DLNA media server implemented in an OSGi environment supporting online content

LUNDS
UNIVERSITET
Lunds Tekniska Högskola

LTH School of Engineering at Campus Helsingborg
Department of computer science

Bachelor thesis: Tobias Lundquist

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

Printed in Sweden
Media-Tryck
Biblioteksdirektionen
Lunds universitet
Lund <20100126>

Abstract

This paper describes the development of a DLNA digital media server. The server runs on top of an OSGi stack on a residential gateway. The server supports adding and removal of content sources at runtime.

The problems with handling updates from diffrent media sources are analyzed and discussed. Also soulutions to these problems are also proposed.

The diffrent techniques for the implementation are described and evaluated.

Keywords: OSGi, Media server, DLNA, UPnP

Sammanfattning
Denna rapport beskriver utvecklingen av en DLNA mediaserver. Servern körs på en OSGi-stack som körs på en hemgateway. Servern har möjlighet att lägga till och ta bort källor medan den körs.

Problemen med att hantera material från olika mediakällor analyseras och diskuteras. Ett lösningsförslag lämnas även på dessa problem.

De olika teknikerna användna för detta projekt finns beskrivna samt utvärderade.

Nyckelord: OSGi, Mediaserver, DLNA, UPnP

Foreword

Both the development and research in this degree project have been very educational. Both skills from past courses and newly learned skills have come in handy. The development has taken place at the Digital Home department at TeliaSonera in Malmö. Me and my tutor, Johan Holmberg, have had many interesting and educational technical discussions regarding implementation, design and structure of the media server.

I surely hope that my work done here, in my degree project, will be of use for TeliaSonera.

Last but not least I would like to thank everybody at the Digital Home department at TeliaSonera in Malmö especially Johan and Per for all the support and feedback during my degree project.

List of contents

1 Introduction 1
1.1 Background 1
1.2 Goal 1
1.3 Methods and work flow 1
1.4 The report structure 2
2 Techniques used for implementation 2
2.1 OSGi alliance and the OSGi Framework 2
2.2 DLNA 5
2.3 WebDAV 5
2.4 XML 6
2.5 Cache 6
3 The DMS structure 6
3.1 Analysis of existing code 6
3.2 First refactorization 8
3.3 From monolithic to modular 9
4 Implementation details 9
4.1 File Cache 9
4.2 Content Directory service, CDS 10
4.3 Configurations 12
5 Evaluation of implementation 12
5.1 File Cache 12
5.2 Content Directory service, CDS 12
5.2.1 Content updating 14
5.2.2 Limitations 14
5.2.3 Prototype 14
5.2.4 Data structure 14
5.3 Configuration 15
6 Learnings done and recommendations 15
6.1 Java runtime 15
6.2 Developing Java for embedded environments 15
6.2.1 Loops 16
6.2.2 Avoid evasive creation of objects 16
6.2.3 Getters and setters 17
6.3 OSGi 17
6.4 DLNA 17
7 Final words 18
8 Dictionary 19
9 References 20

1 Introduction

1.1 Background

TeliaSonera is the largest telecom operator in Sweden. They provide a wide customer base with access to Internet, telephony, mobile telephony.

Today it's common to store and access media, such as pictures, music and videos, at external online services, but few[1] media servers are able to access and aggregate those external services and their content and present them and provide access for local media players.

To give the customers a simple and efficient way to access content that they have stored online TeliaSonera looks into integrating a media server with this functionality in the gateway.

1.2 Goal

The goal of this degree project is to develop a DLNA DMS (Digital Media Server) that is able to aggregate content from various online services and present it for DLNA DMP (Digital Media Players). The DMS shall be run on top of an OSGi-stack and allow sources to be added and removed during runtime.

The DMS shall be lightweight enough to be run on an RGW (Residential GateWay).

1.3 Methods and work flow

The work started of with an existing media server implementation. The code and the structure of the implementation were analyzed and evaluated. Parts of the existing code could be reused but also some parts needed to be rewritten. A new structure was also heavily needed since it was impossible to modularize the existing one.

The modular design that evolved during the work is the effort of hours with whiteboards, pens and discussions with my tutor, Johan Holmberg. The new modular design lead to a framework which allowed for new content sources to be added and implemented with very little effort.

1 Intervew with Johan Holmberg 20091010

1.4 The report structure

It is assumed that the reader has basic computer science knowledge and understand the concepts behind object oriented programming and design. Further is fundamental knowledge of computer networks needed.

The report starts of with descriptions of different techniques used for this implementation. These techniques are later evaluated and some recommendations are made. The way from the monolithic to modular media server is described. At the end of the report evaluation of the implementation is made.

2 Techniques used for implementation

2.1 OSGi alliance and the OSGi Framework

The OSGi alliance, formerly known as the Open Services Gateway initiative, founded in 1999 is an alliance of technology innovators that advance a process to assure interoperability of applications and services based on the OSGi framework[2].

The OSGi framework provides a platform based on Java to allow applications to be installed, removed and updated without the need of a reboot of the platform. Applications are formed as bundles that are normal jar-files with a manifest with framework specific headers.

Illustration 1[3] shows the diffrent layers of the OSGi framework.

2 http://www.osgi.org/About/HomePage (20091204)

3 Graphics from: http://en.wikipedia.org/wiki/File:Osgi-system-layering.svg (20091203) licenced under Creative Commons Attribution ShareAlike 3.0; http://creativecommons.org/licenses/by-sa/3.0/legalcode (20091203)

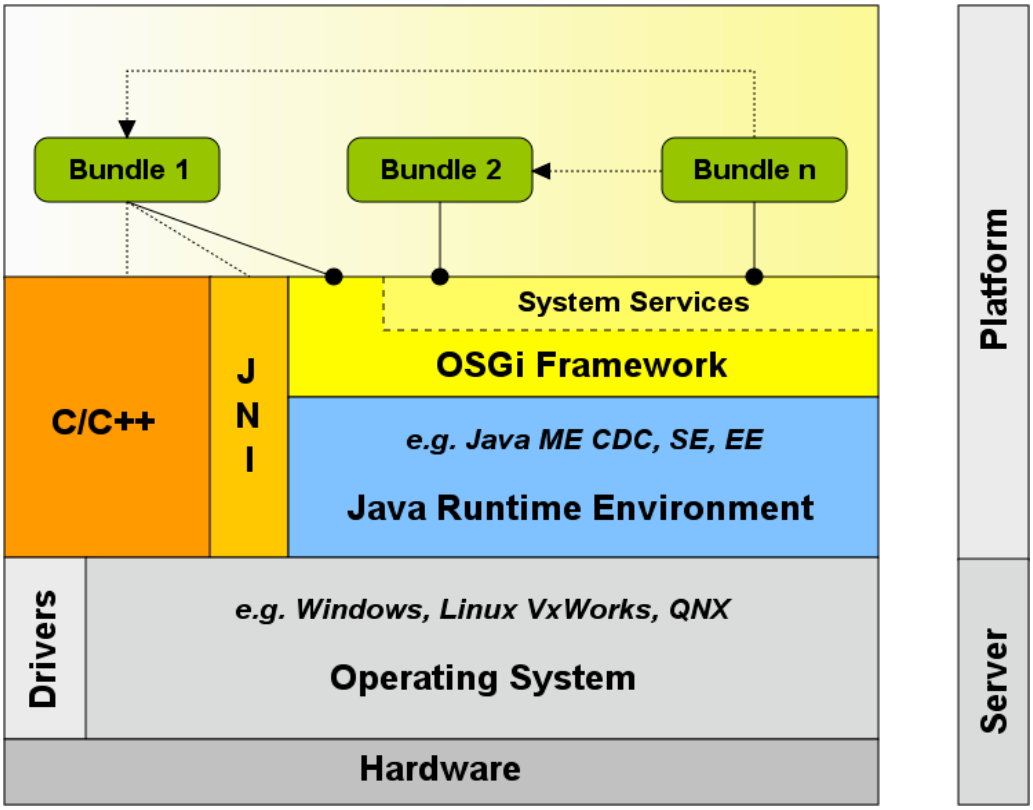


Illustration 1: The different layers of the OSGi framework

The OSGi framework provides an environment for letting larger applications being divided into smaller modules, in this case bundles. The bundles can be specified with dependencies and are allowed to be updated, removed and installed in runtime. A bundle has a life cycle with six different states[4]: Installed, resolved, starting, active, stopping and uninstalled as shown in illustration 2[5].

4 http://en.wikipedia.org/wiki/Osgi#Life-Cycle (20091202)

5 Graphics from: http://en.wikipedia.org/wiki/File:OSGi_Bundle_Life-Cycle.svg (20091203) distributed as public domain; http://en.wikipedia.org/wiki/Public_domain (20091203)

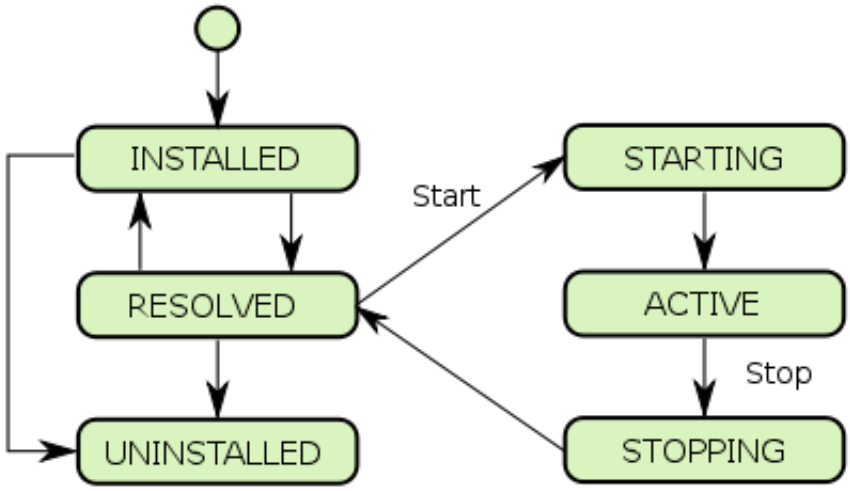


Illustration 2: The life cycle of an OSGi bundle

State	Description
Installed	The bundle is loaded into the framework.
Resolved	All dependencies for the bundled are resolved.
Starting	The framework waits for the bundle's Start method to return.
Active	The bundle's Start method has returned and the bundle is running.
Stopping	The stop method in the bundle is running and the framework waits for it to return.
Uninstalled	The stop method has returned. It's not possible to start this bundles again (without reinstalling it).

2.2 DLNA

DLNA (Digital living network alliance) is a standard that allows DLNA-compliant equipment to share media in a local home network. The standard is adopted by many manufacturers of consumer electronics. Today DLNA has more than 245 members[6].

6 http://www.dlna.org/about_us/about

Devices that are DLNA certified are supposed to be able share with and manage content at each other. The standard also specifies how to control one unit from another one.

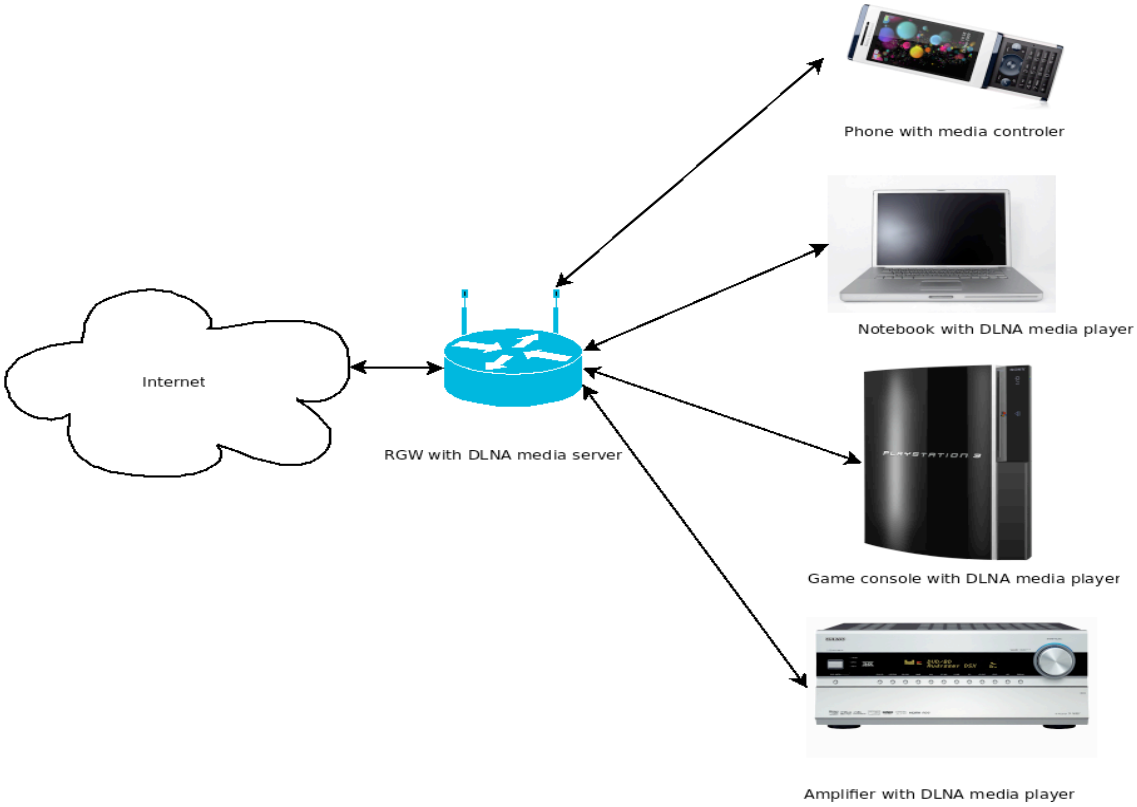


Illustration 3: DLNA

DLNA is based on UPnP and communication between units is encoded in XML. In transport of content (music, images or video) from the server to clients HTTP is used.

2.3 WebDAV

Web-based Distributed Authoring and Versioning, or WebDAV, is an extension of the HTTP protocol which allows for users to manage remote files. The protocol is specified in RFC4918[7]. The protocol is backwards compatible with HTTP. Thus requests formed for HTTP are accepted as WebDAV requests.

WebDAV adds a lot of features to HTTP such as locking, properties, name space management and collections. This makes it possible for users to upload

7 http://tools.ietf.org/html/rfc4918

files to a WebDAV-server which makes the web writable. This is all in line with Tim Berners-Lee's original vision of how the web should work[8].

TeliaSonera provides an online storage service, *Säker lagring*[9], based on WebDAV. This service is referred to as SL, or sl, in the code and parts of the report. There are other WebDAV based online storage services available from other suppliers such as Storegate[10], I[(2)]drive[11] and myDisk[12]. The media server will be compatible with these services as long as they provide a WebDAV access.

2.4 XML

XML, or Extensible Markup Language, is a standard for encoding documents. It's specified in the *XML 1.0 Specification*[13]. It's used in DLNA to send information between units.

2.5 Cache

A fairly wide term in computer science, mostly a technique use to speed up the access to data. It's implemented as storing a copy of data on a faster medium. I.e. if it's a disk cache, a copy of the disk block is stored in RAM.

3 The DMS structure

3.1 Analysis of existing code

The work started with an existing code base containing a DMS and a WebDAV connection. This implementation had several flaws and some major weaknesses. In the beginning the code didn’t even compile due to syntax errors and missing imports. The documentation was obsolete and not too well written and made it hard to really understand what was missing and what went wrong.

To get the DMS to start in the OSGi environment lots of code reading, investigations and debugging was needed. Several bugs were corrected and minor modifications in the structure was made. To get the DMS to present the content to the DMP:s the outputed DLNA XML needed to be corrected.

8 http://www.w3.org/1998/02/Potential.html (20091207)
9 http://www.telia.se/privat/produkter_tjanster/internet/tjanster/sakerhetstjanster/teliasakerlagring.page (20091207)
10 http://www.storegate.se/ (20091204)
11 http://www.i2drive.com/ (20091204)
12 https://mydisk.se/web/main.php?show=home (20091204)
13 http://www.w3.org/TR/2008/REC-xml-20081126/ (20091101)

Further, the code base was a rather monolithic blob and was not well suited for being modularized due to illogical splitting into the different packages.
To achieve a modular structure and allowing new sources to be added a major refactorization was required.

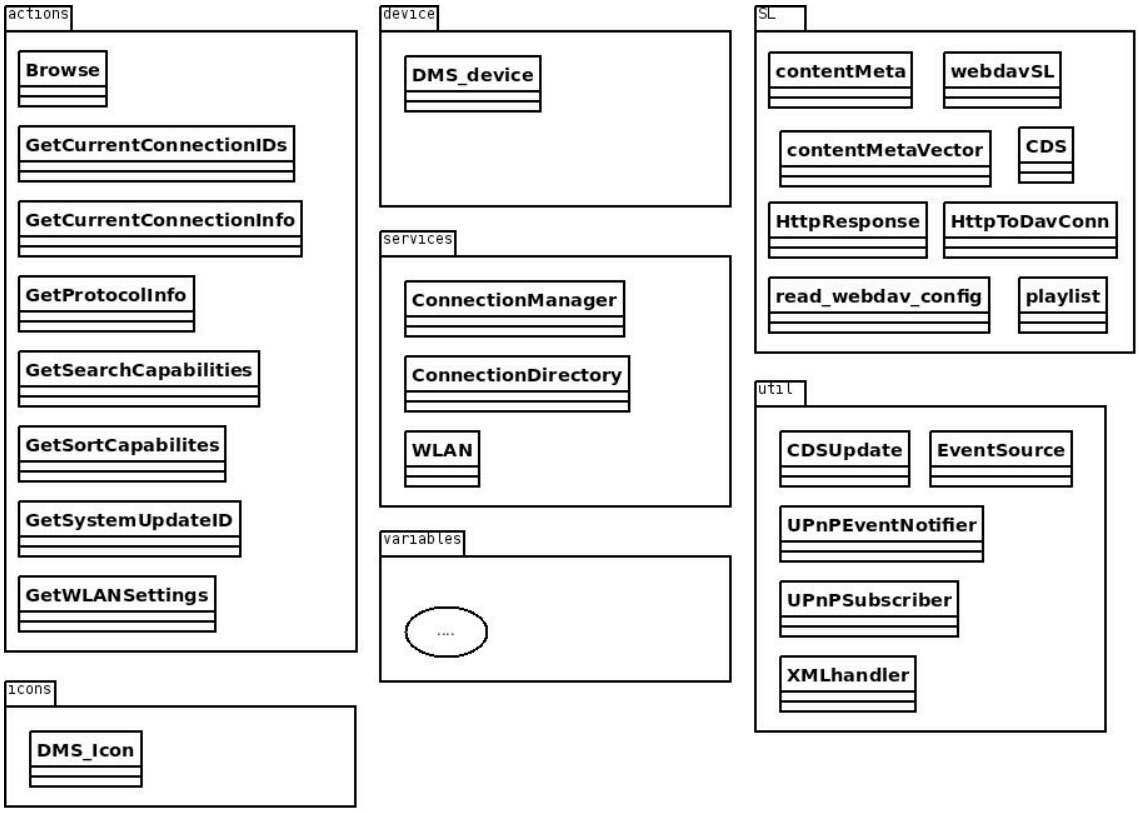


Illustration 4: Structure before the refactorization

As shown in the diagram above, it is impossible to break out the SL-package because the CDS, *Content Directory Service*, that handles available content will be lost in the ”main” package. Also, is this really the right location for this class? It's got nothing to do with the connection to SL - *Säker lagring*.

Also *HttpResponse* and *HttpToDavCon*, which handles http responses and incoming http requests will be lost in the ”main” package. Once again, is this really the right location for these classes?

3.2 First refactorization

To solve some of the illogical placements of classes and to make a more modular structure I introduced two new packages, *httpserver* and *content*. Some of the moved classes were also renamed to something more logical.

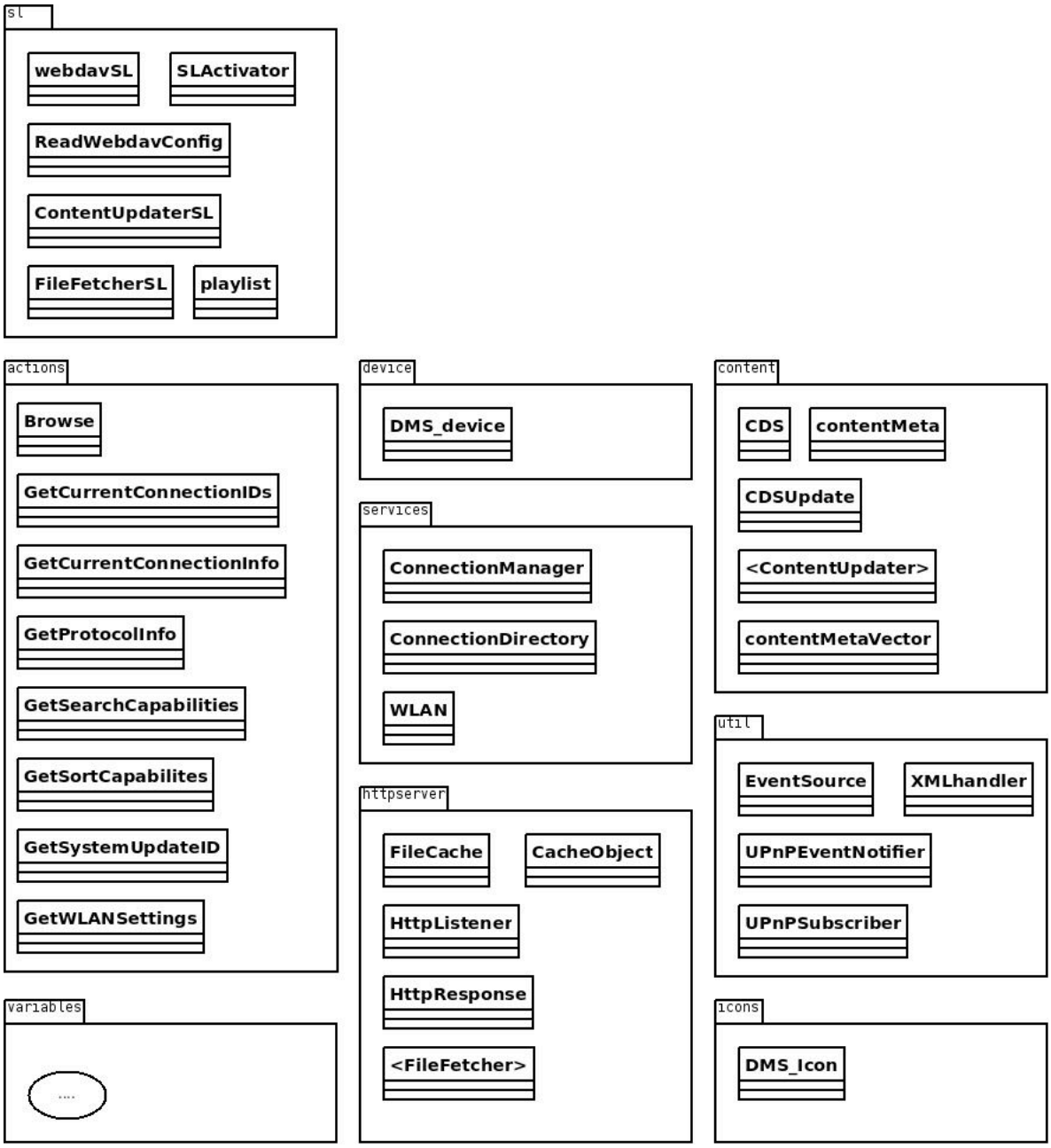


Illustration 5: Structure after first refactorization

Also two new interfaces were introduced, *ContentUpdater* and *FileFetcher*. This is because that's all that differs between different kinds of sources:

- How to get a list of what content is available at the resource
- How to get the actual file at the resource

The new *FileCache* (and *CacheObject*) is used to speed up access on content already accessed. This is explained in detail later in the report. As the diagram shows, it's now possible to break out *sl* and make it an bundle to be loaded and unloaded from the OSGi stack. No vital parts of the *DMS* will be lost.

3.3 From monolithic to modular

The first refactorization made it possible to turn sl into a bundle instead of being a part of the DMS bundle. Some small changes in the sl package were made to make it a bundle, these were all OSGi specific. To allow content from different sources to be updated with different intervals the content now is pushed from the bundle to the *CDSUpdater*.

4 Implementation details

4.1 File Cache

The cache in this project is implemented as an LFU, *Least Frequently Used* cache. This is a rather simple cache algorithm which discards the least frequently used item stored[14]. When there is a request for a file, we check if there is an entry for that file in the cache, if so the cached file is returned. If the file requested isn't in the cache, the file is fetched and stored in the cache and returned to the requester.

When a file is stored in the cache it gets an initial *weight* or *hit count*. This weight is decreased with a specified periodicity until it reaches a minimum weight. Every time a cached file is accessed the weight is increased.

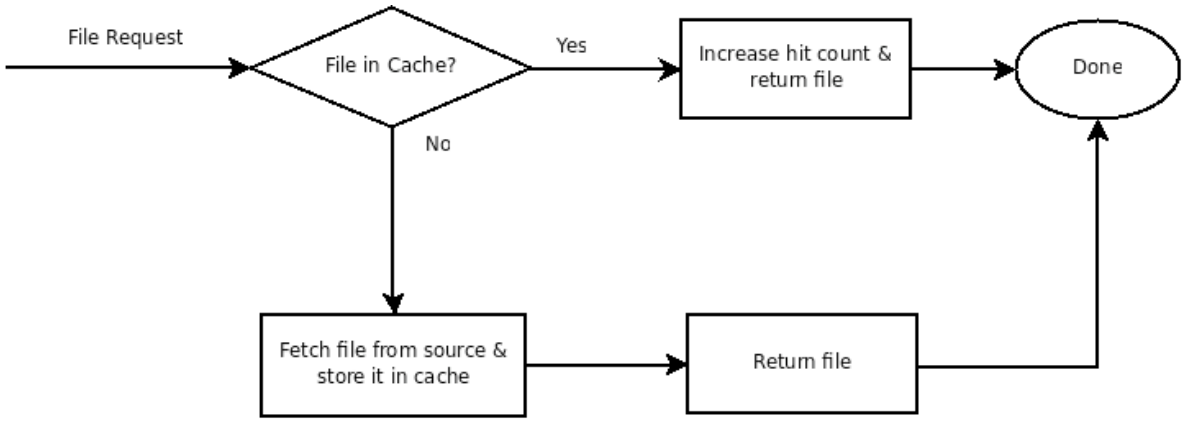


Illustration 6: Flowchart for a file request

If the cache discovers that there is not enough memory free the cache is cleaned.
The item(s) removed are the item(s) least frequently used; in other words, the item(s) with the lowest weight. If two items are equally used the first item cached is discarded.

14 Silberschatz, Galvin, Gagne (2007) Operating system concepts with Java (Wiley)

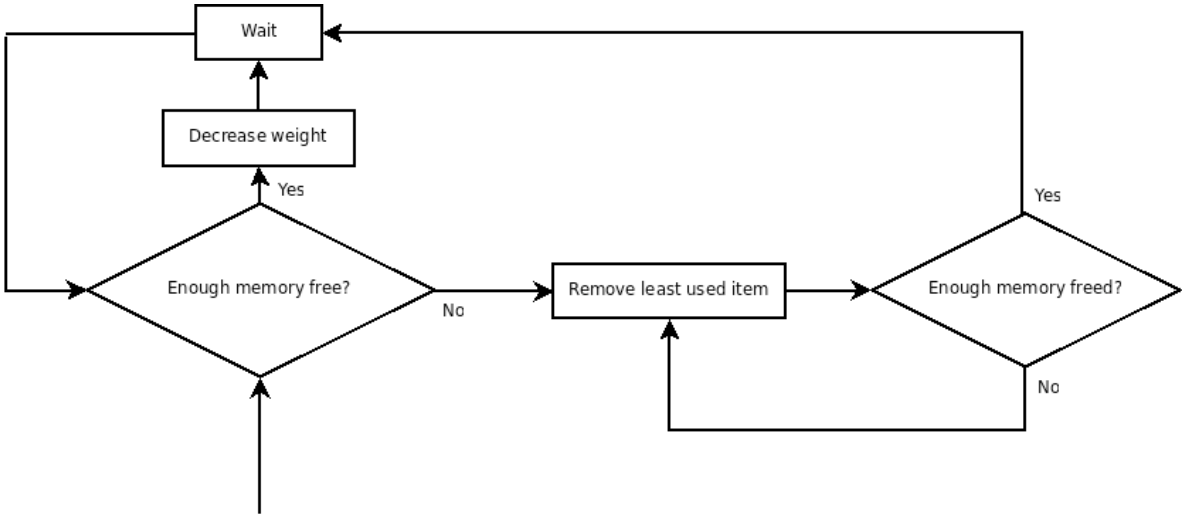


Illustration 7: The cache cleaning process

4.2 Content Directory service, CDS

The CDS keeps track of content available to media players. Today the CDS uses a vector to keep track of the content, a content vector. With only one media source providing content this is a suitable data structure. But when multiple sources were introduced this broke the structure more or less. To understand why, we need to take a look how the CDS works. A very simplified content vector is described below.

Every entry, file or directory, in the CDS has an ID, this ID is the index of the vector. An entry must also have a parent. The parent is the ID of the directory where the files is located. This parent is referred to with a ID. There is also a special case, the root directory has ID 0 and parent -1.

When a client want to browse files, it will most likely start with requesting ID 0. The server will answer with all files that has parent 0. The answer is in fact encoded in XML, the format of the XML is specified in the DLNA documentation and is beyond the scope of the report.

The user is now interested in browsing the pictures and selects the pictures folder which has ID 2. The client requests the content from directory with ID 2.
The server answers with all content that has parent 2, again the DLNA specified XML is used.

Now both files and directories are available to the client and the user. If the user is interested in the picture "a.jpg" the client will use the supplied URL to access the content. No more requests to the CDS are made.
But if the user is interested in browsing the "pets" directory a new request is made to the CDS. The procedure is the same as for access to the "pictures" directory[15].

Content Vector:

Index	Parent	Path	URL
0	-1	/	
1	0	/music/	
2	0	/pictures/	
3	1	/music/1.mp3	http://host/path/file
5	1	/music/2.mp3	http://host/path/file
6	2	/pictures/a.jpg	http://host/path/file
8	1	/music/artist/	
9	8	/music/artist/a.mp3	http://host/path/file
10	8	/music/artist/b.mp3	http://host/path/file
11	2	/pictures/a.jpg	http://host/path/file
12	2	/pictures/pets/	
13	12	/pictures/pets/dog.jpg	http://host/path/file
14	12	/pictures/pets/dog2.jpg	http://host/path/file
15	12	/pictures/pets/cat.jpg	http://host/path/file
16	8	/music/artist/c.mp3	http://host/path/file
17	12	/pictures/pets/ferret.jpg	http://host/path/file

The preexisting implementation always passed a new content vector if the content at the source was changed. This is neither acceptable nor suitable for this data structure when we are handling content from multiple sources, i.e. if one source changes the content all other sources needs to be updated. This consumes CPU time and memory and is therefore not suitable for embedded environments. Even if the media server were to be run at a more power full platform this implementation is questionable since it takes time to scan a source for content, build a file list and merge all content from the source.

15 http://www.upnp.org/standardizeddcps/documents/ContentDirectory1.0.pdf (20091123)

One solution is to only pass new entries and entries to remove. Adding files is not an issue but removal introduced a problem. If the file with ID 10, /music/artist/b.mp3, was removed from the content vector all files with ID > 10 is going to have a new ID, index – 1. This will, unconditionally, make all parent references with ID > 10 obsolete. Also removal of elements in the middle of a vector is rather slow. This is more of an issue when the vector grows large.

One method would be to mark content as removed and let it stay in the content vector. But if the changes are big and many changes are made, the content vector will grow very large since all new content will be added. This will, in the end, consume large amounts of memory. Again, this is not suitable for an embedded environment.

Since a new implementation is, more or less, required a solution is discussed later in the report.

4.3 Configurations

The existing implementation used a configuration file with XML to handle the configuration. It might be hard for the target users to edit this file to update the configuration. To solve this a configuration web page was introduced.

5 Evaluation of implementation

5.1 File Cache

The cache implemented works fairly well but needs heavy testing at the target platform. This to avoid evasive memory usage and evaluate how much performance gain there is in real-world applications. Also modifications of files at the sources are not registered in the CDS and therefore not in the cache. To solve this a modification time needs to be introduced in the CDS and the cache.

5.2 Content Directory service, CDS

To solve problems with handling content from multiple different sources I propose a new implementation of the CDS. Instead of having just one content vector, use one vector for each source and use another vector to keep track of the different sources.

Source ID (SID) is the ID of the source and corresponds to index of the vector that keeps track of the different sources, the Source Vector (SV). Each entry in

the SV points to a Source Content Vector (SCV). The Source Content Vector contains all content at a specific source. The Source Content Vector is more or less the same as the previous described content vector.

Instead of having a rather flat addressing as in the previous implementation we now must now divide the ID or address in to different address spaces.
The last digit of the address is always 0. This is used for addressing the Source Vector. The second last digit is the address to the source and the rest of the address is the address (index in the SV) of the entry. The whole address is called Full ID (FID).

SV

SID	0	1	2	3	4	5	6	7	8	9
Name	Flickr	SL1	SL2							

SCV

Index	Parent	Path
0	-1	/
1	0	/a
2	0	/b
3	1	/a/a.jpg
4	1	/a/b.jpg
5	2	/b/c.jpg
6	2	/b/d/jpg
7	2	/b/e.jpg
8	1	/a/f.jpg
9	1	/a/g.jpg

SCV

Index	Parent	Path
0	-1	/
1	0	/a
2	0	/b
3	1	/a/a.mp3
4	1	/a/b.mp3
5	2	/b/c.mp3
6	2	/b/d.mp3
7	2	/b/e.mp3
8	1	/a/f.mp3
9	1	/a/g.mp3

SCV

Index	Parent	Path
0	-1	/
1	0	/a
2	0	/b
3	1	/a/a.mp3
4	1	/a/a/jpg
5	2	/b/a.mp3
6	2	/b/b/mp3
7	2	/b/c.jpg
8	1	/a/b.jpg
9	1	/a/e.mp3

I.e. The FID 500 is referring to the file "/b/c.jpg" in the red vector and the FID 510 is referring to the file "/b/c.mp3" in the blue vector.

The address space illustrated:

ID/Source Content Vector Index	SID	0

The reason to address the SV with 0 at all times is to simplify the request for the root folder (always ID 0).

5.2.1 Content updating

In this implementation proposal the ContentUpdater in each media source is responsible for generating and updating a Source Content Vector and passing it to the CDS. All the CDS has to do when a media source has changed is to move the reference from the old Source Content Vector to the new one. This should be a lot faster and more resource efficient than fetching and updating all sources and merge the content into the CDS every time a single source has been updated.

5.2.2 Limitations

Since the maximum value of an int in Java is 2,147,483,647[16] the maximum number of files at each source is 2,147,483. The maximum number of sources that can be used, at the same time, is ten since we are using the second last digit to address the source. This can however be extended to be one hundred sources if we reserve the third last and second last digits to represent the source. If this is done we are limited to 214 748 files and folders at each source. This is, in my opinion, not much of an issue in either case. 2,147,483 photos or music tracks with an average size of 4MB requriers about 8 TB of storage space. If we use ten sources, that's about 80 TB. As of today, that is *alot* of storage space.

5.2.3 Prototype

There is a prototype CDS with this functionality implemented. This needs testing and likely minor bug fixes. To fully use this implementation the code that keeps track of the content at the different sources needs to be updated. These changes are rather small and should be easy to make.

Another new feature in the prototype is the support for content searching. Currently there is support for searching by content name and by content type. This can, with very little effort, be extended to other search criteria.

5.2.4 Data structure

The reason to uses vectors as the data structures in the CDS is to keep it simple, fast and efficient. Access to elements will be O(1) and the time complexity to build the Source Content Vector will be O(n). A hash map implementation is not suitable because it makes it hard to map parents and children together. This is simple with vectors since this maping is based on vector indexes.
Of course would a database be preferred, but the target platforms lacks this possibility and therefor is not a database solution discussed.

16 http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html (20091204)

5.3 Configuration

To solve the configuration issue a web page for the configuration handling was introduced. This is implemented as a servlet running at the OSGi stack's web server. Instead of storing configurations in a XML-file a Json-based configuration file is used.

The reason to replace XML with Json is that it is easier to parse[17].

6 Learnings done and recommendations

6.1 Java runtime

To support online content external API:s needs to be used. This is one of the cornerstones of the media server. Most of these, if not all, API:s requires a Java 5 compliant Java Virtual Machine (JVM) to run. Today the target platform runs a Java 1.4.1 compliant Java Virtual Machine. This needs to be updated if external sources are going to be supported.

It is also highly recommended to use a more up to date Virtual Machine from a security point of view. Java 1.4.1 has been marked as End Of Life (EOL) since October 2006 and End Of Service Life (EOSL) since October 2008[18]. Therefore no more security updates for these platforms will be released.

6.2 Developing Java for embedded environments

When developing Java for an embedded environment it is of most importance to save recourses. CPU cycles and memory are limited and are in this case also needed by other functions in the gateway.

17 http://json.org/ (20091202)
18 http://java.sun.com/products/archive/eol.policy.html (20091125)

6.2.1 Loops

When iterating over an vector or an array it is a good idea to copy size of the vector/array to the local scope instead of looking it up every time in the loop[19].

Good	Bad

int size = vector.size();
for(int i = 0; I < size; i++){
//vector operations
}
	for(int i = 0; I < vector.size(); i++){
	//vector operations
	}

6.2.2 Avoid evasive creation of objects

To create an object is expensive and should be avoided if possible.
There will, of course, always be need for temporary objects but reuse them if possible[20].

Good	Bad

Obj temp;
int size = list.size();
for(int i = 0; i < size; i++){
temp = list.get(i);
temp.operation();
}
	for(int i = 0; i < list.size(); i++){
	Obj temp = list.get(i);
	temp.operations();
	}

19 http://developer.android.com/guide/practices/design/performance.html#cache_fields (20091204)
20 http://developer.android.com/guide/practices/design/performance.html#object_creation (20091204)

6.2.3 Getters and setters

Within a class it is not recommended to use getters and setters to access or modify a class's internal values. To directly access the values is far more efficient and therefore more suitable for an embedded environment. This saves a lot of CPU cycles and some memory[21].

Good	Bad

private int a;
private void method(){
  a = 2;
  int b = a;
}
 	
public void setA(int i){ a = i;}
public int setA(){return a;}
	private int a;
	private void method(){
	  setA(2);
	  int b = getA();
	}
	
	public void setA(int i){ a = i;}
	public int setA(){return a;}

6.3 OSGi

The use of an OSGi framework for this task is very well suited. The features facilitating loading and unloading bundles at runtime makes the task to add, update and remove content sources easy. However, the underlying framework, in this case the DMS with all of its components, must be adapted to be run on top of an OSGi stack. It is fairly simple to make these changes and the benefits are massive. I find it hard to find any drawbacks with OSGi since the performance losses are non existing or minimal.

6.4 DLNA

Today DLNA is widely spread and more and more products are supporting the protocol. It is wise choose DLNA as a communication protocol for this server. There are some drawbacks however. DLNA specifies which media formats you must support, this is a good thing, but it also specifies which media formats your are allowed to support. If you choose to support a media format that is not in the specification, you can not DLNA-certify your product.

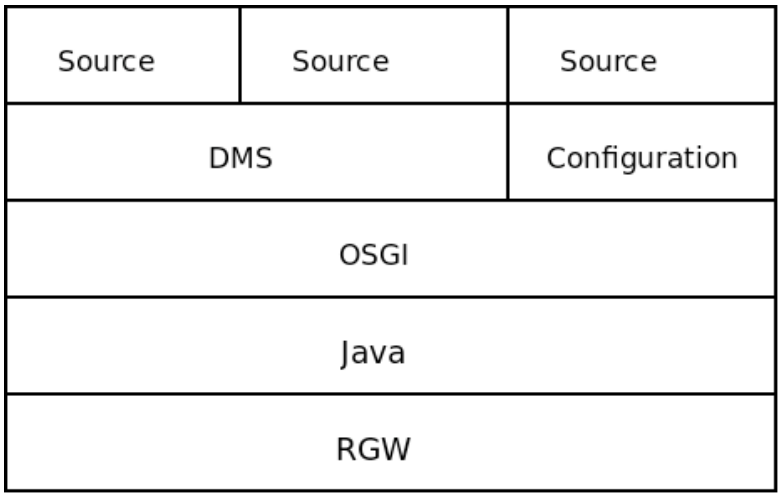
The DLNA specification is also hard to read and get an good grip of. This is due to its rather massive and not to well organized. This makes it hard to implement a DLNA product.

21 http://developer.android.com/guide/practices/design/performance.html#internal_get_set (20091204)

There is also lack of support for synchronizing output at multiple devices. I.e. If you what to play the same music track at your stereo in your living room and your radio in your kitchen there is no way to get the audio in sync.

7 Final words

The results of this degree project is a modular DLNA digital media server that runs on top of a OSGi stack. The media sources are written as OSGi bundles and it is possible to add and remove media sources in runtime.



The media server is resource efficient enough to be run at a residential gateway. To fully use the potential in this solution Java 5, or newer, is needed. This due to that almost all third party API:s requirers Java 5 or newer.

8 Dictionary

CDS	Content Directory Service. Facility that is keeping track of content in a media server.
DLNA	Digital Living Network Alliance. An alliance that is working on a standard for connected media devices.
DMC	Digital Media Controler. Device that provides the possability to control other, complant, DLNA devices.
DMP	Digital Media Player. Device that is able to play media content provided by compatible servers. Specified in the DLNA specification.
DMS	Digital Media Server. A server that provides media to other compatible units. Specified in the DLNA specification.
HTTP	Hyper Text Transport Protocol, specified in RCF2116.
OSGi	Open Service Gateway Initiative.
UpnP	Universal Plug 'n' Play. A set of protocols that simplifies connections and communications between devices in a network.
Webdav	Web-based Distributed Authoring and Versioning. A standard that extends HTTP. Specified in RFC2518.
XML	eXtensible Markup Language.

9 References

1. Intervew with Johan Holmberg (20091010)
2. http://www.osgi.org/About/HomePage (20091204)
3. Graphics from: http://en.wikipedia.org/wiki/File:Osgi-system-layering.svg (20091203) licenced under Creative Commons Attribution ShareAlike 3.0; http://creativecommons.org/licenses/by-sa/3.0/legalcode (20091203)
4. http://en.wikipedia.org/wiki/Osgi#Life-Cycle (20091202)
5. Graphics from: http://en.wikipedia.org/wiki/File:OSGi_Bundle_Life-Cycle.svg (20091203) distributed as public domain; http://en.wikipedia.org/wiki/Public_domain (20091203)
6. http://www.dlna.org/about_us/about
7. http://tools.ietf.org/html/rfc4918
8. http://www.w3.org/1998/02/Potential.html (20091207)
9. http://www.telia.se/privat/produkter_tjanster/internet/tjanster/sakerhetstjanster/teliasakerlagring.page (20091207)
10. http://www.storegate.se/ (20091204)
11. http://www.i2drive.com/ (20091204)
12. https://mydisk.se/web/main.php?show=home (20091204)
13. http://www.w3.org/TR/2008/REC-xml-20081126/ (20091101)
14. Silberschatz, Galvin, Gagne (2007) Operating system concepts with Java (Wiley)
15. http://www.upnp.org/standardizeddcps/documents/ContentDirectory1.0.pdf (20091123)
16. http://java.sun.com/docs/books/tutorial/java/nutsandbolts/datatypes.html (20091204)
17. http://json.org/ (20091202)
18. http://java.sun.com/products/archive/eol.policy.html (20091125)
19. http://developer.android.com/guide/practices/design/performance.html#cache_fields (20091204)
20. http://developer.android.com/guide/practices/design/performance.html#object_creation (20091204)
21. http://developer.android.com/guide/practices/design/performance.html#internal_get_set (20091204)