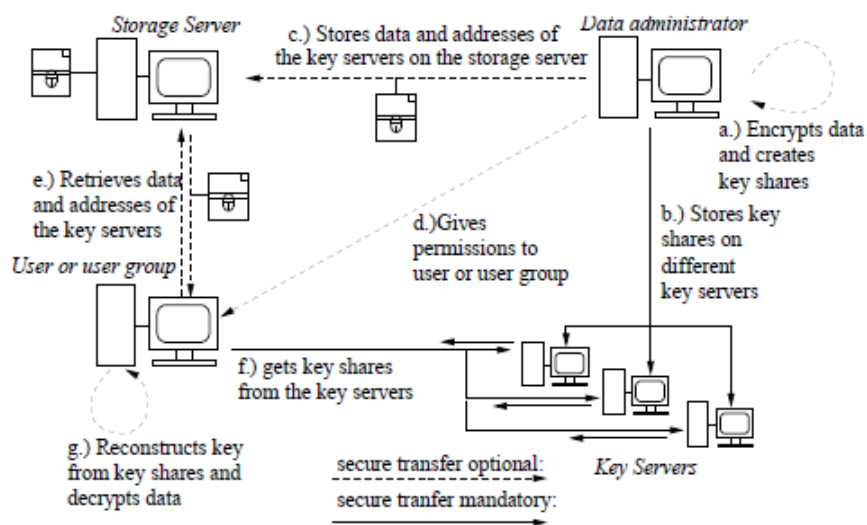


Figure 8 Access procedure when shares of the decryption keys are stored on different key servers²⁴

²² <http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html>

²³ <http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html>

²⁴ (Seitz, Pierson, & Brunie, Encrypted Storage of Medical Data on a Grid, 2005)

Secret sharing algorithms can split a key over n key servers but only m servers with $m < n$ are needed to reconstruct the key.

510 **4.2.1.6.1 Benefits**

This approach has several positive points. The first one is that the key don't resides on a single server and then if someone wants to decrypt a file he will need to hack m servers, which is more difficult than to hack only one. As we do not need to query all n servers to recover the key, we can balance all the requests over the online servers, and a loading
515 balance can be done over the key servers preventing them to be totally flooded by a very important number of requests.

4.2.1.6.2 Disadvantages

The problem with this solution is that we need a new hardware structure composed by a certain number of key servers. This means more costs for the confederation or for
520 states. The second problem is that accesses to those servers take time. This solution risks to slow down considerably application performances and responsiveness during the get, put and update operations, this is clearly not the goal we want to reach.

4.2.1.7 Access to the web service methods

To avoid access to the public web service methods, we need very restrictive access
525 policies.

We can by example allow only calls from a particular class and/or ask for credentials to verify that the user can access the web service method. Just like resources in a Windows™ domain. Normally when trying to get a resource we need to get a ticket that ensures we are correctly authenticated and then a ticket that ensures that we are
530 allowed to access to the resource. We only can access the resource after presenting both tickets.

Restricting access to the web methods will grant a higher level of security impeaching a hacker to call directly the method of the web service and trying to access directly the files without having to provide credentials or other login information.

535 **4.2.1.8 Conclusion**

The present solution responds to all requirements and has a great advantage, it is quite simple to implement and don't need complex systems to work fine. A simple desktop computer with a GlassFish server installed and the web service can run on it.

4.2.2. Web service with standard file system

540 The schema below shows the possible architecture of the storage manager service and dependencies services useful to it to make document translation by example. This solution uses Jackrabbit to manage files in standard file system. Files are stored into directories. Each node corresponds to a directory and each property corresponds to a file.

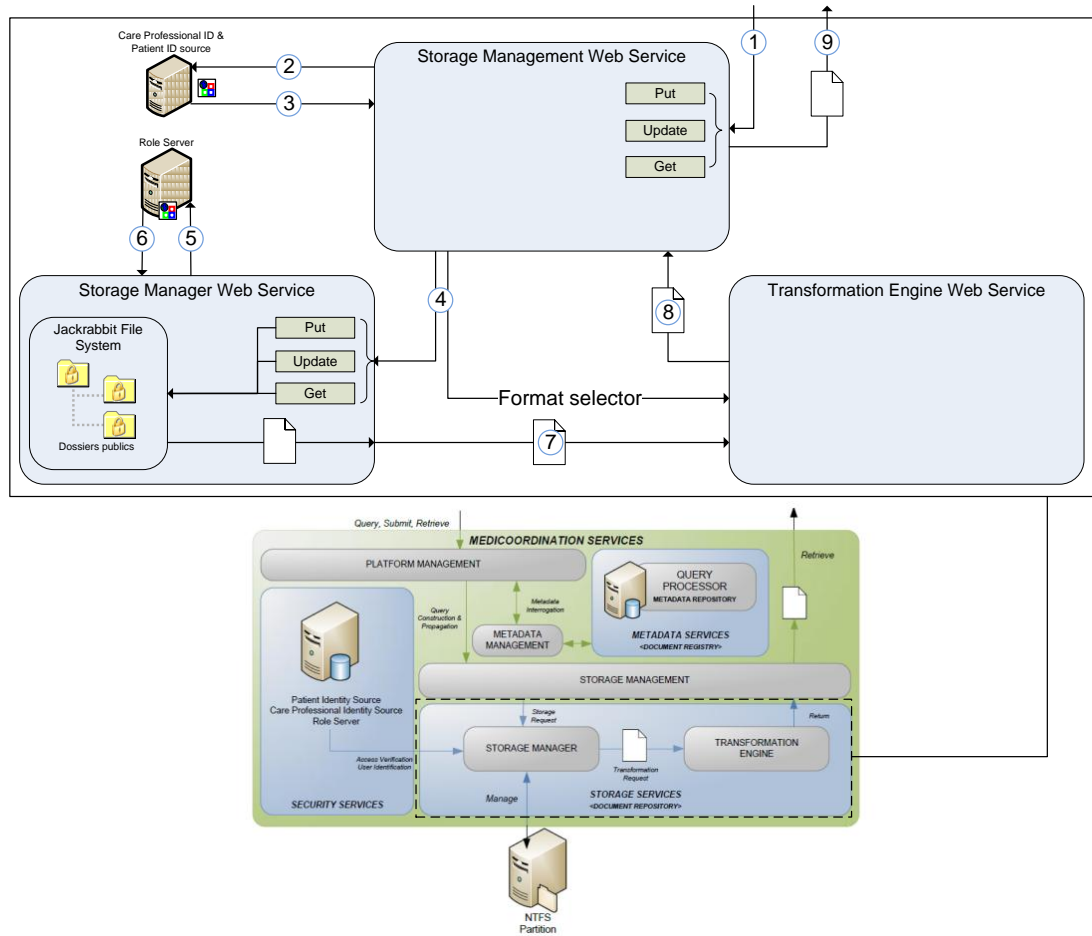
545 **4.2.2.1 Schema**

Figure 9 Storage Management using standard file system

4.2.2.2 Description

550 This version is considerably the same as the solution proposed above except that we do not use Apache Jackrabbit to manage the files but a solution programmed from scratch.

In terms of security the same rules are applied. This means set up of policies that define who can access the web service methods, file encryption, credentials to access the file system, and SSL connections to and from the web service.

555 The major difference stands in the manner we will store the files. Instead storing them inside a repository with indexes and all the mechanisms proper to a repository we will store them in a special folder on the disk.

4.2.2.3 Benefits

560 The biggest benefit using a standard file system to store files is related to the backup that will be easier to undertake. The second benefit is more psychological. With a standard file system the files are not splitted or hided. Then if there is a problem we can recover files easily, by copying them into another storage media, he have the feeling

that the files exists whom is not the case with a repository where files are splitted and hided inside à complex data structure.

565 4.2.2.4 Disadvantages

But with such solution the disadvantages are greater.

1. We lose storage space; an operation system like windows will reserve disk space for file storage by clusters. The size of clusters depends on configuration but normally it is 4 Kb. If the file size is exactly 4 Kb then no space is wasted but if it is only 100 bytes then we lost 3 Kb and 924 bytes of storage space.
2. It is very slow. The Jackrabbit team does not recommend using such file system into production projects.
3. We lose the availability to have versioning information, except if we decide to create programmatically such mechanism.
4. We need to take into account the several file systems available with all specifications and limitations as described in Table 2 and finally.
5. We lose the possibility to attach complementary information to a given file.

Below we have just an overview of the most common file systems and their limitations. We can imagine the complexity to create a File manager that will allow our Web service to store securely the files over so many different configurations.

Table 2 common file system limitations

File System	CD-ROM	ISO 9660	DOS / Win3.1	Win 95/98	Win NT 4.0	Win 2000, XP	Mac OS	OS 2	Linux	Unix	Journal	Max. volume/partition/file system size	Max. file size	Max. files per volume	Max. files per directory	Max. filename / pathname length ⁹
FAT12 ¹⁵	×	✓	✓	✓	✓	✓	✓	✓	✓	×	×	16 MB	2 MB (512 byte cluster)			8+3
FAT16	×	✓	✓	✓	✓	✓	✓	✓	✓	×	×	2 GB ² implemented 4 GB	4 GB	65,536	512 ¹	8+3 / 64
FAT32	×	×	✓	×	✓	×	×	×	✓	×	×	32 GB ⁴ implemented 8 TB theoretical limit	4 GB	4,177,920	65,534 ¹	255 / 260
NTFS	×	×	×	✓	✓	×	×	×	×	×	×	16 TB ⁶ with 4 kB cluster 256 TB with 64 kB cluster 16 EB theoretical limit	16 TB implemented 16 EB theoretical spec.	4,294,967,295	4,294,967,295 ²	255 / 260
HPFS	×	×	×	×	×	×	×	✓	×	×	×	64 GB implemented 2 TB theoretical max	2 GB implemented 4 GB theoretical max.			254 / 260
ext 2	×	×	×	×	×	×	×	×	✓	✓	×	4 ⁷ – 16 TB 32 TB theoretical 8 kB block 8 EB max 64-bit	2 TB with 8 kB block 32-bit 8 EB max 64-bit	10,000 ⁸	10,000 ⁸	255 / 4095
ext 3	×	×	×	×	×	×	×	×	✓	✓	✓	4 – 16 TB 8 EB max 64-bit	2 TB with 8 kB block 8 EB max 64-bit	10,000 ⁸	10,000 ⁸	255 / 4095
Reiser FS	×	×	×	×	×	×	×	×	✓	✓	✓	16 TB with 4 kB block	8 TB implemented 1 EB theoretical max.	4,294,967,293	4,294,967,292 1.2 million ¹⁸	
XFS	×	×	×	×	×	×	×	×	✓	✓	✓	2 TB 16 EB theoretical max.	8 TB 8 EB theoretical max.			
JFS	×	×	×	×	×	×	×	×	✓	✓	✓	2 TB 32 PB theoretical max.	2 GB (varies with o/s) 64 GB theoretical max.	64,000		255 / 1023
JFS 2	×	×	×	×	×	×	×	×	✓	✓	✓	4,096 TB theoretical.	1 TB 32-bit kernel tested 16 TB 64-bit kernel 1 PB theoretical max.			255 / 1023
ISO9660 ¹⁰	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×		2 GB	65,635	253 ¹⁶	8+3 or 32 ¹¹ / 255
Joliet ¹²	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×					64 to 110 (or 128)
Rock Ridge	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×					
UDF ¹³	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	×	1 TB implemented 128 TB v1.5	Volume size			127 implemented 255 / 256 max.
HFS	✓	×	×	×	×	×	×	×	×	×	×					
Roméo ¹⁴	✓	×	✓	✓	✓	×	×	×	×	×	×					128

4.2.2.5 Conclusion

Even if this method seems to be easier to implement we need to take into account the complexity of creating a solution that will work with all the different file systems available at this time, we need to take into account the file systems limitations in terms of maximum directories and files per directory and pathname lengths. Creating a repository on an ext3 partition will imply that we will not be able to have more than ten thousand files per disk which is clearly not enough for a Hospital.

4.2.3 Web service with external data base connection

The last proposition shows a Jackrabbit repository using an external database to store files.

4.2.3.1 Schema

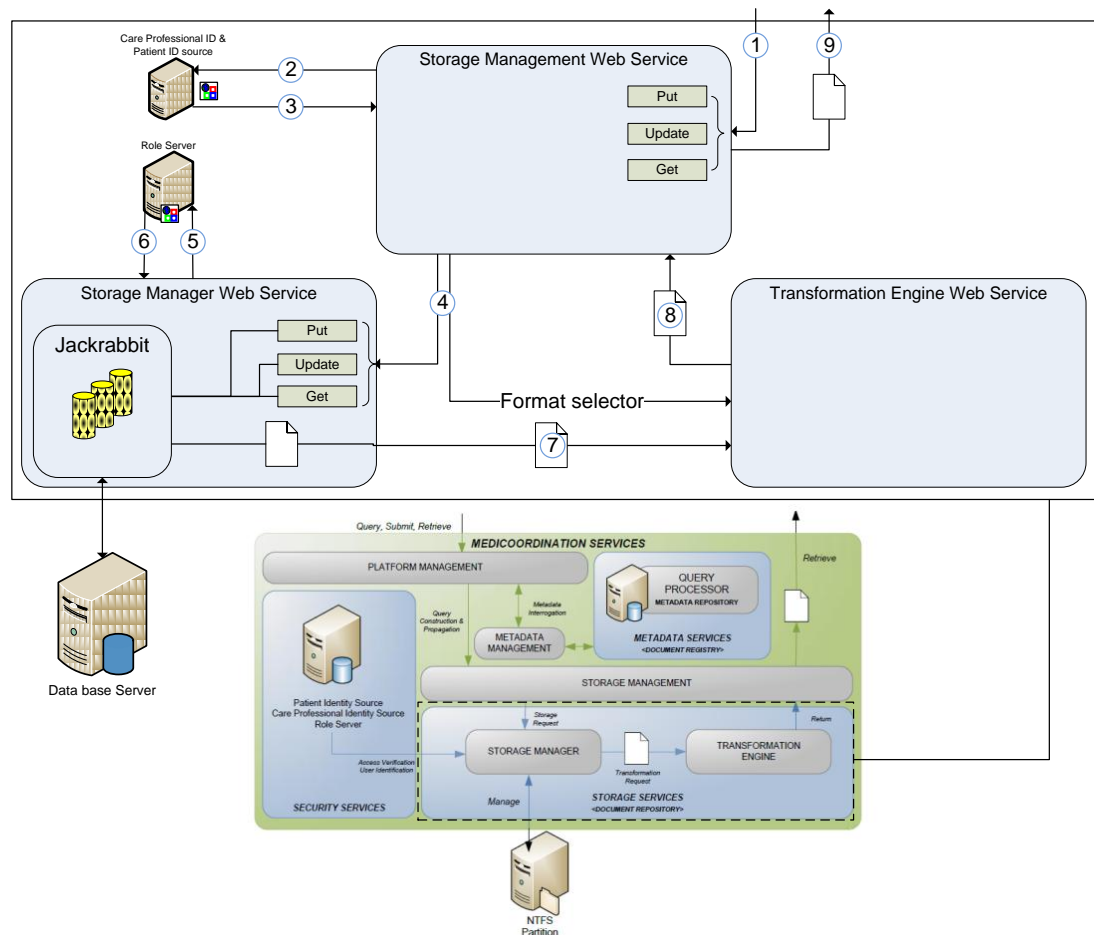


Figure 10 Storage Management using Apache Jackrabbit piloting an external database

4.2.3.2 Description

The next and last proposed solution is basically the same as the two others except that we don't store the files and mapping identifiers into hard disk but directly into a database.

As we can see the global architecture remains unchanged. The only change is that the database server is a standalone server and the Storage Manager Web Service acts as a relay between the Storage Management Web Service and the server itself through Apache Jackrabbit. This configuration is particularly interesting because we can change the kind of database we want to use only changing some configuration lines in the main configuration file of Apache Jackrabbit.

4.2.3.3 Benefits

The advantages using an external database instead of a repository or a standard file system are mainly the performances, backup facilities, data export facilities and the security that is already implemented in most current databases.

4.2.3.4 Disadvantages

The use of a database needs technical computer skills that most medical workers don't have. It means that such solution will only be setup in a working environment where exist an IT service able to lead with databases and having good knowledge about databases and databases security.

Furthermore, we will have to setup a secured connection between Jackrabbit and the Database server. Most databases support SSL connections but we do not know if Jackrabbit does.

4.2.3.5 Conclusion

The current solution seems to be perfectly adapted to big medical structures like Hospitals. In fact managing the content through a database will allow very huge amount of data to be stored searched and queried. Using Apache Jackrabbit to be the interface between database and client requests will allow us to be discharged from all the database implementation. The main inconvenient is that such solution could hardly be setup in a small clinic because unlike the first proposition that is very automated and easy to setup, using a database will need an IT service to setup the database server and to secure it.

4.3 The chosen solution

For the Storage Management web service we choose to implement the third solution, Apache Jackrabbit piloting an external database.

This choice does not exclude the two others. In fact it will be very easy to define a field in the service configuration file that will define the kind of storage needed without having to change any line of code inside the web service or using a programming pattern that would allow loading dynamically the wanted jackrabbit solution.

This total separation between the intelligence embedded inside the service and the manner the files are stored have been fundamental in the final choice of the technologies to be used inside it. Using Apache Jackrabbit allows creating a solution that will fit requirements from small health care center to very big hospitals in a more or

640 less automated manner. Using the repository will allows the creation of a totally automated solution for small clinics and using databases will allows the creation of a robust solution to hospitals like the CHUV of Lausanne.

4.3.1 Points to take care

645 Once more we put the accent on global solution security. Because we deal with very sensitive data we cannot be approximate in terms of data security.

We will summarize the security points that must be respected:

- Secure SSL communication ports between services
- Access limited to web service functions
- File encryption
- 650 • Secure communication port between web service and database
- Authentication of patients and medical workers (ex: Kerberos)
- Accesses to files limited by file access roles
- Backup facilities and versioning

655 Most of databases allow creating a secured data connection. Because database server can be distant from the web service, this solution must be used in order to keep intact the security chain and data protection from the very beginning of the transaction until the end of it.

4.3.1.1 Limitations of the solution

660 The limitations of the solution are deeply linked to the limitations of the database used (Oracle, MySQL) and the internet connection. Because Apache Jackrabbit acts only as proxy the limitations of nodes that have been spotlighted in the first solution proposition are no more a problem.

4.4 Conclusion

665 This chapter gives a first overview of the solution that will be implemented and the technologies used during implementation phase. The selected solution responds to all requirements defined at the very beginning of the document and the use of Apache Jackrabbit brings new possibilities such the use of a different support to the repository depending on the size of the health care center.

670 This last option brings a plus to the software. The application will adapt to the needs of the health care center in terms of hardware and knowledge of the workers, because some data manager employed by Apache Jackrabbit did not need human configuration or maintenance and are therefore particularly adapted to small surgeries.

5 The Storage implementation

675 In this chapter we will about the implementation of the Storage Manager. We will firstly explain how to setup correctly the workspace environment, in terms of software and configuration. It will be particularly useful if you intend to continue with the development of the current project.

In the next chapter we will enlighten deeply the software implementation explaining more in details some technical aspects of the development. We will, by example, explain 680 the class MySQLRepository. The last is the key class of the Storage Manager Web Service.

Then we will examine the project components and the technologies employed and finally the implementation problems and improvements.

The tools explained in the “5.1 Tools” chapter are not critical to the good deployment of 685 the project and can be changed to others that you are used to.

5.1 Tools

This section will describe succinctly the tools used during the development phase of the current project. If you are used with this tools you just can jump to the GlassFish “5.2 Installation and setup” chapter.

690 5.1.1 NetBeans²⁵

Because this project is programmed in Java™ for portability problems, we choose to develop the Storage Manager using the well known IDE NetBeans 6.7 from NetBeans Community.

695 NetBeans is one of the most important java IDE’s actually. It supports the web services and web application creation and deployment and has a very intuitive interface.

We selected this particular IDE because it supports very well GlassFish web application server and connections with MySQL server too and because Web Services creation is a very simple task using it.

700 The choice of the programming IDE is not capital, but for development comfort it is advised that the used IDE supports the Web Services development and automatic deployment on the server.

5.1.2 Sun GlassFish version 2.1²⁶

GlassFish is an open source web application project developed by Sun Microsystems that supports mostly all java web technologies.

²⁵ <http://www.netbeans.org/>

²⁶ <https://glassfish.dev.java.net/>

705 For the present project we used the GlassFish web application server to run the Web
Services and the front-end web application. This choice was done by other
MediCoordination project participants and for compatibility purposes we decided to
continue using it. The main advantage using GlassFish is that NetBeans supports it very
710 well and allows a certain number of operations that are not available or hardly
configurable within other IDE but it is only merely a question of comfort.

If you continue developing this application we recommend continuing using GlassFish
because actually no tests have been made using other Java™ application servers and
therefore we cannot ensure that this project will work correctly on it.

5.1.3 Apache Jackrabbit 1.5.6²⁷

715 Apache Jackrabbit is a java implementation of the Java™ content repository API (JCR).
For more information please read the “4.2.1.3 The Apache Jackrabbit API” and “4.2.1.4
Apache Jackrabbit architecture” chapters.

5.1.4 MySQL version 5.1²⁸

The Storage Manager server uses Apache Jackrabbit to control the way data is stored in
720 a repository. We described in chapter 4, 4 Storage technologies analyze, the need to
have all the files stored in a database instead of in a standard file system for
performance reasons and security. Because this project is more a feasibility test rather
than a really application for production purposes, we decided to use MySQL instead of a
more powerful database system because we are used to MySQL server.

725 Again, the choice of this software rather than another one is only a matter of comfort.
The use of another database server will, normally, not be an insurmountable problem.

5.2 Installation and setup

We will explain succinctly, in this section, where to get the software pieces used during
development and how to install and configure them in order to run correctly the project.

730 5.2.1 Requirements

Before installing NetBeans and GlassFish you will need to install the last Java JDK
(actually the JDK 6.0) if it is not done yet. It is possible to download a bundle with the
JDK and NetBeans²⁹ or the JDK only³⁰. To install the JDK just follow the installation
wizard.

²⁷ <http://jackrabbit.apache.org/>

²⁸ <http://dev.mysql.com/downloads/mysql/5.1.html>

²⁹ <http://java.sun.com/javase/downloads/netbeans.html>

³⁰ <http://java.sun.com/javase/downloads/index.jsp>

735 5.2.2 NetBeans 6.7

NetBeans IDE is downloadable from the NetBeans website³¹. Download a version supporting the Java™ SE, Java™ Web and EE technologies. A bundle containing the GlassFish Enterprise Server v2.1 can be downloaded instead.

After download, just execute the binary downloaded and follow the installation steps. If
740 you decided to install the bundle with Sun GlassFish server you will be prompt to give a GlassFish admin username and password that will be used to administrate the GlassFish server.

5.2.3 Sun GlassFish Enterprise Server 2.1

The GlassFish Enterprise Server 2.1 standalone executable is downloadable from the
745 GlassFish³² website. The installation instructions are available on the download webpage. Once the installation has finished, and before starting for the first time the server, a certain number of steps are necessary to setup the server to run correctly the Web Services of the project.

5.2.3.1 Install all Jackrabbit libraries

750 Because we will run Jackrabbit on the server it will be preferable to install the libraries necessary to run correctly Apache Jackrabbit directly on the server instead having to deploy them with the Web Application or with the Web Service.

At the time this report is written the last available Jackrabbit version is the 1.5.6. You
will need to download it from the Apache website³³ and extracts all the libraries into the
755 ./lib folder of your GlassFish installation.

5.2.3.2 Installation of the MySQL driver

To allow web applications or Web Services to access MySQL databases we need to install the MySQL driver downloadable at the MySQL website³⁴.

After the download has finished, unzip the file and copy the jar library to the ./lib folder
760 of your GlassFish installation.

If the GlassFish server is started you will need to restart it again. The library will be loaded at the next server startup. This is valid to the Jackrabbit libraries too.

To restart the server type in the ./bin folder of your GlassFish installation:

³¹ <http://www.netbeans.org/downloads/index.html>

³² <http://glassfish.dev.java.net/downloads/v2.1-b60e.html>

³³ <http://apache.mirror.testserver.li/jackrabbit/1.5.6/binaries/jackrabbit-jca-1.5.6.rar>

³⁴ <http://dev.mysql.com/downloads/connector/j/5.1.html>

765 *asadmin stop-domain domain1* and after server shutdown *asadmin start-domain domain1* to start the domain again.

5.2.3.3 Install the java Cryptography Extension 1.2.2 (JCE) Unlimited Strength

The third step to accomplish before starting the GlassFish server is to update the policies that rule the JCA (Java Cryptography Architecture).

770 First it will be necessary to download the new policies from the Sun website³⁵. Then unzip the file and follow the instruction of the *README.TXT* file. You will need to replace the existing policies in your `<java-home>\lib\security` with the new files *local_policy.jar* and *us_export_policy.jar*.

The replacement of those files will allow using bigger encryption keys and then to have a better encryption protection.

775 5.2.3.4 First GlassFish server start

Starts a system console, goes to the GlassFish installation directory then to the bin folder and type **asadmin start-domain domain1** to start the server.

780 To change server configuration launch your internet browser and type <http://127.0.0.1:4848> to display the administration console. You will be prompted for a username and password. If you provided that information during the installation (bundle installation described in “5.2.2 NetBeans 6.7” chapter) please use them to log in otherwise use the default username **admin** and password **adminadmin** if you have not been prompted to provide a new username and password during the installation phase.

5.2.3.5 GlassFish configuration

785 Before deploying the Web Services and web applications of this project we will need to configure the GlassFish server. The configuration will be explained using the web interface provided by GlassFish but a command line configuration is also possible for advanced users.

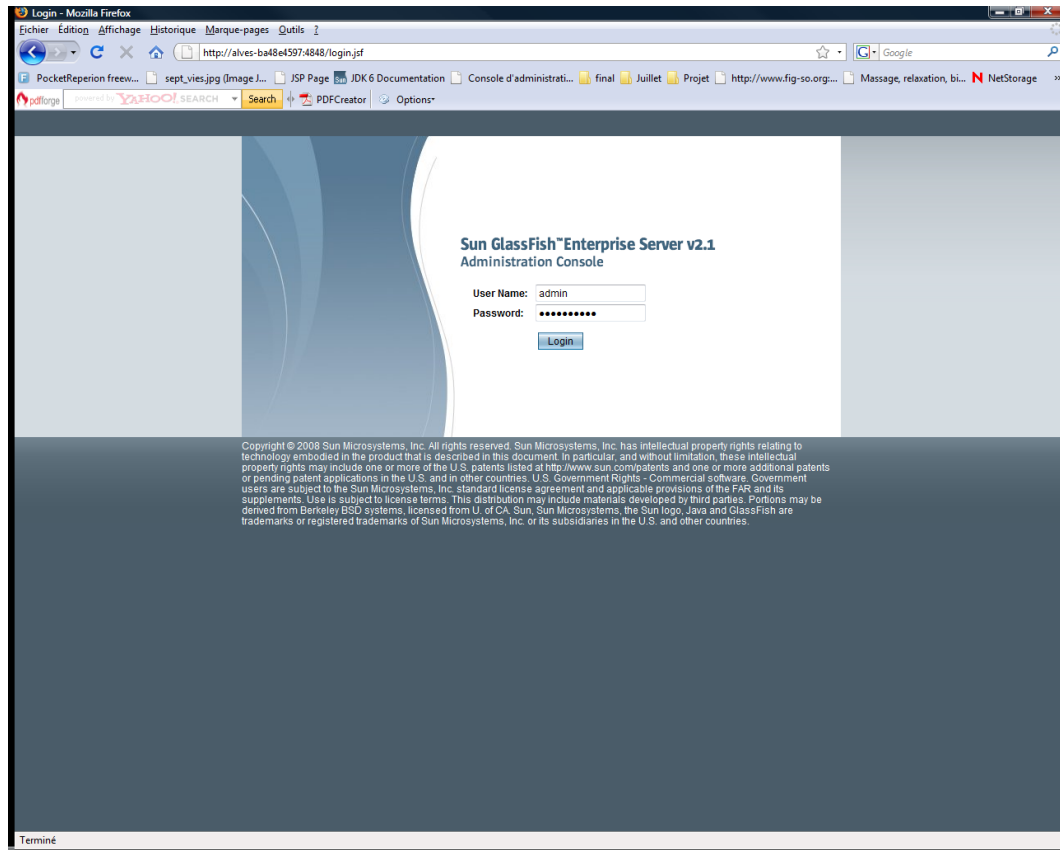
To run the project we will need to create a new user and assign to him a password.

790

³⁵ https://cds.sun.com/is-bin/INTERSHOP.enfinity/WFS/CDS-CDS_Developer-Site/en_US/-/USD/ViewProductDetail-Start?ProductRef=jce_policy-1.5.0-oth-JPR@CDS-CDS_Developer

5.2.3.5.1 Login into the GlassFish administration interface

Type in your browser <http://127.0.0.1:4848>, the address of the administration portal, and then type the username and password. For more information about username and password to provide read the “5.2.3 Sun GlassFish Enterprise Server 2.1” section.



795

Figure 11 GlassFish administration console

5.2.3.5.2 Creation of the GlassFish user

Once connected click on *Configuration > Security > Realms > File*. Finally click on the button Manage users.

800

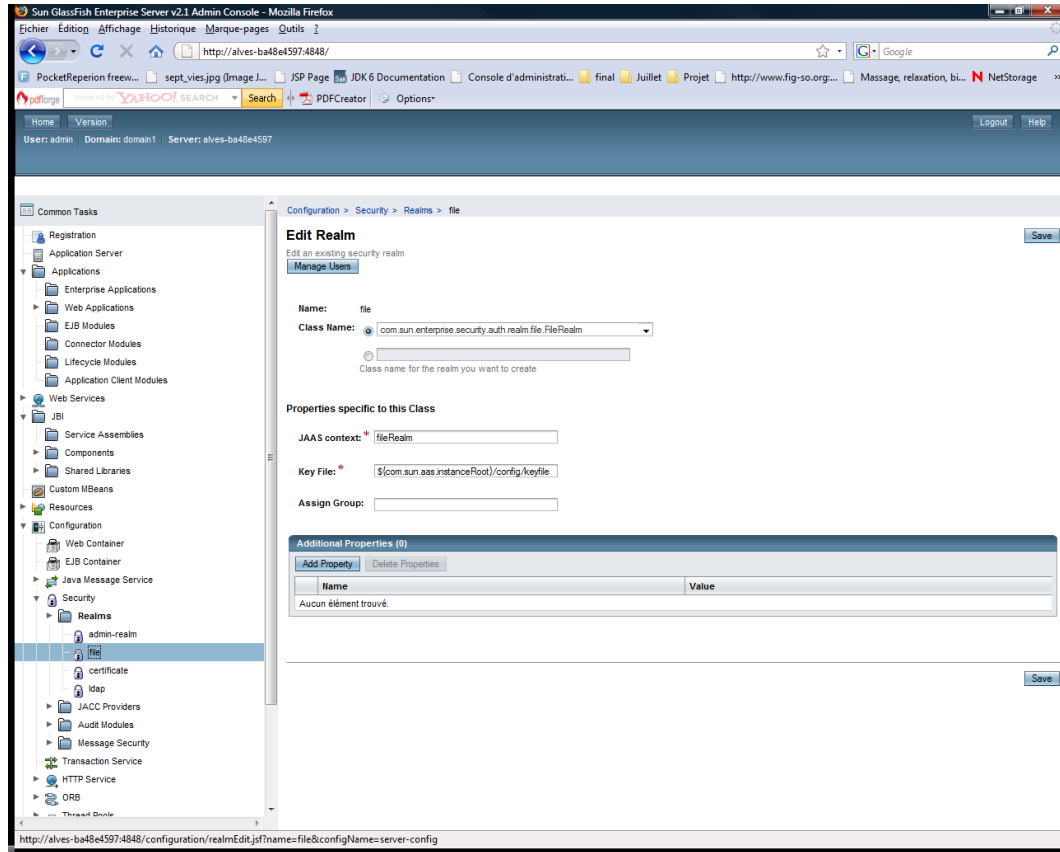


Figure 12 Realms and users

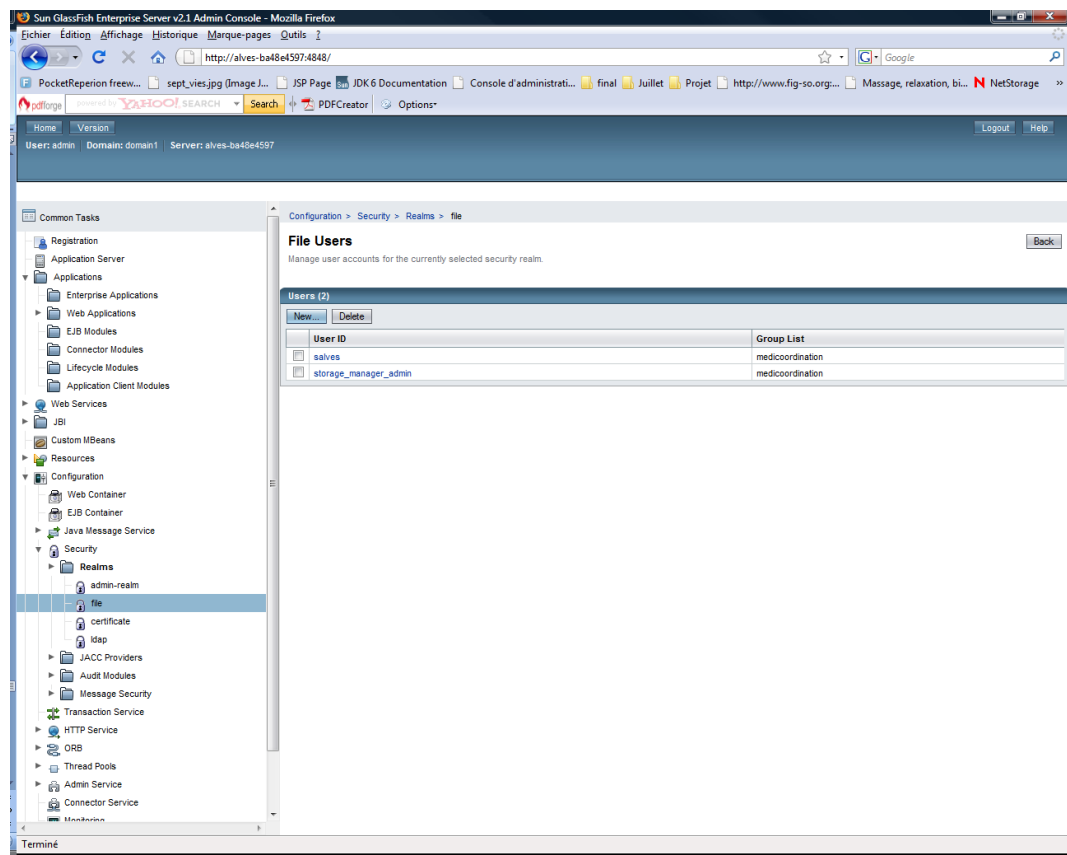
On click, a new window will be displayed listing all the users that can authenticate on the server using username password credentials (file realm). We just want to add the user **storage_manager_admin**.

805

This user represents the administrator account of the Web Services. The administrator will be able to display some component parts that normal users will not. By example, in the StorageManager Web Service, to access the *./Configuration* page the administrator will need to authenticate providing the above username and password.

810

This protection will deny access to this critical part of the Web Service to non authorized users.



815

Figure 13 Users in GlassFish

Click on the “New” button to create a new user. The Figure 14 “New file realm user” shows the form to fill in. In the *User ID* field type **storage_manager_admin**, in the *Group list* field type **medicoordination**, and type **tb2009** for the *Password*. Then click the *OK* button. The new user is created.

820

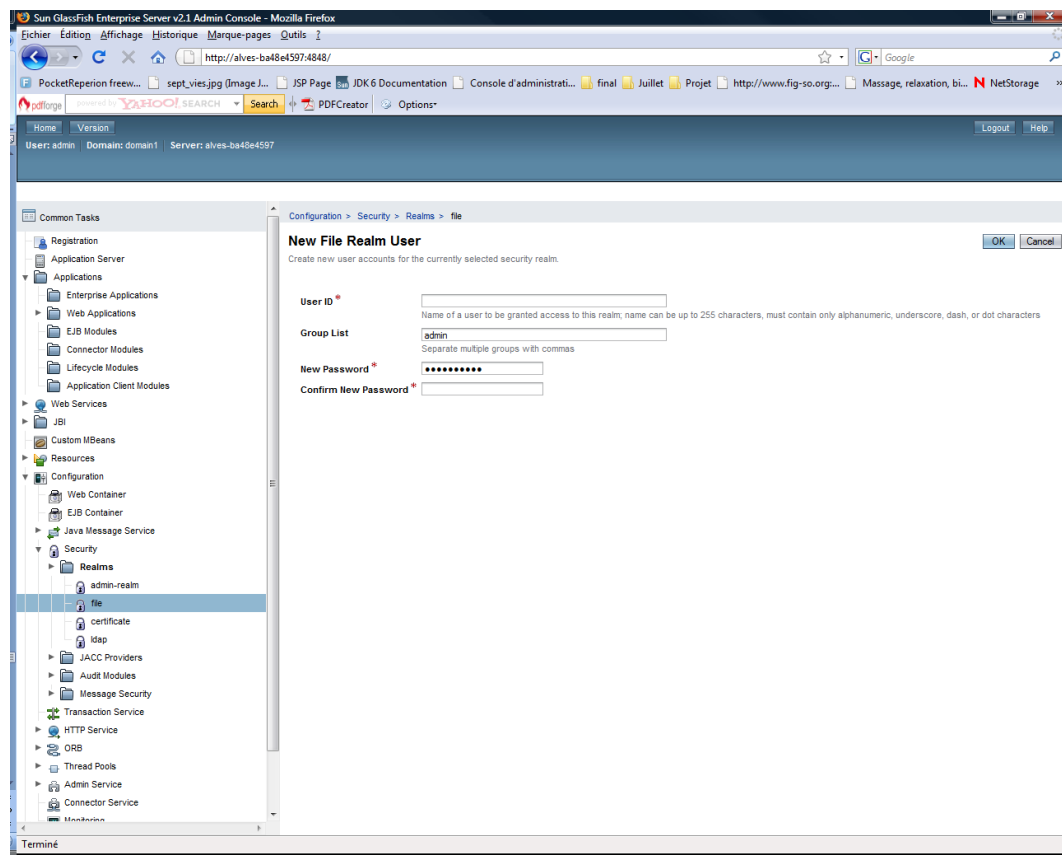


Figure 14 New file realm user

825 5.2.3.5.3 Command line “Add new user”

The command line to add the **storage_manager_admin** user must be typed in a System console and the server must be already started.

Type in the GlassFish bin directory:

```
asadmin create-file-user -groups <group_name> <user_name>36
```

830 where <group_name> represents the group name to wish the user is attached and <user_name> represents the user name we want to add. In this case the username is **storage_manager_admin** and the group name is **medicoordination**.

The user is created and the server is ready to receive the project.

³⁶ <http://docs.sun.com/app/docs/doc/820-1072/6ncp48v4e?a=view>

835 5.2.4 MySQL 5.1 Community Server

The StorageManager Web Service uses a database to store the repository and meta-data describing it. The MySQL database system has been selected because it is free, easily installable and setup.

840 Even if it is not the best database system to a production environment, the MySQL database system is perfectly adapted to development environments. Because the database system acts mainly as a data container we may use any other database system. However some changes may be necessary in the *MySQLRepositoryConfiguration* class in order to run Jackrabbit with a different database system.

845 The MySQL database system can be downloaded from the MySQL website³⁷. We installed the community server in non exclusive mode it means that the computer where we install it will not only act as a MySQL database server but can at the same time act a web server. This mode prevents the MySQL server to run threads in high priority allowing other processes to run on the same computer.

5.2.4.1 Installation

850 To install MySQL Community Server just follow the installation wizard. After the installation the configuration wizard will startup. Follow the wizard.

855 At the end of the setup wizard, you will be prompted for the root password and if the root user can accept remote connections. Defines a root password and allow distant connections or, after installation, create a new user that can receive remote connections.

5.2.4.2 Setup

860 After installation, connect to the database system with the root user or with the new user you created, and create a new schema for the repository. In future versions of the StorageManager Web Service the creation of the schema will be automated, but for time this step needs to be done manually.

After the schema creation and after granting all the rights to the schema creator the setup is finished.

5.2.4.3 StorageManager configuration changes

865 After the StorageManager Web Service deployment on GlassFish server, type <https://myserver:8181/StorageManager/Configuration> to configure the repository.

You have to authenticate with the username **storage_manager_admin** and password **tb2009** as defined in the section 5.2.3.5.3 Command line “Add new user”.

³⁷ <http://dev.mysql.com/downloads/mysql/5.0.html>

870 Change the MySQL username and password to match the username and password of the administrator of the repository schema in the database. Do not forget to change the schema name too in the configuration page.

The installation of the database is finished and we can run the MediCoordination web application front-end.

Just load projects into NetBeans and start deploying them. Follow this deployment order:

- 875
1. SecurityManager
 2. TranslationManager
 3. StorageManager
 4. MediCoordination

You can run the application typing this url : **<https://localhost:8181/MediCoordination>**.

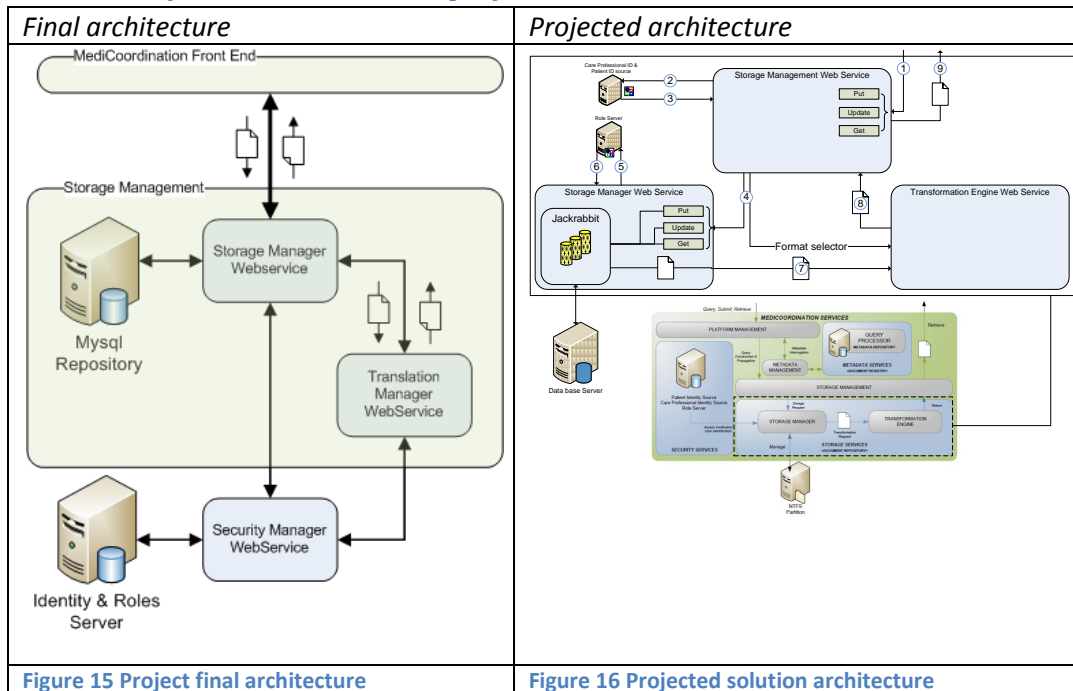
880 5.3 The solution implemented

In the chapter “4.3 The chosen solution” we fixed the solution to implement. The current chapter will show what was done in order to realize our choice. We will explain the final architecture and the objects that linked together form the Storage Management system. The next chapter will explain deeply the different components of the solution.

885

5.3.1 Schema

Figure 15, Project final architecture, shows the schema of the Storage Management as developed and Figure 16, Projected solution architecture, represents the architecture as imagined during the analyze phase.

890 **5.3.1.1 The final architecture vs. projected architecture****5.3.1.2 Description**

Some changes have been made in order to simplify the software architecture. We eliminated the StorageManagement Web Service because it was totally redundant in the design and we created the SecurityManager Web Service that handles with all authentication and authorizations questions.

To simplify the development, spare bandwidth, and avoid complex logic, the files will no more transit between three Web Services but only between the StorageManager and TranslationManager Web Services.

With the old architecture, we had to create complex callbacks mechanisms. The StorageManagement would not receive the translated file directly from the StorageManager Web Service but from the Translation Web Service after translation had finished.

This solution had some disadvantages:

1. The implementation complexity described above.
2. The bandwidth/time waist. In fact, if no document format translation is needed, the files will be sent anyway to the TranslationManager Web Service and sent to StorageManagement Service without changes. With the new design if files do not need to be translated they will be sent directly to the client.

To avoid more bandwidth waste we can imagine running the TranslationManager and the StorageManager Web Service in the same intranet. The final user will enjoy the comfortable data rates transfer and a better web application responsiveness.

5.3.2 The security implementation end to end data protection

During the analyze phase, data security and privacy was the leitmotiv. We were aware that this project would lead with very sensitive data and therefore special care had been taken to ensure its protection and privacy.

The most important security question, discussed in chapter “3 Security analyze”, was how to be sure that data will be protected from the client web browser to the MySQL repository. To respond to that question we implemented a certain number of security mechanisms on each Web Service that participates to the global system.

We will discuss more deeply about those mechanisms in the next four points.

5.3.2.1 SSL secured connections

When browsing on the web, sometimes we are required to provide private data. At this time a little lock (Figure 17) is displayed by the browser with a little message telling that we entered in a secure zone. To avoid data to be read by other people, data is encrypted and could only be decrypted by the server and vice-versa. The protocol used to secure data is called SSL for Secured Socket Layer. SSL ensures authentication, confidentiality and integrity of the data sent over the internet³⁸.

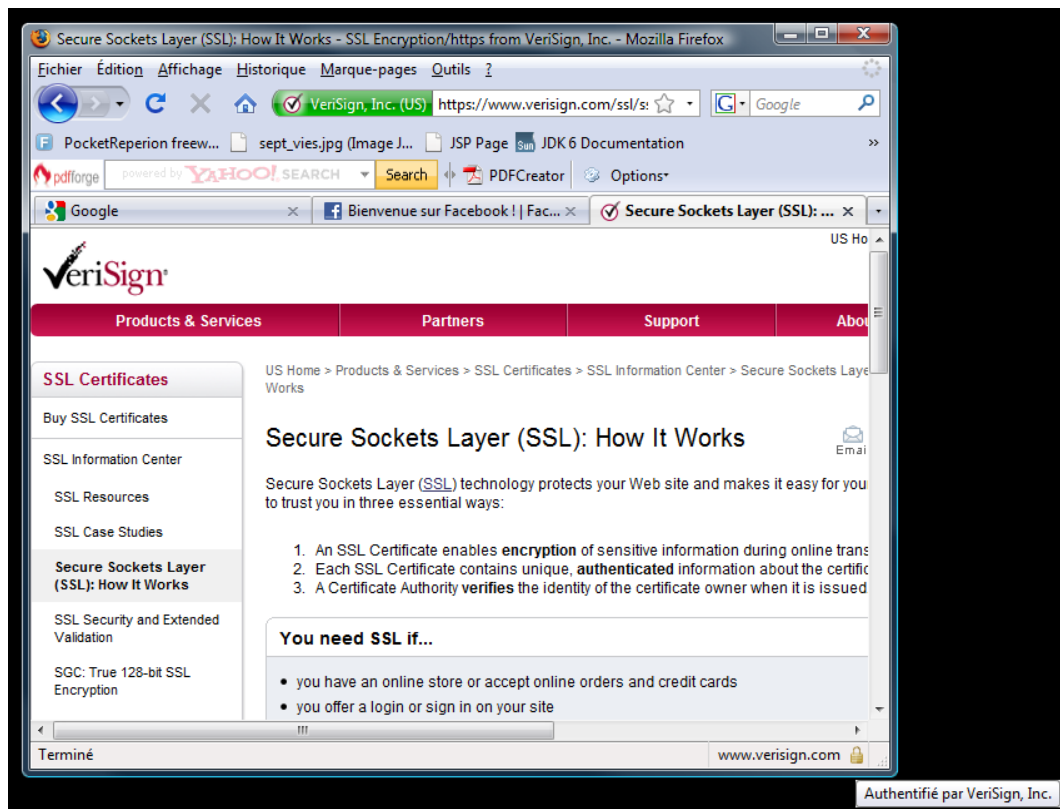


Figure 17 SSL connection and visible signs

³⁸ <http://www.verisign.com/ssl/ssl-information-center/how-ssl-security-works/index.html>

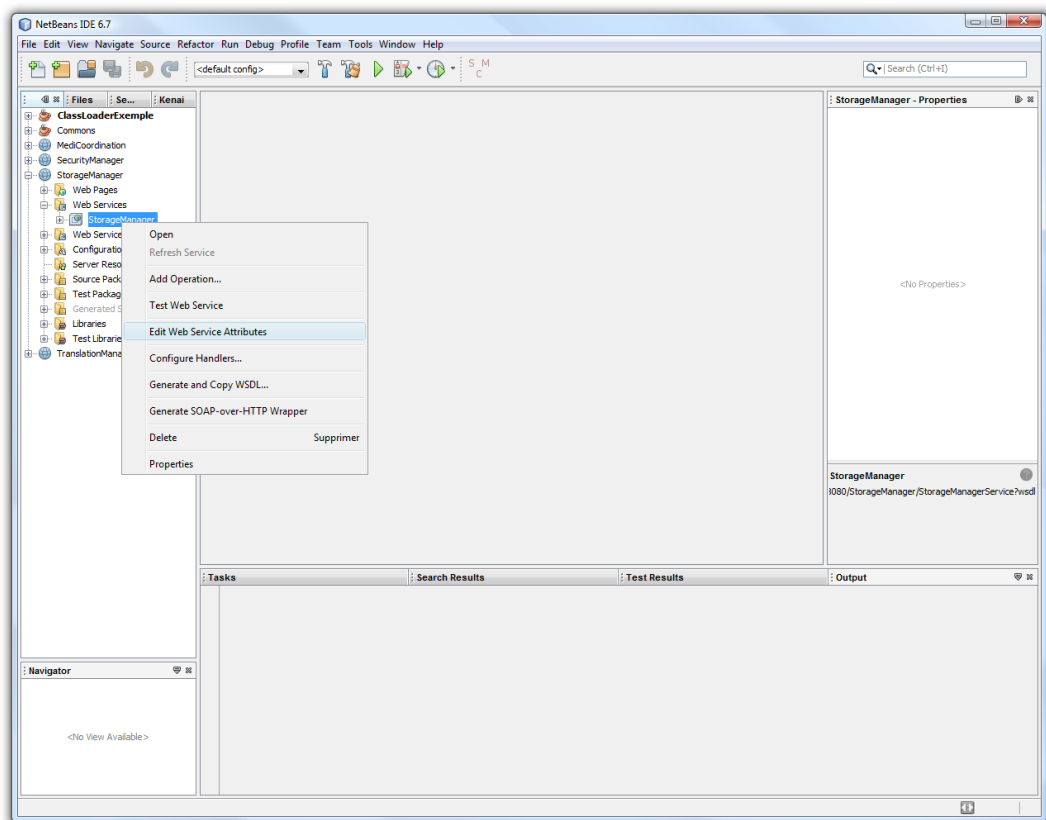
930 The lock signifies only that data will not be readable by third party persons, but SSL cannot ensure that data will not be sniffed by a hacker. Even if SSL is not THE solution it is an excellent compromise between security and easy to setup. Furthermore, nowadays it is the most used solution when talking about secured data connections.

To setup SSL connection in a Web Service, using NetBeans, there are some steps to respect. We will now explain, step-by-step, how to configure it. To illustrate it, we will use as example the StorageManager Web Service.

5.3.2.1.1 Step 1 – Define a secure Service

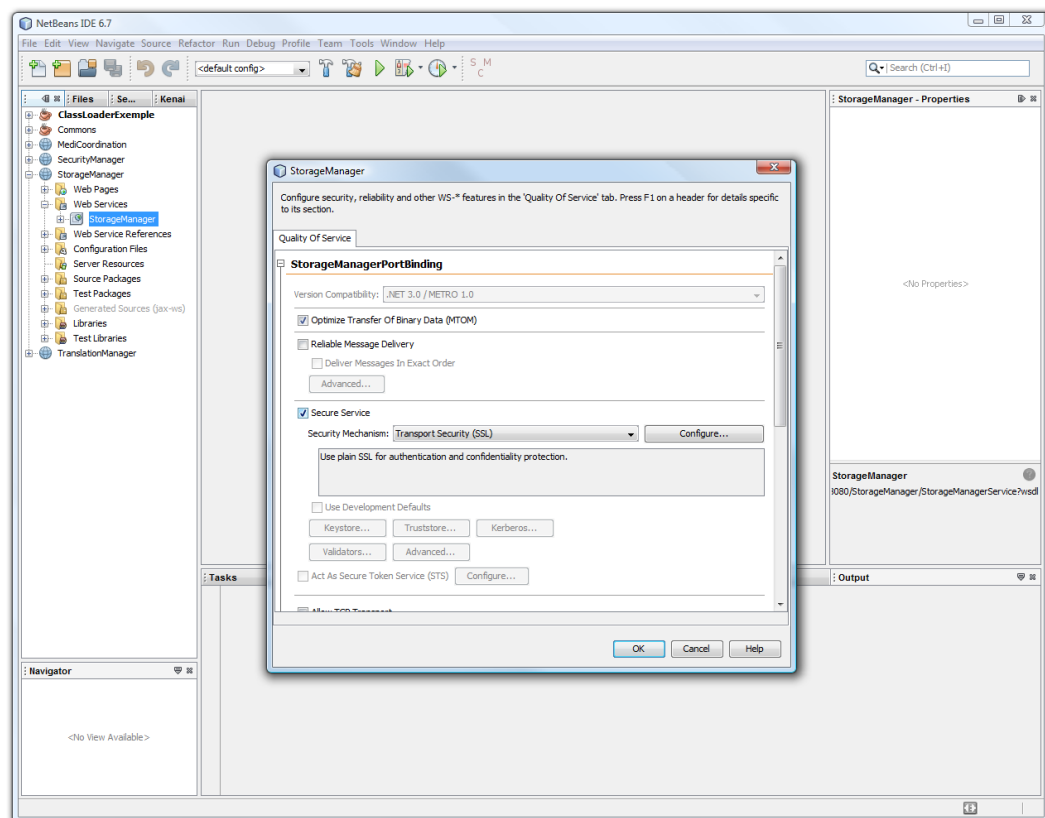
The first step in order to set working the SSL security is to change the security mechanism of the Web Service. By default a Web Service has no security mechanism attached to it.

To attach the security to the Web Service go to the *Project tab* in NetBeans and right click on your Web Service and click *Edit Web Service Attributes*. The Web Service attributes window will be displayed (Figure 18, StorageManager Web Service attributes).



945 **Figure 18 StorageManager Web Service attributes**

Select *Secure Service* checkbox and in the *Security Mechanism* drop list select *Transport Security (SSL)* and click on the OK button.



950 **Figure 19 StorageManager Security Mechanism definition**

The first step is done. Now, we have to change the `web.xml`, it is the main Web Service configuration file, to define what resources inside the `.war` file have to be SSL protected.

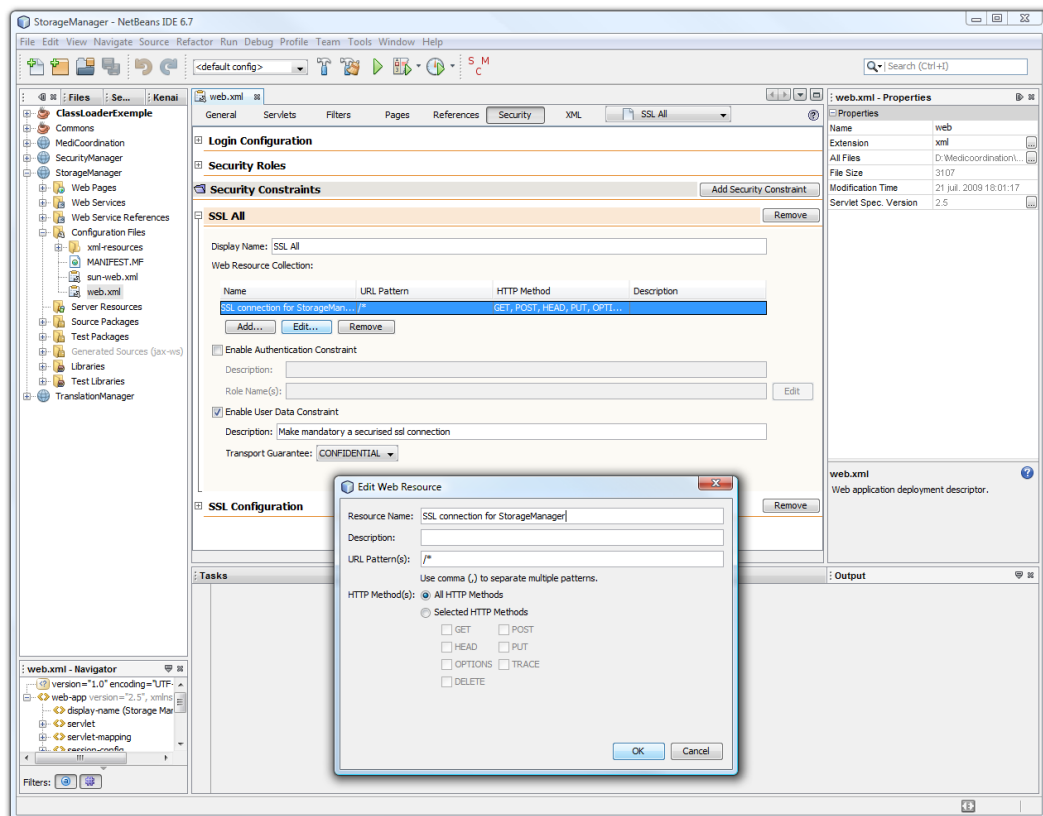
5.3.2.1.2 Step 2 - Define what resources may use SSL connection

955 The first step only defines what kind of security mechanism must be used. Now we have to define what resources need a secure connection.

Find the `web.xml` file and double click on it. Select the *Security* pane. Under *Security Constraints* section click on *Add Security Constraint* button to create a new constraint. The constraint will define what URL needs to be accessed using SSL.

960 In the *Web Resource Collection* click on the *Add* button to define a new security constraint to the Web Service. Fill the form and set the URL Pattern field pointing to the root of protected area of the Web Service.

The *HTTP Method* field defines for what http commands a SSL connection is required. By default protect all the HTTP Methods and use `/*` as URL pattern to require, for the Web Service, an SSL connection all the time.



965

Figure 20 Add new Security Constraint

At this level the SSL is not yet totally configured. To have it working you will need to check the *Enable User Data Constraint* checkbox and to set the *Transport Guarantee*. Select *CONFIDENTIAL* to enable SSL encrypted connections.

970 5.3.2.1.3 Step 3 - Project deployment

To finish, save your project and deploy on the server. When asking for a page in the StorageManager Web Service, the SSL security will be started and you will see the little lock in your browser.

975 Because the certificate used by the server is auto signed, some web browsers will ask if you want to trust the web site. Accept, creating a security exception. The web page will be finally displayed.

5.3.2.2 File encryption

Apparently Apache Jackrabbit already encrypts files when storing them into the MySQL database (nothing is told about this in the official documentation and searches over the internet gave no results), the StorageManager Web Service implements a class that will do encryption/decryption over byte arrays or streams.

980

The file encryption will ensure that even if the database is hacked no one will read the files stored inside. This feature is particularly interesting if a future programmer decides

985 to store the files outside the database, by changing the StorageManager configuration file, doing so data risks to be no more encrypted and then will be readable by anyone who has access to the file storage server.

To prevent such case, files are automatically encrypted no matter if they are stored in database in a directory. Doing so implies supplementary processor costs but it is a small price to pay to have high data protection and security.

990 The choice of encrypting files again has been made because we have no information about the kind of encryption algorithm and key size used by Jackrabbit. If the algorithm is considered as strong then we can deactivate, in StorageManager configuration page, the encryption engine (default value) letting Jackrabbit doing the task.

5.3.2.2.1 UML schemas of the cryptography packages

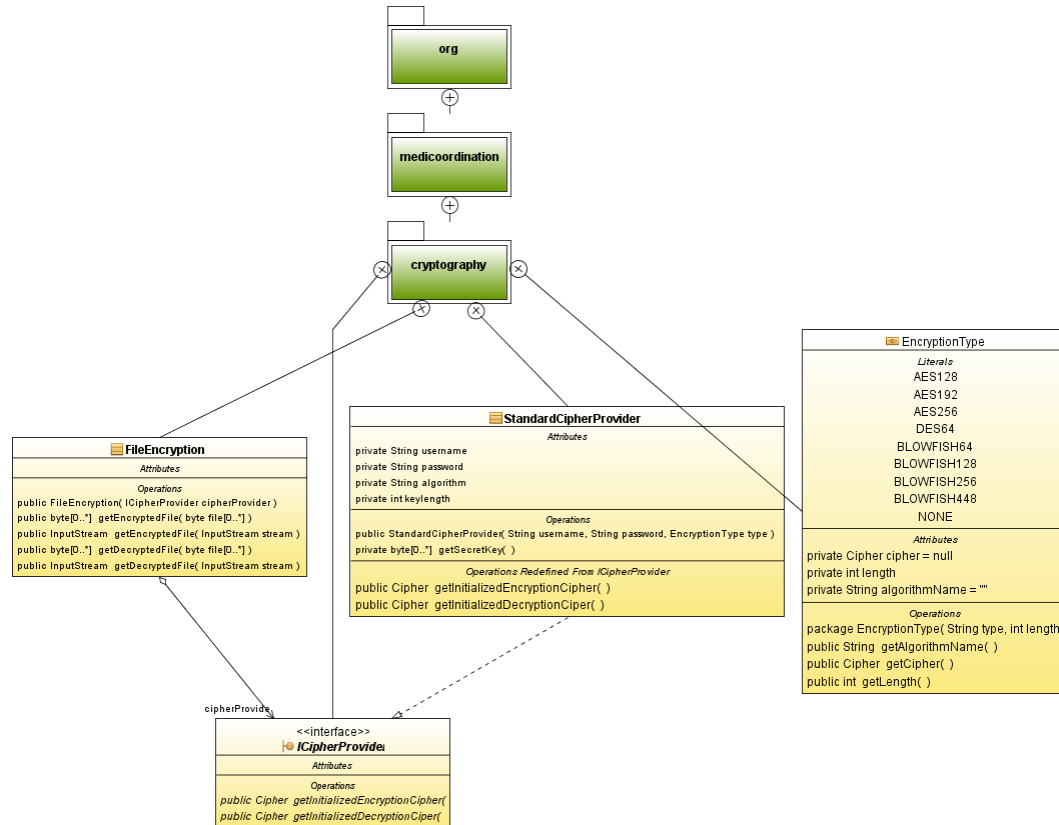


Figure 21 the UML diagram of the cryptography package in StorageManager Web Service

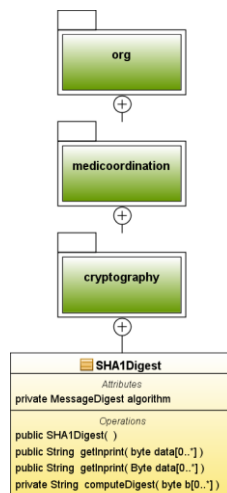


Figure 22 the UML diagram of cryptography package in Commons

There are two cryptography packages in the whole project. The first, Figure 21 the UML diagram of the cryptography package in StorageManager Web Service, is part of StorageManager Web Service packages while the other, Figure 22 the UML diagram of cryptography package in Commons, is a common package available in the Commons project. The last only contains a class that calc the SHA1 digest of a file.

5.3.2.2.2 The cryptography package classes description

All classes inside the project that touch the cryptography domain are stored in the *org.medicoordination.cryptography* package.

5.3.2.2.2.1 SHA1Digest class

At the beginning of the implementation phase, the class that calc file digests has been created to control that files transferred from client to the repository were not corrupted. This class parses a byte array and returns a 16/20 bytes length string representing the digest of the file. If a byte is changed in file, a new run over the data will produce a different digest. If the digests are not equal we can then assume that file has been corrupted.

After a better understanding of the SSL mechanisms we realized that SSL already mark data packets with a digest to ensure no one changed data inside. That SSL mechanism make the first obsolete and then totally unused in the project.

We decided to keep that class inside the project to further developments. In fact data integrity is ensured from the web browser until the reception of it by the server, but no controls are made between the moment data is stored and retrieved from database and resent back. Files can be corrupted by viruses or by human hand and there are no security methods to control and prevent it.

5.3.2.2.2.3 *StandardCipherProvider* class

1025 This class represents the encryption engine embedded in *FileEncryption* class. We use the *Cipher* class provided in the Java cryptography Extension API (JCE) to create an encryption/decryption engine.

1030 This class implements the *ICipherProvider* interface. The last defines two methods. The first returns an initialized cipher for encryption and the second an initialized cipher for decryption. The *Cipher* class in package *javax.crypto* «forms the core of the Java Cryptographic Extension (JCE) framework»³⁹. These two ciphers are used to encrypt or decrypt data in the *FileEncryption* class.

5.3.2.2.2.4 *EncryptionType* enumeration

This enumeration provides all the cryptographic algorithms and key sizes supported by the application. The JCE defines some more algorithms but some of them present security issues and therefore are not defined in the present enumeration.

1035 5.3.2.2.2.5 *ICipherProvider* interface

This interface must be implemented by all the objects that have to act as cipher providers.

5.3.2.2.2.2 *FileEncryption* class

1040 The *FileEncryption* class is the final class that we must use if files have to be encrypted. This class provides methods to encrypt and decrypt data, in byte arrays or streams form, with symmetric encryption algorithms.

1045 When instantiating it, you will need to supply a cipher provider object that implements the *ICipherProvider* interface. That object initializes the encryption/decryption engine with one of the algorithms available in the *EncryptionType* enumeration. That enumeration provides the algorithm name and key size used by the algorithm.

The use of the *ICipherProvider* interface as object type in the *FileEncryption* class will allow, in future developments, to create a new provider that will use the private key of a certificate to generate the encryption/decryption engine.

5.3.2.3 *Web Methods access restrictions*

1050 Another security issue we detected during software analyze phase and described in the deliverable *D3.1A: Interoperability Architecture* is related with Web Method accesses.

A Web Method, it means a public method tagged as accessible in a Web Service, is a public object that can be accessed by any external actors without control.

1055 The principal problem resides in the fact that some methods may not be accessed by anyone but only by authenticated users, or by certain Web Services calling the method.

³⁹ <http://java.sun.com/j2se/1.4.2/docs/api/javax/crypto/Cipher.html>

5.3.2.3.1 The Java™ Security Manager class

The Java™ API provides a very advanced security manager that can handle the security denying access to call from non authorized classes. The security is managed by the Security Manager. Permissions have to be setup in the Security Manager to deny or allow access to certain functionalities or resources.

The biggest problem resides in complexity of the system. The other problem is that a Web Service is running in a server having already a set of permissions. If one of them denies new permissions to be added from the Web Service then the Web Service have to stop executing, because no permissions can be added to the security manager or continue working without them. The last represents a security issue that cannot be accepted.

5.3.2.3.2 The adopted solution

To avoid the security issue described above we decided to build a system that would be simple, easy to setup, and that would be coupled with the authentication Web Service and able to work every time.

We notice that every user working on the system needs to authenticate. If the authentication succeeds, a token is sent to him. The token contains certain information about the current session, and about the user logged. The security token UML description can be found below.

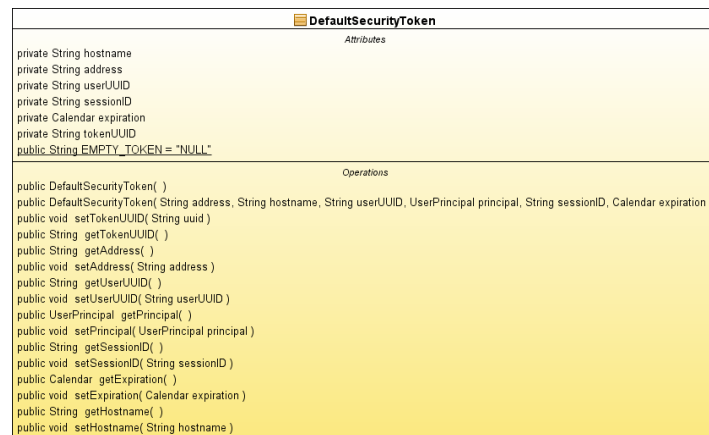


Figure 23 the security token UML schema

To protect access in Web Methods from non authenticated users, we just need to check, that the user calling the method is already authenticated. If true then the execution continues normally otherwise an *AccessDeniedException* exception is thrown.

1080 5.3.2.3.2.1 Schema

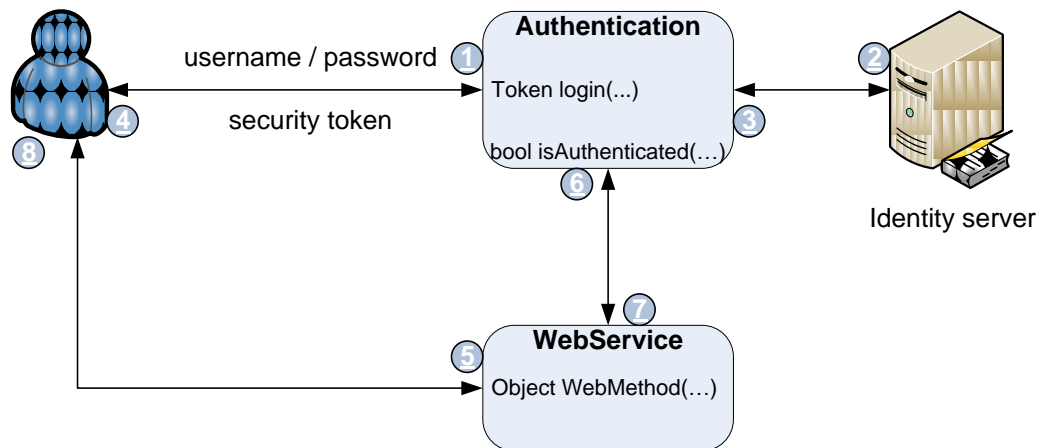


Figure 24 Web Methods access restriction system

5.3.2.3.2.2 Step-by-step description

To explain the restriction mechanisms that we implemented in Web Methods, we will simulate the access to a Web Method from a user. Steps from 1 to 4 are only realized the first time the user log into the system.

- (1) A user asks the AuthenticationManager Web Service to log into the system.
 - (2) The AuthenticationManager queries the identity server to check that the user can login or not
 - (3) The AuthenticationManager receives the response from the identity server.
 - (4) The user receives a non null security token if authentication succeeds or a null token if the user cannot login.
 - (5) The user connects to the Web Service that requires the token UUID (universal unique identifier).
 - (6) The Web Service connects to the Authentication Web Service and asks if the token UUID provided by the user refers to a valid and authenticated session.
 - (7) The Web Service answers to the request.
 - (8) The Web Method only executes and returns a result, if the token UUID belongs to a valid session.
- The Figure 25 shows how we implement the security inside a Web Method. We tested to access directly a Web Method without a valid security token and the response from the service was clear. The system returned an *AccessDeniedException* exception.

5.3.2.3.2.3 An example by the code

```

1105  /**
1106   * @param tokenUUID The security token UUID
1107   * @throws AccessDeniedException Thrown if an Authentication object does not hold a required authority.
1108   */
1109  @WebMethod(operationName="webMethod")
1110  public void webMethod(@WebParam(name="tokenUUID") String tokenUUID) throws AccessDeniedException{
1111      Authentication authenticationPort = securityService.getAuthenticationPort();
1112      if(authenticationPort.isAuthenticated(tokenUUID)){
1113          // ...
1114      }else{
1115          throw new AccessDeniedException("Access denied to this service. ...");
1116      }
1117  }

```

Figure 25 a Web Method protection sample code

5.3.2.3.2.4 Description

1110 The method receives the token UUID and asks the authentication service if the token refers to an authenticated session. If true, the software continues running otherwise an exception is thrown.

5.3.2.3.3 Conclusion

1115 The solution implemented allows to protect the access to the Web Service methods while remaining easy to setup. The only requirement is that the authentication service must be online otherwise all Web Services using that pattern to protect Web Methods risk to throw an *AccessDeniedException* and all operations will be aborted.

We will discuss and present more deeply the architecture of the AuthenticationManager and all the security classes attached to that service in the chapter 5.4.2 SecurityManager Web Service.

1120 5.3.3 The storage interface implementation

We touch finally to be most important and exciting part of this project, the Storage Manager and all its dependencies.

5.3.3.1 Definition

1125 The storage manager represents the core of the actual system. The StorageManager Web Service is responsible for:

1. Reception of files transmitted by medical workers.
2. Transmission of files, by version, to medical workers or patients.
 - a. The transmission is governed by roles.
 - b. A file could only be transmitted if the actor requesting it has the rights to download/upload it.
 - c. Rights over files are defined by the owner of those files, the patient.
 - d. All accesses to the Web Service have to be authenticated to ensure transaction security and authentication of the sender/receiver.
3. Versioning of files.
4. Store data ensuring data integrity and confidentiality.
5. Store data independently of the file storage support.
6. Store data independently of the operating system running the StorageManager.

1140 The StorageManager Web Service acts only as an interface between external requests and the internal repository engine. When a get/put request is received, it is responsible to deliver or not the document requested, depending on the rights over the file that the client has. It is also responsible for the file translation into other document format.

The internal repository engine, Apache Jackrabbit, is piloted by the *MySQLRepository* class. The last will initialize setup and manage all activities related with it.

5.3.3.2 Schematic view

1145 The Figure 26, StorageManager internal representation, gives a better understanding of the internal imbrications of the repository driver. We see that the Web Service drives the *MySQLRepository* that drives itself the Apache Jackrabbit repository.

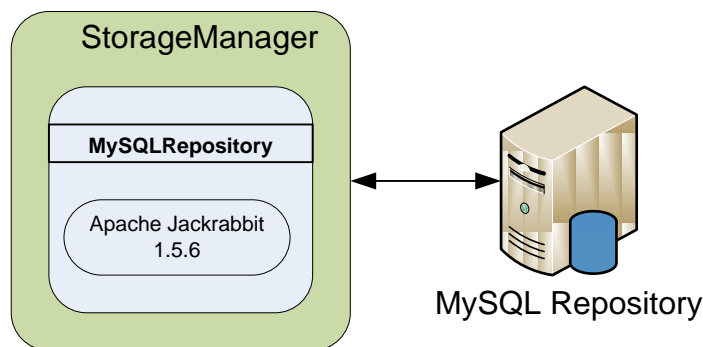


Figure 26 StorageManager internal representation

1150 Such architecture allows interchanging the repository engine, in this case *MySQLRepository*. Because *MySQLRepository* implement the interface *IRepository*, if we want to change the way how data is stored then we just have to create a new class implementing the interface and driving the repository the way we want. Finally we just need to change, in StorageManager constructor, the object implementing *IRepository* with the one we want to use.

We can imagine in a future version to have an xml configuration file with the fully qualified class name of the storage manager and the *IRepository* implementing driver we want to use with the Web Service. The constructor loads those classes dynamically through the dynamic class loader class *org.medicoordination.reflection.DynamicClassLoader* in *Commons* library and at startup we have a whole new StorageManager Web Service storing files in a WebDAV server instead of in a MySQL database repository, by example.

5.3.3.2 StorageManager Project UML Schema (collapsed version)

1165 The UML schema below represents all the packages and classes available in the StorageManager project.

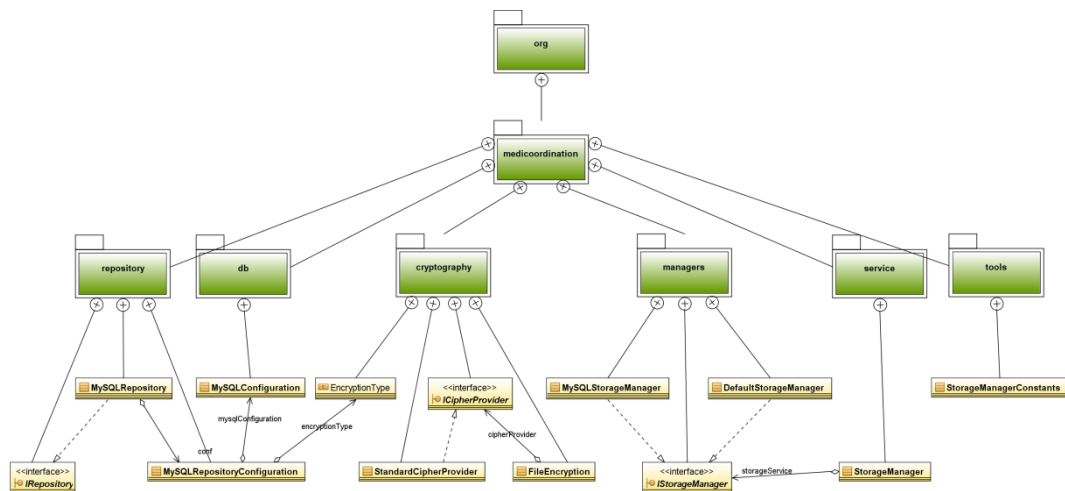


Figure 27 StorageManager UML diagram

5.3.3.2.1 org.medicoordination.db package

1170 In this package we will find all the classes that are related with database. There we find the *MySQLConfiguration* class. That class is a serializable configuration class. This class is used by *MySQLRepositoryConfiguration* class. Together, these classes contain all the initialization configuration of the StorageManager Web Service.

5.3.3.2.2 org.medicoordination.repository package

1175 This package contains all classes that drive the Jackrabbit repository. It includes the repository configuration class that contains repository authentication information and the Jackrabbit configuration class, the *apache.jackrabbit.core.config.RepositoryConfig* class.

1180 The package also contains the class, the *MySQLRepository* class, which drives the Jackrabbit content manager. That class could be considered as the most important class of the project and will have a complete chapter, the 5.4.4.2 *MySQLRepository*, to better understand its architecture. This class only exposes the methods of the Jackrabbit repository which are interesting for sending and receiving files and for session management.

1185 Below stands the UML diagram of the *MySQLRepository* class. As told before, we will take more time to better understand that class and its architecture in a latter chapter.

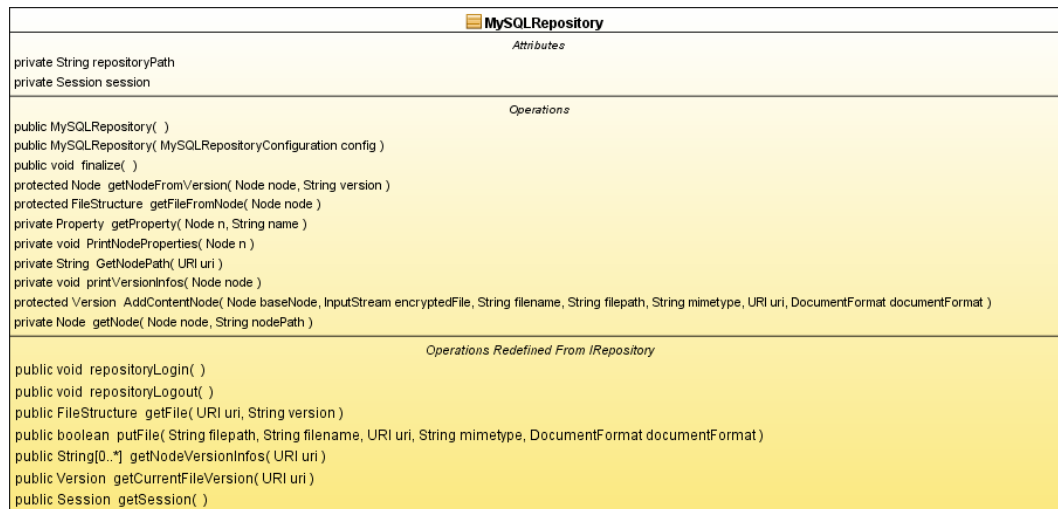


Figure 28 the MySQLRepository class (UML diagram)

5.3.3.2.3 org.medicoordination.managers package

1190 This package contains the classes and interfaces used to drive repositories. The repositories must implement the IRepository interface.

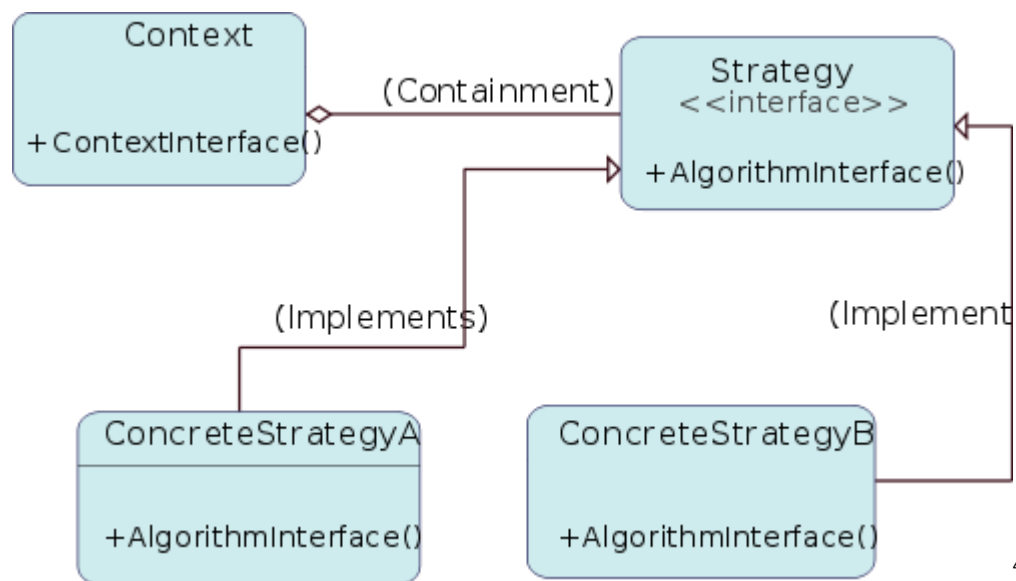
Before refactoring, that package did not exist. In fact all the code that can be found in the *MySQLStorageManager* was a part of the StorageManager Web Service itself. To allow future improvements or changes in the way the Web Service drives the repository, we decided that all control code had to move into another class, the *MySQLStorageManager* class, which may be changeable with another one better engineered. We applied the strategy pattern to allow such flexibility.

1195 In the StorageManager Web Service we only have an object, *storageService* of type *IStorageManager*, which is dynamically instantiated when the Web Service connection is created. It's easy to understand that, soon as the class we want to act as manager implements *IStorageManager*, we can modify radically the way the Web Service manages the repository by changing in the StorageManager Web Service the class that will manage the *IRepository* object.

5.3.3.2.3.1 The strategy pattern

1205 The strategy pattern (also known as the policy pattern) is a software design pattern whereby algorithms can be selected at runtime. (Gamma, Helm, Johnson, & Vlissides, 2005)

1210 The Figure 29, the strategy pattern UML diagram, shows the pattern UML diagram. For more precisions about the strategy pattern we invite the reader to examine the document cited above from pages 315 to 323.



40

Figure 29 the strategy pattern UML diagram

The algorithm selection in the StorageManager Web Service is done by dynamically loading the class that will play the role of the manager.

- 1215 Actually the name of the class is fixed in a variable, the variable *className*, but the goal is to put that class name in a configuration file.

When a new instance of the StorageManager will be created, then the class name will be read from the configuration file and an instance of the manager object will be created at runtime without fixing it in code. But actually, the name of the class we want to dynamically load still stays hardcoded in the java file. (C.f. below)

- 1220

```

30  /**
31   * Storage Manager WebService default constructor
32   */
33  public StorageManager() throws ClassNotFoundException {
34      String className = "org.medicoordination.managers.MySQLStorageManager";
35
36      ClassLoader classLoader = Authentication.class.getClassLoader();
37
38      try {
39          this.storageService = (IStorageManager) classLoader.loadClass(className).newInstance();
40      } catch (Exception e) {
41          //On error trying to load a dynamic Authentication module loads the default one
42          this.storageService = (IStorageManager) new DefaultStorageManager();
43          // Throws classnotfound exception
44          throw new ClassNotFoundException("The class " + className + " " +
45              "could not be loaded dynamically. The default class is used");
46      }
47  }
  
```

Figure 30 the dynamic loading of the storage manager class

40

http://upload.wikimedia.org/wikipedia/en/thumb/4/4c/Strategy_Pattern_Diagram_ZP.svg/500px-Strategy_Pattern_Diagram_ZP.svg.png

The next schema shows only classes that contribute directly to the driver structure. It means that we do not take into account all configuration classes or cryptography classes but we only concentrate on links and relations between classes that compound the core of the StorageManager Web Service.

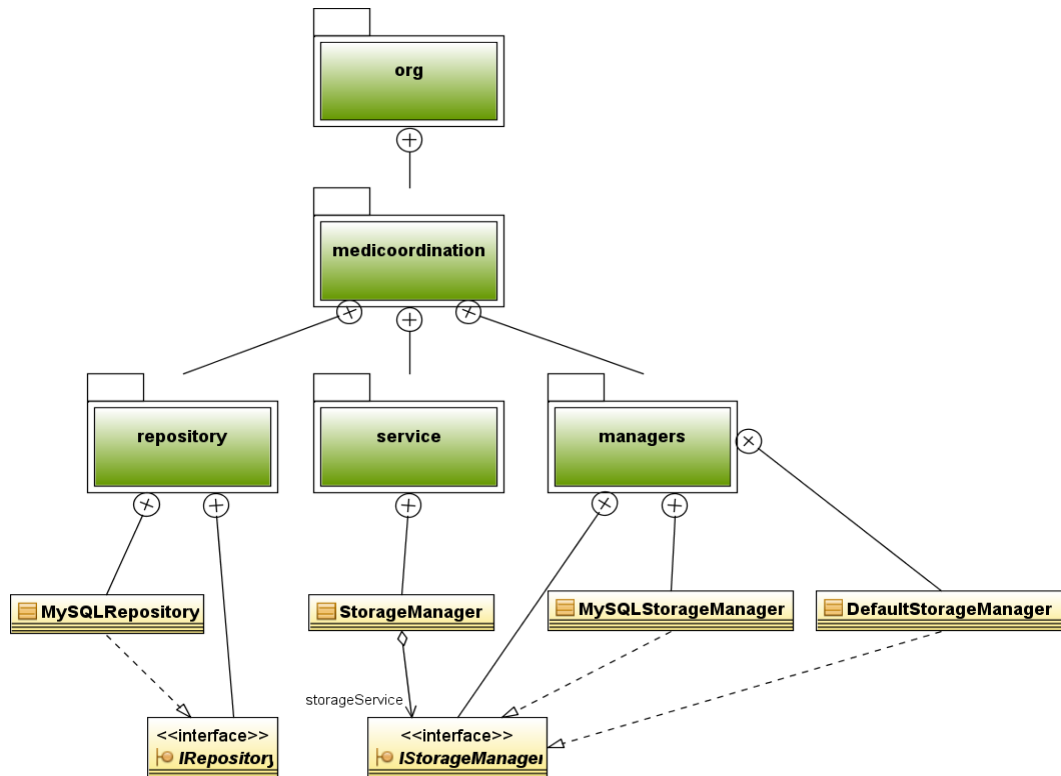


Figure 31 the core of the StorageManager Web Service (Strategy pattern diagram)

To conclude, the *StorageManager* Web Service is compound of:

1. The *StorageManager* service class, the only class visible by web clients.
2. The *MySQLStorageManager* class that implements the *IStorageManager* interface and is used as interchangeable algorithm class by the *StorageManager* service class (strategy pattern)
3. The *MySQLRepository* class that implements the *IRepository* class and acts as an Apache Jackrabbit driver. If in future we decide to change the storage support and we choose a normal repository on hard disk then we just need to create a new class, *NormalDiskRepository*, by example, which implements *IRepository*, and set that class name in the *StorageManager* configuration file. At startup the class will be dynamically loaded and instantiated and files will be stored into the hard disk repository instead of in a MySQL database repository.

5.4 Project components

In this chapter we will explain each subprojects of the Storage project and try to explain how they interact together. Every time it is necessary, we will present the architecture of certain key classes more deeply.

5.4.1 Commons library

The commons library is charged to supply certain classes that are potentially used by all the subprojects. Instead of having the same class in every subproject we code once and reuse it by including the library in the project.

5.4.1.1 The UML schema

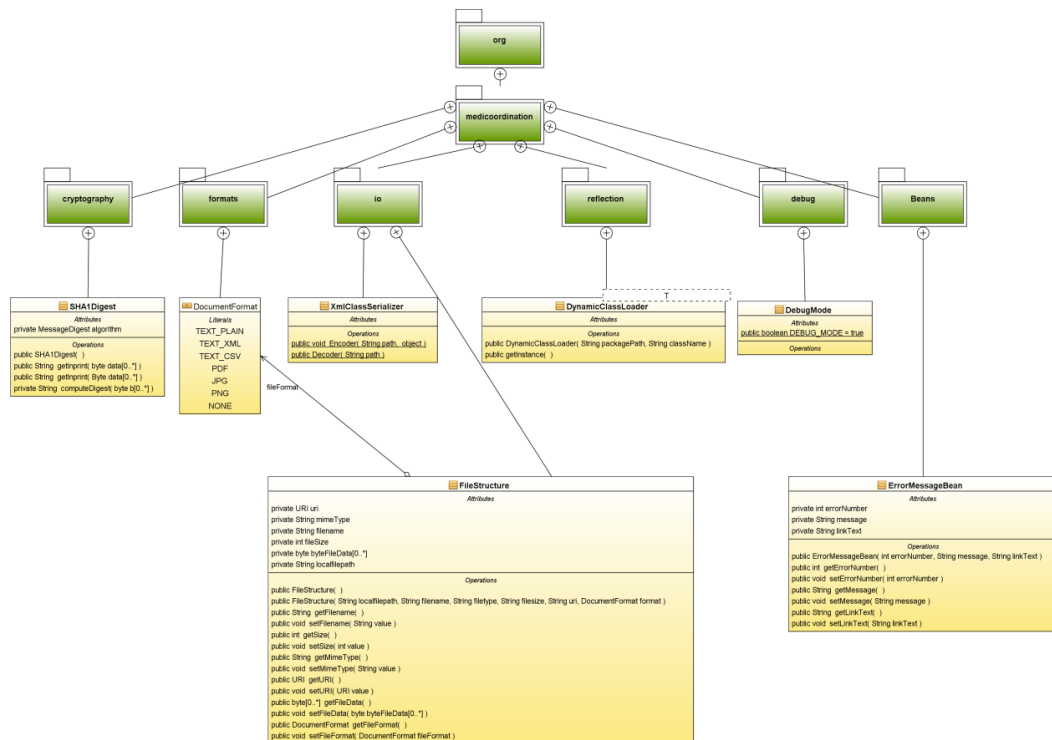


Figure 32 the commons library UML diagram

The above diagram gives an overview of the tools available to all classes compounding the Storage project. In the next chapters we will give a description of each one.

5.4.1.2 SHA1Digest class

As described in a previous chapter, this class calculates a digest from a byte array that can represents a file.

This class is particularly useful if we want to ensure that during a put/get phase the data is not altered. In fact a byte array will always return the same digest. If only a bit changes in the byte array then the digest will change too.

To ensure that a file has not changed, we just have to create the digest of a file and then compare to the digest produced after an operation that can corrupt the data with the first one.

5.4.1.3 *DocumentFormat enumeration*

1265 This enumeration is only for test purpose. The enumeration contains document formats codes. Those codes are used by the TranslationManager Web Service to define the type of the output file.

This kind of solution is not very elegant and not very flexible. Indeed, adding new formats to the enumeration will imply the recompilation and redeployment of at least
1270 the TranslationManager service. It is not acceptable in production conditions but for tests purpose we can accept such solution.

5.4.1.3.1 *Possible solution to solve the DocumentFormat problem*

A best solution would be to replace the actual numeric codes with full qualified class names. For each document format defined in the enumeration, a class that manages the
1275 translation must be accessible by the TranslationManager. When the last receives a translation request, it only has to try to dynamically load the class from its fully qualified name provided by the element in the enumeration and use it to translate the document. If no class is available, then the service returns an empty *FileStructure* or throws a *ClassNotFoundException*, which means that the service failed to find the class used to do
1280 the translation.

5.4.1.4 *XmlClassSerializer class*

This generic class allows serializing any class into its xml representation and deserializing it into an instantiated object.

It is a very useful class as soon as we need to deal with configuration files or we want to
1285 save the class content into a file and restore it later.

5.4.1.5 *FileStructure class*

This serializable class is intended to be sent by the StorageManager service to the client as a response to the *getFragment* request. The structure contains the file data and some information about de file such as:

- 1290
1. The file name
 2. The file type
 3. The document type
 4. The file size
 5. The file URI

1295 This class acts only as a data container.

5.4.1.6 DynamicClassLoader class

This class loads and instantiates a class from its fully qualified class name and the class file path. If the class file cannot be found then it throws a *ClassNotFoundException*.

This class is very useful as soon as we need to dynamically change the behavior of an object.

5.4.1.7 DebugMode class

This class defines a constant that is used to indicate that the software must work in debug mode. By example, many classes use that constant to exhibit some information that can only be displayed when debugging the software. As soon as the software runs correctly, and we do not need to debug anymore, then we set the constant to false and lots of debug messages will not be displayed anymore.

5.4.1.8 ErrorMessageBean

This class is used by the JSP pages of the front-end to carry information to the error.jsp page. The class provides three fields, the error number, the error message and the link to bring the user back to a defined state, by example to bring him back to the login page after a session timeout.

5.4.2 SecurityManager Web Service

The SecurityManager Web Service is in charge with all the authentication and authorization questions within the storage project.

This project has two Web Services, the *Authentication* and the *ResourcesAuthorisation* Web Service. The other classes of the SecurityManager are satellite classes for the two services.

5.4.2.1 UML schema

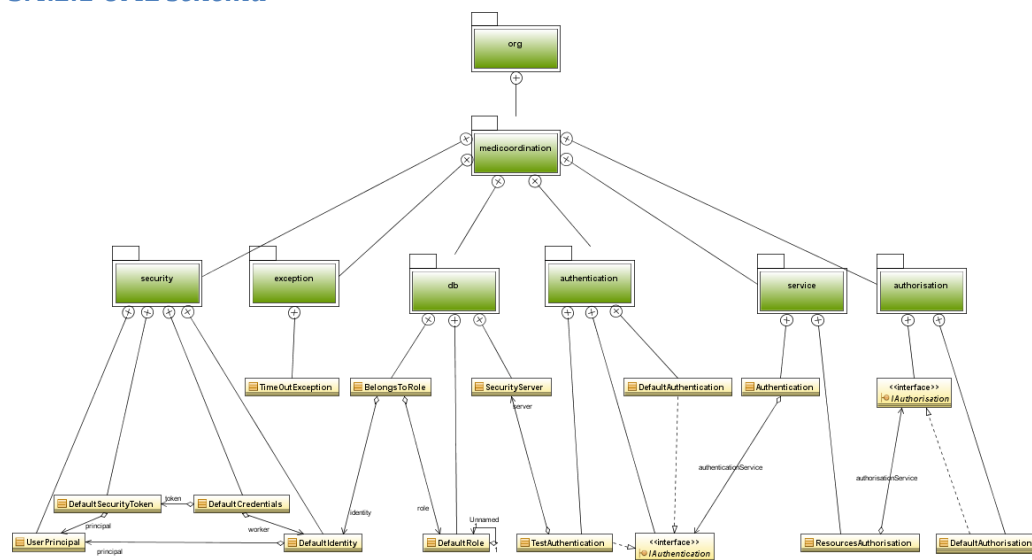


Figure 33 SecurityManager UML class

5.4.2.2 The *org.medicoordination.service* package

This package contains the Web Services java files. We will explain the role of each class in the project during the next two chapters.

5.4.2.2.1 The Authentication Web Service

1325 This service acts as a middleware between the MediCoordination project and an authentication system as Kerberos, by example. It is an authentication abstraction.

The conception of this Web Service follows the strategy pattern that was already discussed in the chapter 5.3.3.2.3.1 The strategy pattern. This methodology is particularly interesting in Web Services because it allows, coupled with dynamic class loading, to provide to clients, without redeploying the service, always the same interface
1330 but allows changing the algorithms used within the methods.

An example of the utility of the strategy pattern in Web Services conception, after deployment if we find a bug we do not need to redeploy the service. We just have to compile the class and put it inside the class path of the service. The next time it is
1335 instantiated we will dynamically load the new version of the class without the bug.

5.4.2.2.2 The ResourcesAuthorization Web Service

This service is responsible to notify the caller if the user that requested a write or read action can perform it.

As for the Authentication Web Service, this service is built using the strategy pattern and acts as an abstraction allowing the services that need such information to concentrate
1340 only on the essential, it means, to have the response to the question: "Can the user X read or write the file Y". All the queries and accesses to the database(s) are delegated to the ResourcesAuthorisation Web Service.

Below we can see the methods provided by the service. To keep simple, the first indicates if the authenticated medical worker can read a file from the patient
1345 *patientUUID* while the other indicates if the he can write a file to the patient personal electronic health record (pEHR).

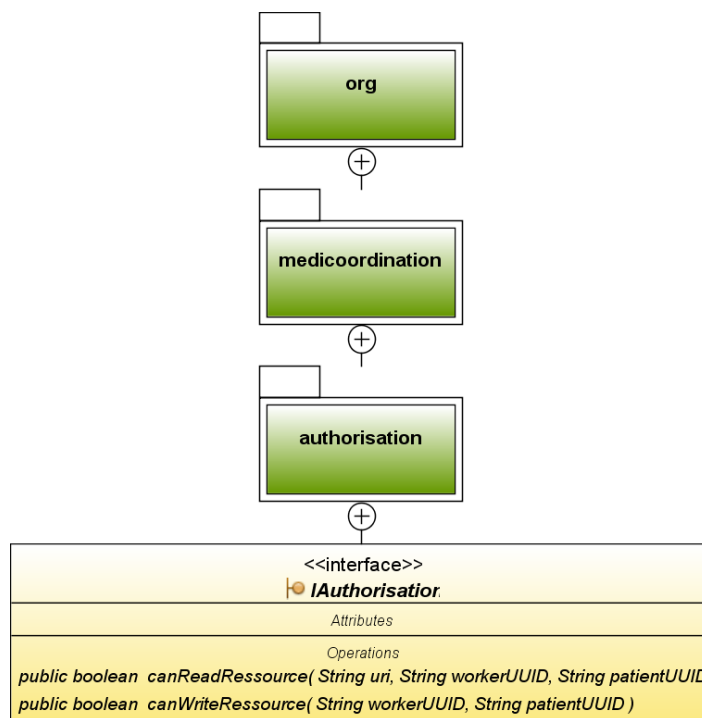


Figure 34 the methods provided by the ResourcesAuthorisation Web Service

1350 This Web Service implements the smallest set of functions useful to allow the
StorageManager Web Service to control if a medical worker is authorized to get or put
the file.

We let to the MediCoordination team the care to choose the final architecture of the
authentication and authorization Web Services, keeping in mind, that actually, the
1355 storage module is built and running using all the defined functions found in both Web
Services.

5.4.2.3 The *org.medicoordination.authorization* package

This package contains all the classes needed by the ResourcesAuthorization Web
Service. Most classes are the result of the application of the strategy pattern over the
1360 Web Service. The Figure 35 shows those classes disposed as in the Figure 29, the
strategy pattern UML diagram. The same diagram can be obtained with the
Authentication and StorageManager Web Services classes because the same pattern
was applied to these subprojects.

5.4.2.3.1 The *IAuthorization* interface

1365 This interface defines all the methods visible in the ResourcesAuthorization Web Service.
In the last there is an object of *IAuthorization* type. This object is only instantiated at
runtime and with a dynamically loaded class. The type of the loaded class is normally
stored in the configuration file as a fully qualified class name. The value is read and used
to create dynamically the new instance.

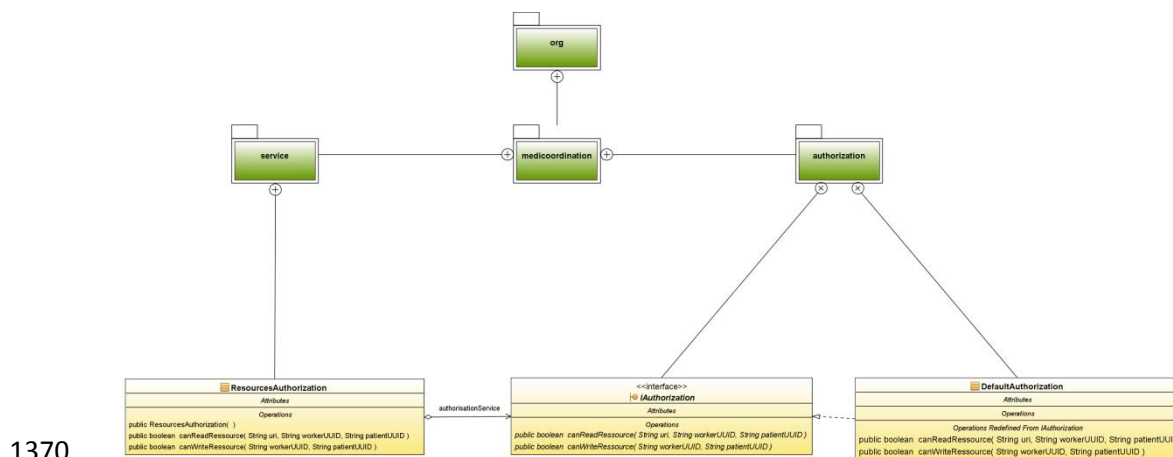


Figure 35 The ResourcesAuthorisation strategy pattern refactoring UML diagram

5.4.2.3.2 The DefaultAuthorization class

This class is the default implementation of the interface *IAuthorization* and is loaded if the dynamic load fails. This class is only instantiated when a *ClassNotFoundException* occurs and its role is to avoid the Web Service to crash. The only response sent to clients is false for read and write requests.

5.4.2.4 The org.medicoordination.authentication package

This package contains all the classes used by the Authentication Web Service. The classes, as for the ResourcesAuthorization Web Service, are obtained after refactoring using the strategy pattern. The same schema will appear one interface and two classes, a default implementation of the interface, the *DefaultAuthentication* class and the normal implementation used by the Web Service to respond to the requests.

5.4.2.4.1 Strategy pattern applied to the Authentication Web Service (diagram)

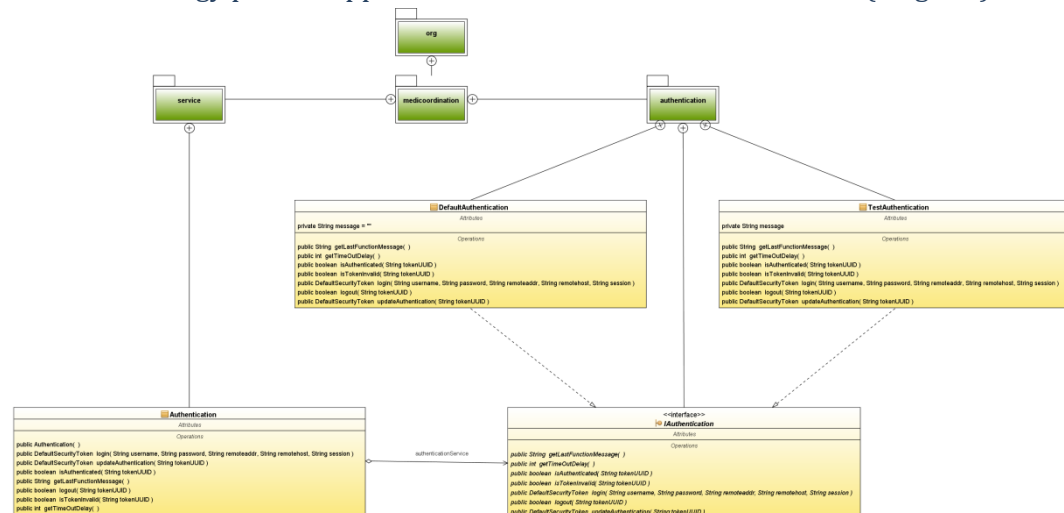


Figure 36 Strategy pattern refactoring of the Authentication Web Service (UML diagram)

As for the previous package, the authentication package has quite the same elements with the same roles, that is why we will concentrate us only on the new class, the

1390 *TestAuthentication* class. This class is a realistic working implementation of the *IAuthentication* interface unlike the *DefaultAuthentication* who is only intended to avoid Web Service crashes.

5.4.2.4.2 The *TestAuthentication* class

As introduced above the *TestAuthentication* class is a realistic implementation of the *IAuthentication* interface. It means that when a user will query the service to login, the response will not be always false.

1395 To simulate working identity servers without having to setup a complex infrastructure we created some classes in the *org.medicoordination.db* package that simulates remote database servers. Therefore, this class must be used only to test the web methods within the Web Server. A new implementation of the class, this time communicating with real servers will allow a true authentication mechanism.

1400 To have more information about methods and classes please consult the javadoc, for the current project, available in the CD that is distributed with this document.

5.4.2.5 The *org.medicoordination.security* and *db* packages

Those packages provide a certain number of classes that have a less important role inside this subproject.

1405 All the classes present in both packages are used by the Authentication Web Service. Those from the *security* package are used when the login occurs while those from *db* are used to simulate database servers.

We will nevertheless bring some pieces of information about the *DefaultSecurityToken* class.

1410 5.4.2.5.1 The *DefaultSecurityToken* class

This class represents the base of a security token. The token is sent to the client after a successful login and keeps a track of certain useful information like token UUID or the session id or the user UUID among other.

1415 When a user browses from one page to another one and when an authentication check is needed the only value that is transmitted to the Authentication Web Service is the token UUID. If the last is unknown by the system then the access to the page is denied and the user is invited to authenticate through the login form. If the token is recognized as valid, then the timeout value is updated and the updated token resent. More information is available in the javadoc of the class.

1420 5.4.3 TranslationManager Web Service

The present service is a non functional prototype, only for test and connection purpose. Actually it only receives a file and sends it back as is, without any translation.

The present subproject will no more be discussed because it is beyond the scope of the present diploma work. Indeed, The TranslationManager Web Service only exists because the StorageManager will be connected to such translation service. This Web Service is a lightweight proposition to a future Translation Service and has no other pretention.

To conclude, below you find the UML diagram of the project. It is only for intellectual purposes.

5.4.3.1 UML schema

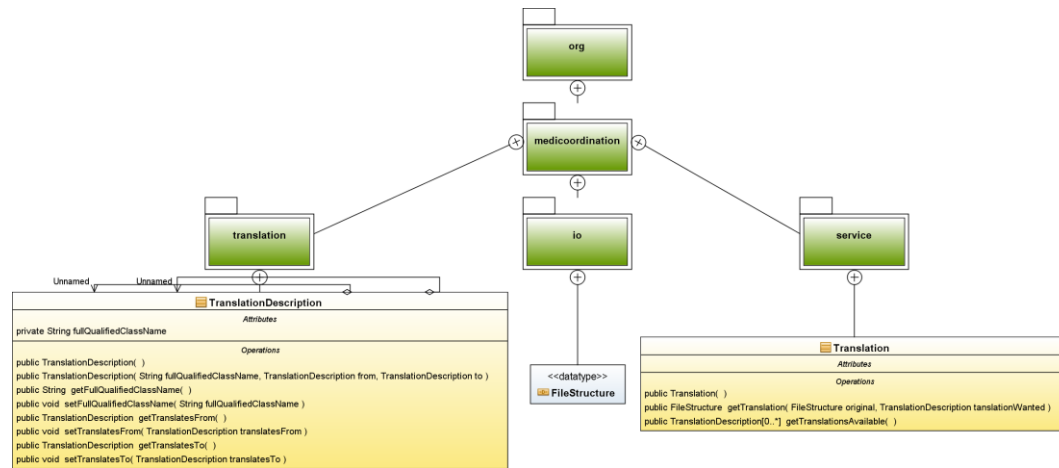


Figure 37 The TranslationManager project (UML diagram)

5.4.4 StorageManager Web Service

In the chapter 5.3.3 The storage interface implementation, we spoke in means of storage implementation. In this chapter we will discuss about the components that compose the StorageManager Web Service and the links between it and other services.

The UML diagram in Figure 27, StorageManager UML , gives an idea of the internal complexity of the storage manager, but it is only a part of the problem. In fact the StorageManager Web Service is also connected to other Web Services. These services provide functionalities to the StorageManager Web Service and are part of the global solution and are mandatory in order to make storage service to work properly.

Because the implementation part of the diploma work consists only on the realization of a repository totally hardware and software independent, the TranslationManager, the SecurityManager and the MediCoordination front-end can be considered as an extra work accomplished to give a better overview of a possible final solution.

During the next chapter we will present the most important classes of the project but we will skip the *MySQLRepository* class and all classes related to it because we will speak about in the 5.4.4.2 MySQLRepository chapter.

5.4.4.1 Description of the functionalities

1450 The StorageManager provides a very small set of functionalities to the end user. Those functionalities are listed in Figure 38, The functionalities provided by the StorageManager Web Service.


<<interface>>	
 IStorageManager	
Attributes	
Operations	
<pre> public FileStructure getFragment(String tokenUUID, String uri, DocumentFormat documentFormat, String version) public String[] getVersions(String tokenUUID, String uri) public boolean setFragment(String tokenUUID, String uri, String filename, String mimetype, DocumentFormat fileFormat, byte data[0..*]) </pre>	

Figure 38 The functionalities provided by the StorageManager Web Service

1455 The methods do not need other explanations because their signature are enough talkative to avoid more comments. We will only explain why the tokenUUID is used by every method.

In the chapter 5.3.2.3 Web Methods access restrictions in page 55, we discussed about Web Methods security. To deny Web Methods access to non authorized users, we need to receive a *tokenUUID* that corresponds to a valid authenticated session. Otherwise the function will throw an *AccessDeniedException* exception.

During the analyze phase we have define the minimal subset of functionalities provided by the storage service. A supplementary functionality, the *updateFragment*, was defined that is not present in the implemented solution.

1465 This lack is not an oversight, but rather the result of the use of a versionable repository. Indeed, when we add a new file to an existing node, in the repository, the new node is considered as the current node and the others as versions of the same node. Then, a file update consists in changing a file and then commits the changes saving it. The old file is replaced with the new one. The repository will do the same but the old version is not definitely lost. We can have it back again by choosing the version of the file we want to get. That is why the *getFragment* Web Method takes three parameters one of which is the file version to get.

5.4.4.2 MySQLRepository

1475 The MySQLRepository is a driver that controls a repository using the Jackrabbit API. This class is in charge to create the repository, if it does not exist, and control the files that goes into it.

1480 This class implements the IRepository interface. So, we can select in the *MySQLStorageManager* class the repository driver we want to use to pilot the repository. If the driver is loaded dynamically at startup, then, we will not have to redeploy the Web Service every time a bug is discovered. We just have to update the driver class and restart the GlassFish server.

The next diagram shows the dependencies of the *MySQLRepository* class. As we can see the class contains the methods to put and get files as well as a function to get the versions of a given file. The last is particularly interesting. Medical worker will be able to get any version of a file stored.

1485 5.4.4.2.1 UML Diagram

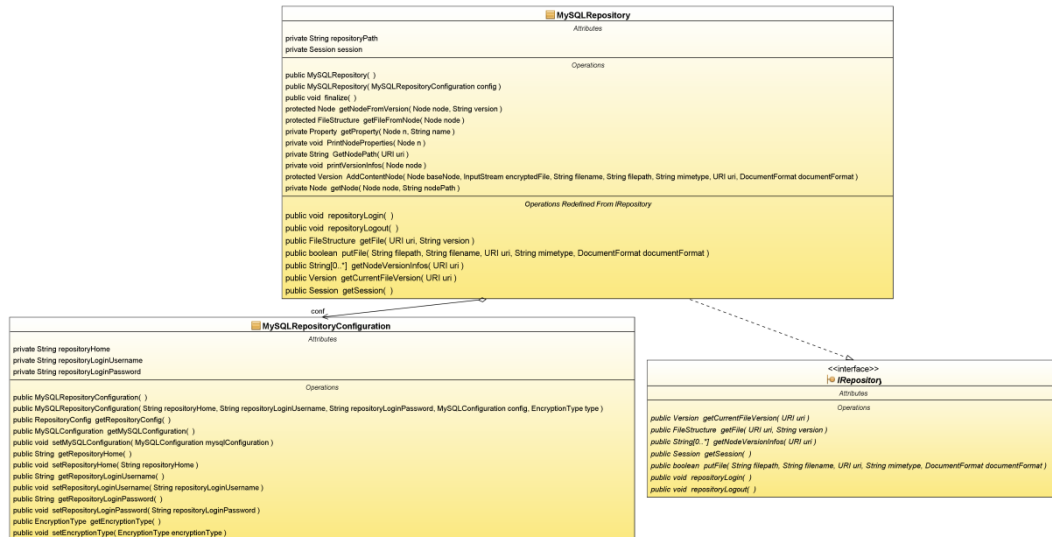


Figure 39 UML diagram of the dependencies of the *MySQLRepository* class

At current moment some changes will be made on this class. Indeed, actually only one connection can get or put files. If two users try to put files at same time, then the first will lock the repository and the second will receive an error explaining that the repository seems to be already locked. This behavior is normal within Jackrabbit.

Indeed, Jackrabbit allows creating multiple sessions, independents from each one but this is only true if the JCR repository that creates the sessions is the same. As soon as a new instance of the repository is created, then, an error is thrown to warn the user that the repository is already locked by another session.

The StorageManager-Test project, it can be found in the CD accompanying this report in the sources folder, brings the response to both problems. This project brings a solution to solve the problem of multiple concurrent runs of the same code. This solution allows multiple concurrent runs of the same code and by the way will make possible concurrent puts or gets even if multiple JCR repositories are running in parallel.

To allow concurrent accesses to the repository, this class will create only one instance of the Repository object in the JCR API. We use the singleton pattern to get always the same instance. While this instance is working, all login tries, inside the same JVM instance, will get the same repository object. Otherwise, if a repository already exists, but in another instance of a JVM, then a new instance is created and will wait until the first removes the locks. As soon as locks are removed then the second instance starts

responding to requests. The second instance will find the moment the first release the locks by pooling regularly if the repository is available or not.

5.4.5 MediCoordination front-end web application

1510 The MediCoordination front-end represents the visible part of the project. It's the
1515 human interface that allows a health worker to get or put files into the pEHR of a
patient. It allows testing the underlying services, such as the StorageManager and the
SecurityManager.

The front-end consists in a certain number of JSP pages that are linked together via
1515 session beans and with a protected access. The `./Login` page is the only entrance to the
system. Without a valid authentication the access to the services is denied. The next
diagram shows the sitemap of the MediCoordination Web Application and the links
between pages. It shows, also, that any try to access any other page without a valid
authentication will results in an *AccessDeniedException*.

1520 5.4.5.1 Sitemap Diagram

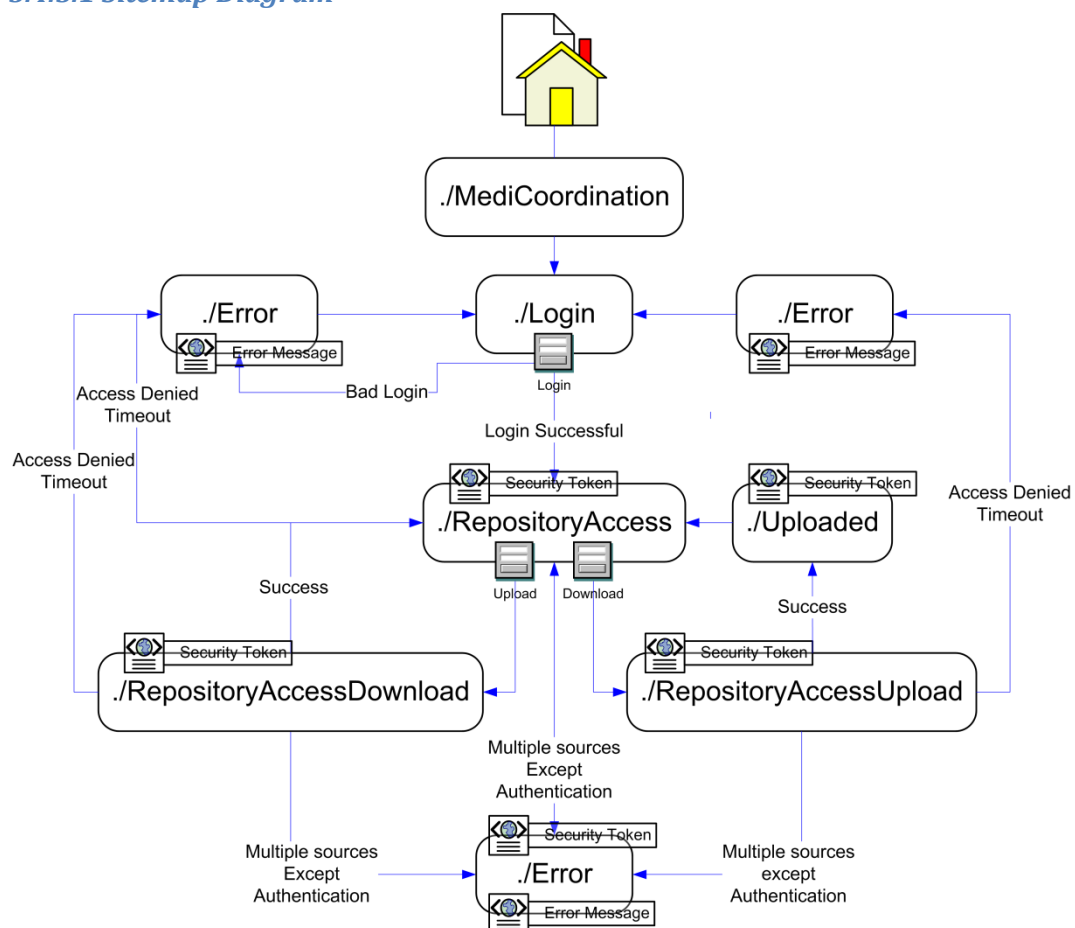


Figure 40 Sitemap of the MediCoordination front-end

5.4.5.2 Description of the functionalities

1525 The MediCoordination front-end only provides a login page and a transfer page. This JSP page allows getting or putting files into the repository.

5.5 Implementation problems and solutions

Just like other projects, this one brought lots of surprises when implementing it. The major difficulty was to develop an application without having the debugger working.

1530 This little detail made all the difference.

In fact, when implementing a solution, the debugger can accelerate very quickly the development phase, it helps finding conception errors or unexpected behaviors. Developing without a debugger is much like trying to find the way in a maze at night without flashlight. To find and solve a bug we are in darkness, only some very esoteric messages sent by the JVM can point approximately to the place where the bug stands. The smallest error can turn into an unsolvable problem. This is only one problem, over lots of others, encountered during the development phase that we had to surmount.

In the next points we will discuss about other development problems and the solutions found to solve them.

1540 5.5.1 Web Services

The first big problem encountered was related with the Web Services security. Some very strange behaviors of the NetBeans IDE, related with SSL secured Web Services, will be discussed in this section.

5.5.1.1 Security and authentication

1545 One of the biggest challenges during the project was to secure all the accesses to the Web Services. Things as simple as how to setup a server based authentication to a page could take very long hours especially when we are not used with Web Services security.

5.5.1.1.1 Context

1550 We have a page that must be protected by a password, by example a configuration page, but we do not want to use the security system used to protect Web Methods.

5.5.1.1.2 Problem encountered

The problem encountered was how to configure the Web Service to require authentication from the user that want to access to a particular page.

5.5.1.1.3 Solution

1555 GlassFish proposes some authentication mechanisms to secure accesses to resources. The authentication starts with configuration of users inside the server.

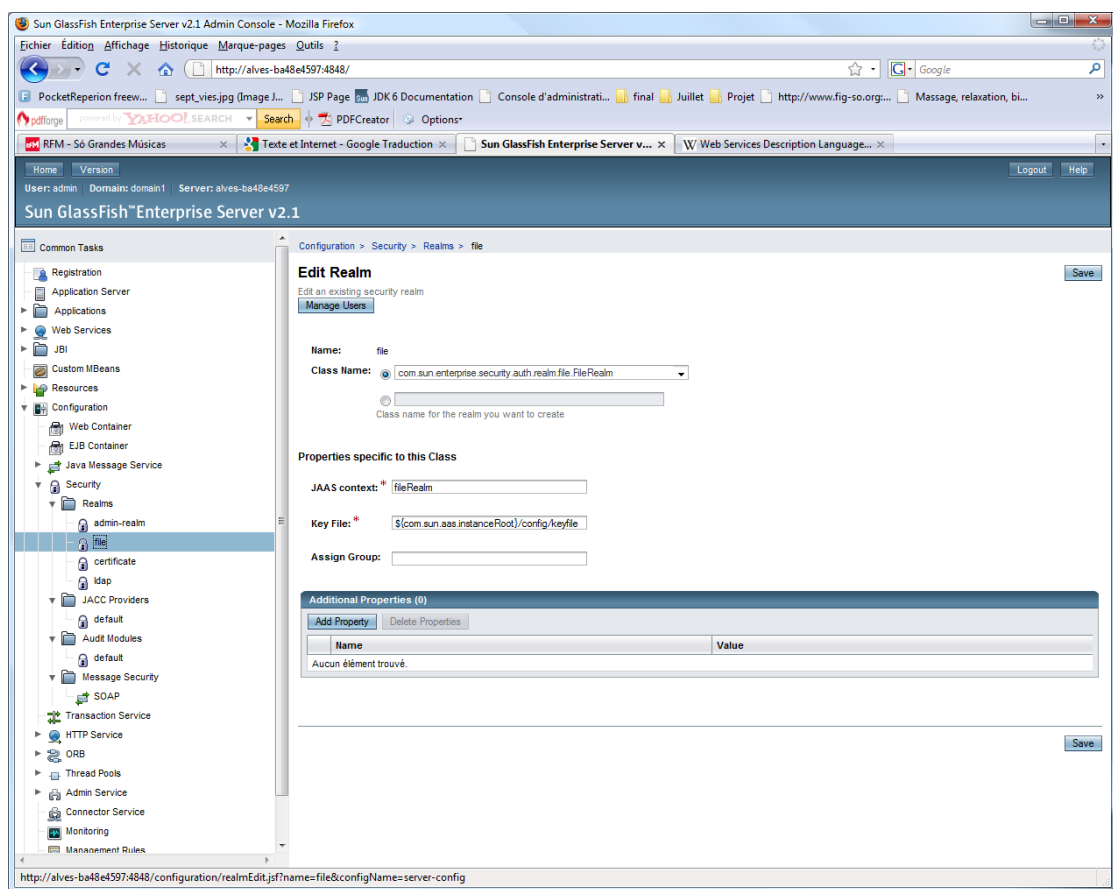
The screenshot, Figure 41, Server authentication mechanism setup, shows in *Configuration > Security > Realms* all kind of authentication mechanisms supported by GlassFish. Other authentication mechanisms can be added. The user will only need to

- 1560 develop his own realm (C.f. Creating a Custom Realm⁴¹) and add it to GlassFish and finally configure the *Realms* category to add the new authentication method.

Because, in the real world, very little people hold their own certificate, the simplest way to setup an authentication is by asking for a username and password. GlassFish allows such authentication by using the file realm.

1565 *5.5.1.1.3.1 Add new user credentials to the GlassFish server*

The first step is to add the username and password of the user that will be authorized to access to the protected page. We invite the reader, if he is not used with user addition within GlassFish, to examine the chapter 5.2.3.5.2 Creation of the GlassFish user.



1570 **Figure 41 Server authentication mechanism setup**

Once a user exists, then we need to make some changes in the *web.xml* and *sun-web.xml*, the application server configuration file⁴², to enable user authentication via the authentication mechanism of your GlassFish server.

⁴¹ <http://docs.sun.com/app/docs/doc/819-3672/beabo?a=view>

⁴² <http://docs.sun.com/app/docs/doc/819-3660/beaqj?a=view>

5.5.1.1.3.2 web.xml configuration

- 1575 In the Web Service you want to setup authentication, select the web.xml configuration file and go to the *Security* tab. Because in the server we created the user in the file realm, it means that the username, password are stored in a simple text file, we have to indicate that in the *web.xml*. Then, in the *Login Configuration* folder we need to indication the realm we want to use to the authentication and how the authentication must happens. The standard way is to request the web browser to display a little box with two fields, username and password. To do so, we need to select the *Basic* login mechanism.
- 1580

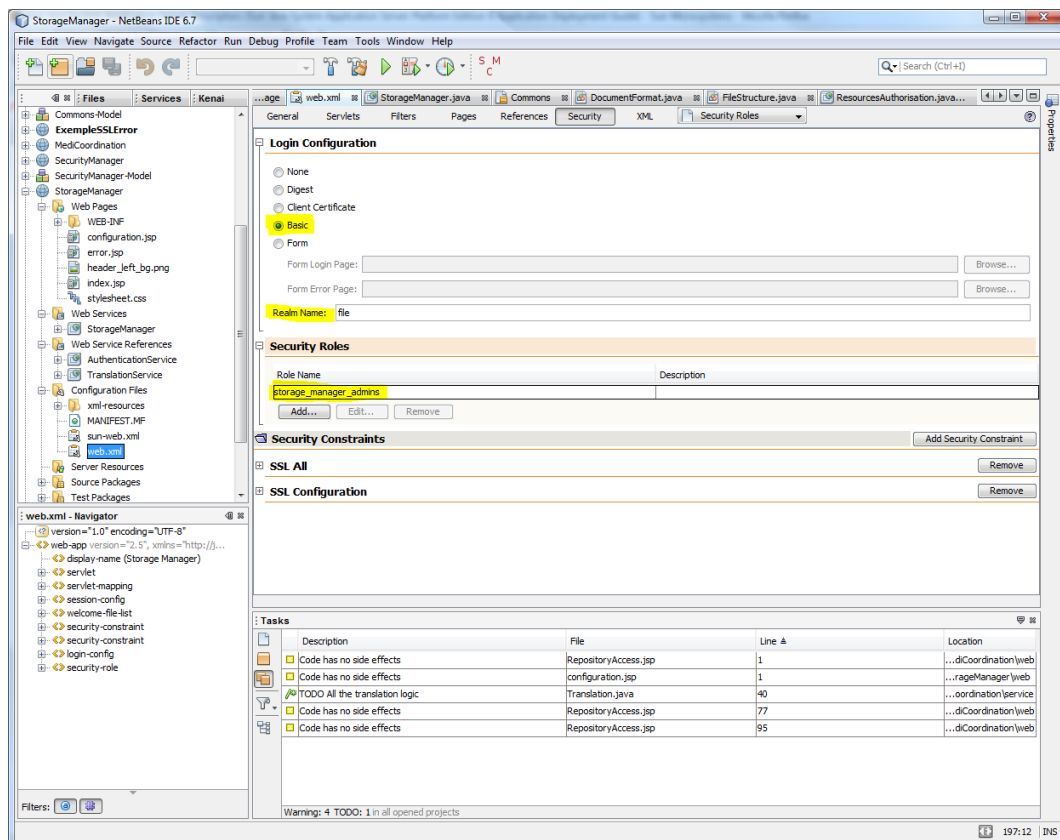
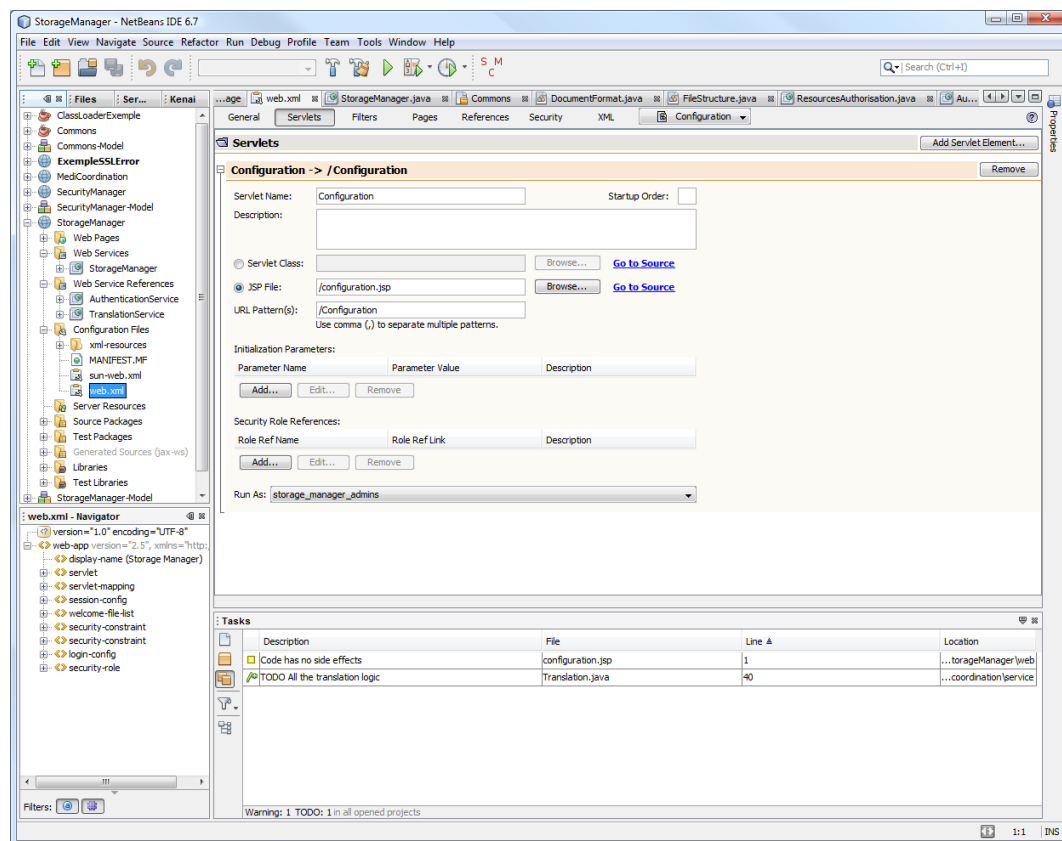


Figure 42 Servlet login configuration

- 1585 The next step is to create a security role name. A security role corresponds to an abstract logical grouping of users that will potentially use the application. In our case, the only user we want to give access to the configuration page is the administrator of the Web Service. Then, in the Security Roles section we define a new role **storage_manager_admins**. This role groups all the administrators of the Web Service, only them can, after a successful login, access the configuration page of the service.
- 1590

Still in the *web.xml* configuration file, we need to specify who can view the configuration.jsp file. We select, in the *Run as* listbox, the security role created previously, the **storage_manager_admins** security role.

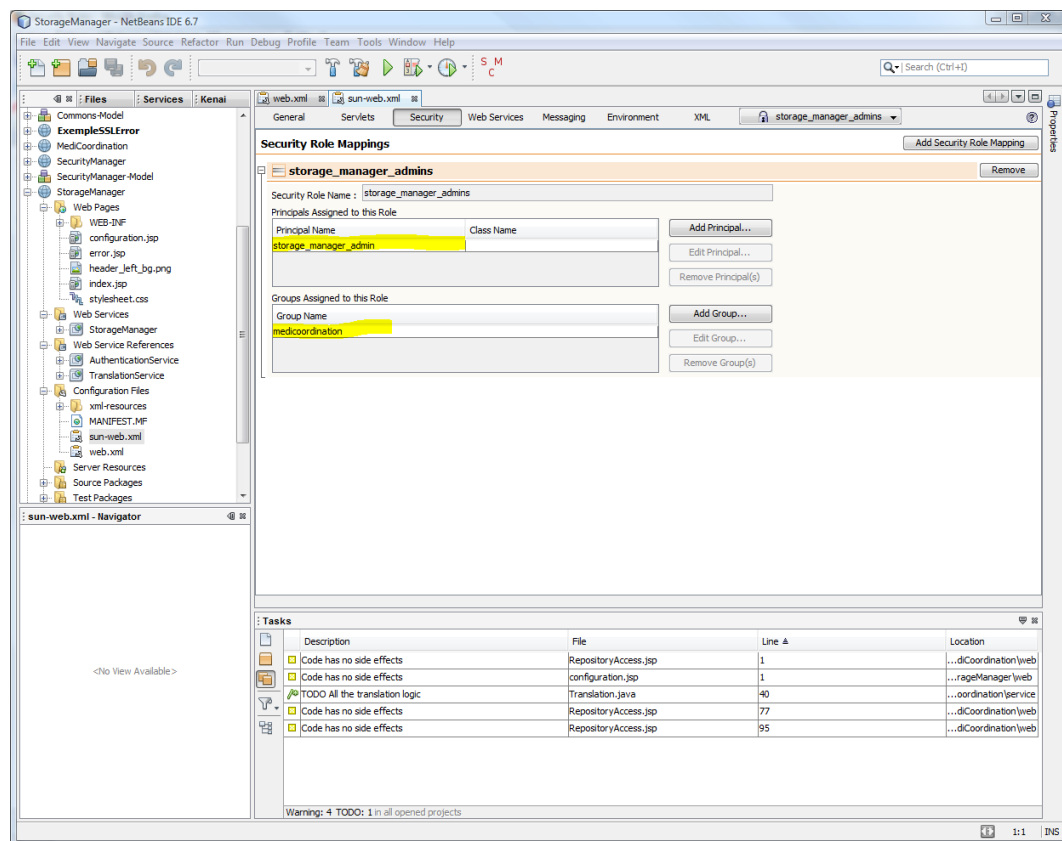


1595 Figure 43 web.xml Servlets authentication configuration

5.5.1.1.3.3 sun-web.xml configuration

To finish the configuration of the authentication, we need to map the security role to the principal we created in the GlassFish file realm. We can map the security role to some principals or to principals and groups or only to groups.

- 1600 Only the principals or groups mapped to the security role **storage_manager_admins** will be authorized to login into the configuration page. The Figure 44, Security Role Mappings, shows the configuration that you should have, at the end, to have the authentication mechanism correctly setup.



1605 **Figure 44 Security Role Mappings**

Now the authentication is correctly setup. You can deploy and test. An input box asking for the username and password will be displayed. If you have the accreditations to access the page, it will be displayed.

1610 To know who is connected to the JSP page you can call the *getUserPrincipal()*. The last method will return null if no authentication lead to a successful login.

```

192 // Tests if the user is authenticated
193 if (
194     (request != null) &&
195     (request.getUserPrincipal() != null) &&
196     (request.getUserPrincipal().getName().equals("storage_manager_admin"))
197 )
198 {
199     if (DebugMode.DEBUG_MODE) {
200         System.err.println("131 - Starting printing the page");
201     }

```

Figure 45 Sample code to display the connected principal

5.5.1.2 SSL Implementation

1615 During development, the implementation of secured communication channels between Web Services and the client was a very hard problem to solve and took lots of time. The problem is related with the creation of a Web Service client in the NetBeans IDE.

5.5.1.2.1 Context

We have a Web Service secured via SSL. We want to create a client to access that Web Service within NetBeans.

1620 5.5.1.2.2 Problem encountered

When creating the Web Service client, NetBeans throws an exception.

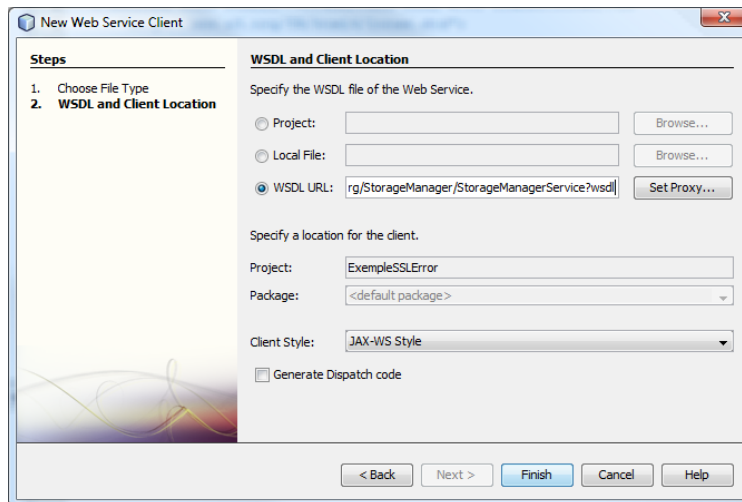
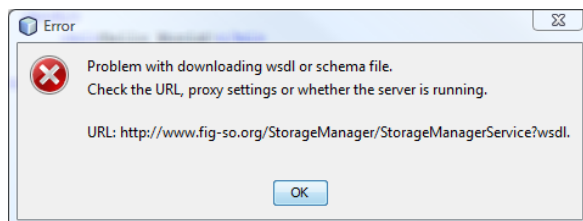


Figure 46 Web Service client creation

When creating a new Web Service client, the above window will be displayed. There are three manners to create the client.

- 1625
1. Picking the Web Service project in the project group. Just click the browse button and select the Web Service you want to automatically generate the Web Service Client
 - 1630 2. Getting the local Web Services Description Language (WSDL) of the Web Service. The WSDL must be previously generated. Right click on the Web Service in the Solution exploration and then click on menu *Generate Local WSDL*.
 3. Getting the WSDL of the deployed Web Service.

When creating the client with the third proposition we receive the error displayed below.



1635

Figure 47 Exception thrown by NetBeans

5.5.1.2.3 Causes and solutions

When installing the GlassFish server, the last will create a certificate that will serve to authenticate it every time a client asks for an SSL connection. On that certificate figures the *Common Name* - Figure 49 Web Site Certificate viewer window. When we create a Web Service client for a Web Service secured with a SSL connection, we have to provide in the WSDL URL field of the *New Web Service Client* window, Figure 46 Web Service client creation, the same *Common Name* as in the certificate otherwise NetBeans will raise an error.

```
30 juil. 2009 18:05:02 : Retrieving Location: https://192.168.1.2:8181/StorageManager/StorageManagerService?wsdl
Error: An I/O error occurred. Software caused connection abort: socket write error
```

Figure 48 Error report from NetBeans

Even if `https://192.168.1.2:8181` is the same server as `https://alves-ba48e4597:8181` the first variant raises the error while the second no.

Then, the right URL to create the StorageManager Web Service client is given below:

`http://alves-ba48e4597:8080/StorageManager/StorageManagerService?wsdl`

or

`https://alves-ba48e4597:8181/StorageManager/StorageManagerService?wsdl`

With one of above URLs and after clicking on finish button of the Figure 46 window, the window below will be displayed. Just click yes to accept the certificate. After that NetBeans will parse the wsdl and generate all the classes to communicate with the Web Service.

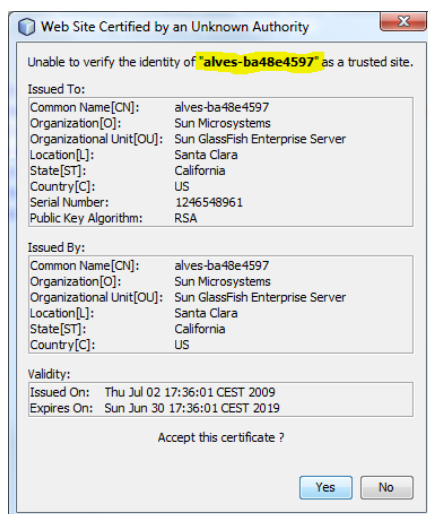


Figure 49 Web Site Certificate viewer window

- 1660 To conclude, it is really important, when creating a Web Service client, to provide the same server name as in the certificate, otherwise NetBeans will refuse to load the wsdl file, and by the same, it will refuse to generate all the classes needed to invoke the Web Service.

5.5.2 Apache Jackrabbit

- 1665 The second most important source of problems involves Apache Jackrabbit and GlassFish. Even if it is a relatively easy to understand API, some concepts are badly explained in the documentation and barely discussed on the internet.

Such a lack of documentation resulted in some sleepless nights trying to find a solution to the given problem.

5.5.2.1 Configuration

In standard working mode, Jackrabbit is very easy to setup and put working. Below we have the minimal code to setup and startup a valid Jackrabbit repository.

```
import javax.jcr.Repository;
import javax.jcr.Session;
import org.apache.jackrabbit.core.TransientRepository;

/**
 * First hop example. Logs in to a content repository and prints a
 * status message.
 */
public class FirstHop {

    /**
     * The main entry point of the example application.
     *
     * @param args command line arguments (ignored)
     * @throws Exception if an error occurs
     */
    public static void main(String[] args) throws Exception {
        Repository repository = new TransientRepository();
        Session session = repository.login();
        try {
            String user = session.getUserID();
            String name = repository.getDescriptor(Repository.REP_NAME_DESC);
            System.out.println(
                "Logged in as " + user + " to a " + name + " repository.");
        } finally {
            session.logout();
        }
    }
}
```

43

1675 **Figure 50** Logging in to Jackrabbit

Things seem rather easy, but, as soon as we try to change the standard working mode, repository in a directory of the hard disk, to set it working with MySQL then things became more complex.

- 1680 We have to edit the default jackrabbit configuration file and define the persistence manager to use with Jackrabbit, the database file system, setup the workspace configuration, the security configuration, the repository configuration and the versioning

⁴³ <http://jackrabbit.apache.org/first-hops.html>

manager. A single error on one entry and it is the whole system that would not work properly. One can find, on Apache Jackrabbit home site, a *How to*⁴⁴ that helps to setup the repository and the default *repository.xml* file can be found on Jackrabbit home site⁴⁵.

1685 5.5.2.1.1 Solution

After lots of hours on internet, we found a standard Jackrabbit configuration file. We incorporated it in the *MySQLRepositoryConfiguration* class. Configuration fields needed to connect properly to a database are dynamically replaced with data from *MySQLConfiguration* class and with repository login data provided by the *MySQLRepositoryConfiguration* class itself. Finally, we only have to call the *getRepositoryConfiguration* method inside the class and a *RepositoryConfig* object correctly configured is generated and returned. We pass it to the repository constructor and the repository is correctly instantiated.

5.5.2.2 Sessions and Locks (Concurrent accesses to the repository)

1695 We already discuss, in a previous chapter, about the problem of multiple sessions running on the same repository. As soon as multiple sessions are created within the same repository object, everything work perfectly and we can have as much sessions as we want without errors, except if two session try to access the same node. Adding **mix:lockable** mixin type to the node will allow the node to be lockable. Then we can use lock mechanisms provide by the Node class of the JCR API to avoid concurrent accesses.

But the problem is much complicated. Indeed, creating a static Repository object, in the StorageManager Web Service, that multiple connections can access would be a solution. But the scope of a static object is the JVM. If we start twice the web service then the static field will not be the same in both JVM because JVM's do not share their memory.

1705 Consequently what will happen is that the first instance of the Repository object created will lock the repository database and only no other connections from other JVM's will be accepted.

5.5.2.2.1 Solution

1710 A solution consists to limit the session existence to the strict minimum. We login just before saving the file and logout session directly after it finishes. When no more sessions are open inside the Repository instance locks are released and another Repository waiting for the locks release will be able to start to respond to requests.

If new requests arrive to the first instance, then they must wait until all locks from the second instance are released.

⁴⁴ <http://jackrabbit.apache.org/jackrabbit-configuration.html>

⁴⁵ <http://jackrabbit.apache.org/jackrabbit-configuration.data/repository.xml>

1715 This solution has been tested with the project StorageManager-Test project. If running normally 5 Threads are created and concurrent accesses managed without problem. All files are stored and at the end locks are released.

Now to test that multiple JVM's running the same application can access concurrently to the repository, just push some times on F6 within NetBeans IDE to start a new run of the application. We can see observe that all instance will write files to repository without errors.

1720 An Instance running in a separate JVM will pool, checking the state of the repository. As soon as the repository is free then the repository acquires the Lock for the repository and start working.

1725 5.5.2.3 Node Versioning & Locking

Jackrabbit supports versioning. To setup it, we have to make some changes to the repository configuration file.

5.6.2.3.1 Configuration of the repository

1730 Firstly we have to add a new section, if not done already. The example below shows the section to add to the *repository.xml* file.

```
<versioning rootPath="${rep.home}/version">
  <filesystem .../>
  <persistenceManager .../>
  <ISMLocking .../> <!-- optional, available since 1.4 -->
</versioning>

<versioning rootPath="${rep.home}/version">
  <filesystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
    <param name="path" value="${rep.home}/version"/>
  </filesystem>

  <persistenceManager class="org.apache.jackrabbit.core.persistence.bundle.DerbyPersistenceManager">
    <param name="url" value="jdbc:derby:${rep.home}/version/db;create=true"/>
    <param name="schemaobjectPrefix" value="version_"/>
  </persistenceManager>
</versioning>
```

Figure 51 Versioning entry in the repository.xml configuration file

1735 The first versioning tag is only to see the main configuration entries of the versioning configuration. The second tag contains only the file system and persistence manager tags. The file system tag indicates where to store the different file version while the persistence manager specifies where to store Meta information about versions.

It is possible to set the file system to one of the three below⁴⁶:

1. LocalFileSystem, data is stored in hard disk.
2. DatabaseFileSystem, data is stored in a database, to be used with a database manager as repository and workspace file system.

1740

⁴⁶ <http://jackrabbit.apache.org/jackrabbit-configuration.html#JackrabbitConfiguration-FileSystemconfiguration> (File System configuration)

3. `MemoryFileSystem`, data is stored in RAM. All data is lost as soon as the repository is closed. The file system is intended to tests only or small read-only workspaces.

1745 Once the versioning is set in the configuration file of Jackrabbit, we have very little operations to do before using it.

5.5.2.3.2 How to make a node versionable `mix:versionable`

1750 Within Jackrabbit, every node can be versionable. We have to add the node type *Mixin* to the node we want to make versionable. The **`Node.addMixin ("mix:versionable")`** instruction will indicate to the Jackrabbit repository that the node and children and all properties have to be versioned.

The function **`Node.checkin ()`** returns the last version of the node while to add a new version to it will imply to make a **`Node.checkout ()`** to make it writable, then to write the data and to do a **`Node.checkin ()`** to make it read-only.

1755 Jackrabbit provides several classes, in the *javax.jcr.version* package, to lead with versioning. It will be possible to restore old versions of a node, to remove versions, to save space, to get a precise version and much more functionalities.

5.5.2.3.3 `mix:lockable`

1760 To set a node lockable is sensitively the same as for the versioning. We just have to add the type `mix:lockable` to the node with the method **`Node.addMixin("mix:lockable")`**. Then we just have to use the Lock methods embedded in the Node object to control who can access the node.

5.6 Improvements

1765 Even if great efforts have been made to create a piece of software that responds to most of the requirements edited in the chapter 2, some of them could not be completed, mainly because the time was not enough to do everything we planned to do.

The next chapters will describe the future improvements that could be made on StorageManager Web Service. We discussed about the strategy pattern and the fact that storing the full qualified name of classes into the configuration file will allow to dynamically changes the behavior of the storage service.

1770 5.6.1 Dynamic loading integration

The integration of such mechanism inside the Web Service and in the manager class to allow dynamic loading of the repositories implementation will be a plus. It will especially facilitate the development, because one big problem with web services implementation is that deployment takes too much time.

1775 With this mechanism, no need to redeploy. We just need to update the class and reload the page.

5.6.2 Jackrabbit session management

The second improvement that must be taken is related with the Apache Jackrabbit repository.

- 1780 A solution must be found to load only one instance of the repository accessible by the Storage Service. This instance will have the same life cycle as the server. It would starts with the domain starts and stops with the domain stops.

If no solution is found, the concurrent accesses to the repository will always represent an obstacle to create a reliable piece of software.

1785

6 Storage Manager Compatibility with IHE recommendations

For the present project, IHE recommendations are not available or are only partially applicable.

6.1 Storage compatibility

1790 IHE don't advocate the use of a specified technology to implement the storage module.
Therefore we are totally free to choose the implementation that we will judge the better.

IHE just recommends that the system sustaining the file storage would allow putting, getting, updating files with a versioning system integrated with it, that's the case of
1795 Apache Jackrabbit for the three propositions we made above.

6.2 File exchange (XDS)⁴⁷ compatibility

IHE recommends XDS (Cross-Enterprise Document Sharing). IHE only recommends that files must be stored in a repository in a reliable and secure manner. The accesses, permissions and how patients authorize medical workers to access their files are also
1800 described inside this chapter but are not useful for the storage management service itself because the Storage Management service will delegate such requests to a third party web service specialized on such actions.

6.3 Service security compatibility

Concerning web services security, IHE recommendations are more precise and we have
1805 following them to choose and to implement our solution.

The most important recommendation, in terms of service security, concerns the Web Methods access security. They recommend restricting accesses to Web Services to non authorized actors. This recommendation has been integrated in the Storage Manager Web Service through the Security Manager Web Service and Security tokens.

1810 The other concerns communications channels. This second recommendation has been integrated in the service too. Indeed all communication channels are SSL protected.

6.3.1 User authentication (EUA)⁴⁸ compatibility

As for XSD the user authentication is not directly linked to this module. Effectively we will use authentication in order to allow or refuse web service access but the work to
1815 query server, the kind of server to use and all considerations about this precise action is deported to the web service that will do authentication.

⁴⁷ IHE IT Infrastructure Technical Framework, vol. 1 (ITI TF-1) : Integration Profiles (§10 page 64)

⁴⁸ IHE IT Infrastructure Technical Framework, vol. 1 (ITI TF-1) : Integration Profiles (§4 page 27)

7 Conclusion

1820 The present document describes the implementation of the storage service for electronic medical content. The current project was intended to be a prototype that could possibly open research ways to the final Storage Manager in the MediCoordination project. Most requirements have been fulfilled successfully but some other need more time in order to solve some problems that impeach the implemented solution to run efficiently.

1825 The choice of Jackrabbit seemed to be a good idea but in order to confirm that technologic choice a battery of test must be done. Time becoming less and less, all the testing part of the project had to be abandoned to concentrate on the most important task, creating a running prototype.

1830 In addition to the Storage Manager Web Service, two other services have been partially developed and can, therefore be used as starting point to future implementations. These services have been developed because they are closely integrated with Storage Manager Web Service. Without them it would be very tricky to create a real functional prototype.

8 Bibliography

- Alves, B., & Schumacher, M. I. (2009). *Deliverable D3.1A: Interoperability Architecture*.
1835 Sierre: Institut d'informatique de gestion.
- Barik, T. (2005, August 23). *Introducing the Java content repository API*. Retrieved June 05, 2009, from IBM.com: <http://www.ibm.com/developerworks/java/library/j-jcr/>
- Bragg, R. (2009). *The Encrypting File System*. Retrieved May 26, 2009, from [technet.microsoft.com](http://technet.microsoft.com/fr-fr/library/cc700811(en-us).aspx): [http://technet.microsoft.com/fr-fr/library/cc700811\(en-us\).aspx](http://technet.microsoft.com/fr-fr/library/cc700811(en-us).aspx)
- 1840 Carr, C., & Davis, D. (2008, December 12). Integrating the Healthcare Enterprise. *IHE IT Infrastructure (ITI) Technical Framework Volume 1 : Integration Profiles*. IL, USA.
- Dai, W. (2009, March 31). *Speed Comparison of Popular Crypto Algorithms*. Retrieved May 26, 2009, from Cryptopp.com: <http://www.cryptopp.com/benchmarks.html>
- Day Management AG. (2005). *JCR v1.0 Specification HTML version*. Retrieved June 1,
1845 2009, from day.com: <http://www.day.com/specs/jcr/1.0/>
- Day Software. (2005, may). *The java Community Process(SM) Program - JSRs : Java Specification Requests - detail JSR# 170*. Retrieved june 10, 2009, from jcp.org: <http://jcp.org/en/jsr/detail?id=170>
- Day Software. (2009, march). *The java Community Process(SM) Program - JSRs : Java Specification Requests - detail JSR# 283*. Retrieved june 10, 2009, from jcp.org: The java
1850 Community Process(SM) Program - JSRs : Java Specification Requests - detail JSR# 170
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (2005). Strategy. In E. Gamma, R. Helm, R. Johnson, & J. Vlissides, *Design Patterns Elements of Reusable Object-Oriented Software* (pp. 315-323). Westford, Massachusetts: Addison-Wesley.
- 1855 IHE.net. (2008, November 28). *Cross-Enterprise Document Sharing (XSD) - IHE Wiki*. Retrieved June 5, 2009, from [wiki.ihe.net](http://wiki.ihe.net/index.php?title=Cross_Enterprise_Document_Sharing): http://wiki.ihe.net/index.php?title=Cross_Enterprise_Document_Sharing
- IIG - HES-SO // Valais. (2009, May 04). *MediCoordination*. Consulté le June 15, 2009, sur [MediCoordination](http://www.medicoordination.ch/): <http://www.medicoordination.ch/>
- 1860 Metro. (2009, April 12). *metro: Metro Guide - Configuring SSL and Authorized Users*. Retrieved July 2, 2009, from [metro.dev.java.net](http://metro.dev.java.net/guide/Configuring_SSL_and_Authorized_Users.html#gfxzy): http://metro.dev.java.net/guide/Configuring_SSL_and_Authorized_Users.html#gfxzy
- Pospisil, J., Kuchtiak, M., & Wielenga, G. (n.d.). *Asynchronous JAX-WS Web Service Client End-to-End Scenario*. Retrieved July 12, 2009, from [Netbeans.org](http://www.netbeans.org/kb/55/websvc-jax-ws-async.html): <http://www.netbeans.org/kb/55/websvc-jax-ws-async.html>
1865

Seitz, L., Pierson, J.-M., & Brunie, L. (2005). *Encrypted storage of medical data on a grid*. Lyon: January.

Seitz, L., Pierson, J.-M., & Brunie, L. (2005, January). Encrypted Storage of Medical Data on a Grid. (S. GmbH, Ed.) *Methods of Information in Medicine* , 2.

1870 Shinder, D. (2006, August 23). *Implementing EFS in a windows Server 2003 Domain*. Retrieved May 27, 2009, from WindowsSecurity.com:
<http://www.windowsecurity.com/articles/Implementing-EFS-Windows-Server-2003-Domain.html>

1875 Sun Microsystems, Inc. (2002, August 4). *Java Cryptography Architecture*. Retrieved June 12, 2009, from java.sun.com:
<http://java.sun.com/j2se/1.4.2/docs/guide/security/CryptoSpec.html>

Sun Microsystems, Inc. (2002, January 10). *Java Cryptography Extension*. Retrieved June 12, 2009, from java.sun.com:
<http://java.sun.com/j2se/1.4.2/docs/guide/security/jce/JCERefGuide.html>

1880