FACILITATING END-USER DEPLOYMENT OF APPLICATIONS

C.J.M.F.C. Raemaekers

Graduation Project Report

January to August 2008

Print Logic Technologies

FACILITATING END-USER DEPLOYMENT OF APPLICATIONS

version 1.2 (final)

PUBLIC SUMMARY

Technische Universiteit
Eindhoven
University of Technology

C.J.M.F.C. Raemaekers
TU/e ID: 517967
August 2008

A. Abstract

This document is the graduation project report of C.J.M.F.C. Raemaekers executed at Océ Print Logic Technologies in Créteil, France. The project consisted out of research into end-user deployment of applications in a Java-based environment and the application of it in a commercial setting regarding submitter applications for printers.

The project focused primarily on a solution based on Java Web Start technology that was found to be most suitable in practice to use with the mentioned submitter application. This report covers the entire process from the identification of the requirements, the available solutions and a practical approach towards the end product.

Note: as a result of the importance of this project within the commercial strategy of Océ Print Logic Technologies, this report is a public summary of the full report that can be consulted after explicit authorization of Océ PLT in case of need.

B. Table of contents

1. Preface

This section serves as the preface of this document describing its purpose, scope and intended audience.

1.1 Purpose of the document

The purpose of this document is to group all documentation together that applies to the graduation project executed at Océ Print Logic Technologies in Créteil, France. The documentation resulting from both the research and development component of the project should give thorough insight in the activities executed in this context.

1.2 Scope of the document

This document serves as the final documentation on the research project into end-user deployment of applications in a Java-based environment, especially focused towards a submitter application Océ.

1.3 Intended audience

The audience of this document are primarily students of the faculty of Mathematics and Computer Science of the Eindhoven University of Technology that are interested in this particular subject. This document is a public summary of the complete graduation report [GRAREP] that can be consulted after explicit authorization of Océ PLT in Créteil when there exists a specific educational need to do so.

2. Introduction

The introduction of this document describes the setting of the project, its characteristics and the nature of the project. We will also look at a brief description of the problem statement, an issue that will be addressed in depth in the following section. In the project characteristics description you can also find a detailed description of the phasing used for the project as well as the key activities that made a part of it.

2.1 Project execution period and location

As a result of great personal interest in the Île-de-France region of France and Paris in particular, this location was chosen to execute the graduation project for the Master of Science studies in computer science for the Eindhoven University of Technology (TU/e). The known, close relationships between the TU/e and the Océ headquarters in Venlo (both located in the Netherlands at just 65 km of each other) resulted into negotiations with the Océ Research and Development (Océ R&D) in Créteil, located in the Val-de-Marne department (94) in Île-de-France, just 10 km south-east of Paris.

They offered an interesting research and development project for about eight months within the application systems department on facilitating end-user deployment of applications. This project fitted perfectly within the profile followed in the Master of Science studies at the TU/e where the profile of Information Systems (IS) was chosen as the area of expertise. In particular as a result of the already indicated wish of Océ to investigate the possibilities using a web-based approach, which has been the primary orientation within the curriculum followed within the IS-group at the TU/e. With the choice for France, a strong international character could be assured, following the preparations made at the center for language and technology (CTT), also part of the TU/e.

The project started at January 3rd 2008 and ran until August 29th 2008. Eight full months were available for research, explorative prototyping and the development of a demonstrative and most feasible solution. The project was executed within a setting of in general French colleagues, but also with people from various other nationalities. French was used as primary language, but English is used within Océ R&D as the default language for official communication and documentation.

2.1.1 Company profile

Océ is a large company from Dutch origin currently focused on the production of printing devices for professional use. The origin of the company starts in 1877 when Lodewijk van der Grinten developed a coloring procedure for margarine in order to give it the same color as normal butter. In the twenties of the twentieth century two grand children of Lodewijk van der Grinten developed a special copying procedure that became known under the German name “Ohne Componente” (which literally means “Without Components”). The abbreviation (O.C.) became the basis of the later brand written phonetically as

“Océ”. Today, Océ focuses primarily on the production of printing and copying devices. Océ has about 24.000 people working for them worldwide in over 30 countries. About 2.000 people are working in the R&D department in Europe and in the USA. With this considerable amount of people working in this field, a tradition dating from the period of the Van der Grintens is kept intact. The headquarters are still located in Venlo, The Netherlands, the city where Lodewijk van der Grinten started his business over 130 years ago [CORBRO].

2.2 Project characteristics

In this section we will discuss the project characteristics by shortly describing its problem statement (a thorough discussion on this can be found in section 3) as well as the nature of the project.

2.2.1 Short problem statement

The problem statement can be summarized as finding a suitable solution for the deployment of a Java-based application having multiple Microsoft Windows based external resources (DLL’s). The original problem description received at the beginning of the project stated a series of symptoms and possible solutions to this deployment problem. In section 3 this analysis is made in depth.

Within the problem statement was included that the impact on the current (Java based) source should be as minimal as possible and that a solution was desired that also supported different user workflows of the application, notably the possibility of launching the application from a web-based environment.

2.2.2 Nature of the project

This project is largely characterized by a lot of (practical) research and explorative prototyping. There is very little existing scientific research available in the field of the deployment process. This might be the result of the fact that the deployment process within software engineering was usually taken as a necessary phase in order to get the product at the client’s place, but that had relative to the rest of software development, very little attention. This tends to change when new technologies are arising to make this process easier for both developers and end-users.

As a result of this, research and explorative prototyping was considered to be of vital importance in order to create a clear view within the possibilities available and to be able to come up with a feasible solution that might solve the initial deployment problem.

2.3 Phasing and activities

As a result of the strong explorative character of the project, the coordinators at Océ R&D decided to follow a rather loose planning scheme based on key activities and phasing. These key activities are discussed in section 2.3.2. This project is in its nature different from most graduation projects where a standard phasing can be applied. Therefore a proper phasing has been made to meet the nature of this particular project. These phases are described below. In appendix A details about the deliverables can be found.

2.3.1 Phases

Preparation phase

The preparation phase consisted out of the following activities:

- Getting to know Océ and the project group;
- Preparation of the material to be used (configuration of PC).

Product research and orientation phase

The product research and orientation phase includes the following activities:

- Investigation of the problem;
- Investigation of the current program;
- Research into possible solutions.

Competitor analysis phase

This phase is dedicated to an investigation into competitor applications.

Environmental analysis phase

This phase includes research into the environmental issues concerning the project, like constraints as a result of the Operating System and programming language used for the current program.

Prototyping phase

This phase is entirely dedicated to explorative prototyping. The prototyping was executed in order to find the most suitable way to proceed for the actual development phase.

Documentation phase

This phase is a parallel phase that includes the documentation for each of the phases.

Development phase

The development phase includes the development of the final product as well as changes to the existing program architecture in order to make the final product possible and integration into the web-based server environment using JSP.

Testing and integration phase

This phase includes the following activities:

- Testing and integration of changes to the original application;
- Testing and integration of the portal application;
- Testing and integration of the interface with printing devices.

Finalization phase (graduation report and related activities)

This phase includes the following activities:

- Integration of the code deliverables from the development phase;
- Testing of the changes made to existing software as well as the newly written components;
- Writing the graduation project report and its public summary (this document, internally referred to as [GRAREP-PUB]);
- Creation of the final presentation;
- Finalizing administrative activities.

At several moments in time, some phases were executed in parallel. The scheme below visualizes the phases during the execution period running from January 3rd to August 29th 2008.

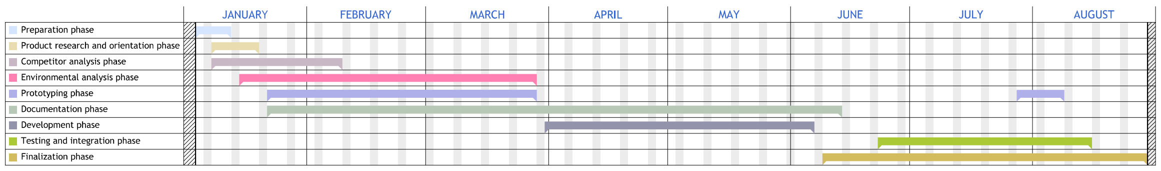


Figure 1: Gantt Chart depicting the phased planning of the entire project

2.3.2 KEY ACTIVITIES

After the necessary primary research was performed in order to be able to have a view about « what's out there » on possible solutions, the key activities could be formulated which apply to the key objectives of the project. In total ten of these objectives were identified of which two were considered to be optional in the case when extra time would be available at the end of the project. These objectives with a brief description are presented hereunder.

- Study on Microsoft-based installations (how does Microsoft deploy installations, what strategy is used);
- Study on Microsoft-based operating systems (Windows versions, in order to determine how in general deployment works and which constraints are presented by the operating system);
- Study on Java Web Start (what are the features Java Web Start offers and how can they be used);

- Study on the Java Web Start environment (how does Java Web Start behave in different browsers and on different Java Runtime Environments);
- Study on the role of Java Web Start in the deployment of applications;
- Construction of an experimental prototype of the application to be deployed with Java Web Start;
- Construction of a demonstrative prototype of the application using Java Web Start, hosted on a target;
- Construction of a demonstrative prototype of the application using Microsoft-based Active Directory deployment (using a Group Policy Object);
- *Interoperability between different installation modes;*
- *Integration in the existing web environment on printing devices using JSP and AJAX).*

The last two objectives in italic font are the ones considered optional. At this point in the document the reader might be a little overwhelmed by the terminology used in this list of objectives. This terminology is explained later in the document when the research part is discussed. The reader should not worry when the terminology is not completely clear at this point.

As indicated in the short problem statement in section 2.2.1, we have as a constraint that there are native libraries involved that are Microsoft Windows based (DLL’s). This is the reason why we also included platform dependent deployment issues in this project, but much of the discussion and concepts presented throughout this document can also apply for other operating systems such as Linux or Unix.

3. Problem statement

In this chapter the problem statement will be presented, or in other terms the project mission. It is in this sub domain where it becomes clear that computer scientists are engineers, for they have the difficult task of interpreting the wishes of the client. Client wishes must be translated to specific requirements that the computer scientist can later use to create his functional and architectural design, as well as to make the implementation and check whether it meets the original requirements. The requirements form therefore the foundation of the entire project. And since no decent building can be built without a decent foundation, also no successful project can be executed without a correct set of requirements. This is what this section is all about.

3.1 Problem given

The description of the problem to be solved given at start of the project was a rather vague one, though clearly complicated. Before actually describing the problem, we first need to understand the context of the problem, which is explained below.

3.1.1 Business context

Printing documents is not always a simple task, especially when you are using dedicated machinery for professional use that has sometimes thousands of parameters that can be set. The Océ R&D department in Créteil focuses primarily on the so called Wide Format Printing Systems (WFPS), which enable users to print usually on large roles of paper. Such printing devices are used for example for architectural designs, maps, banners, posters, etc.

Next to that, Océ aims at providing customers with printing devices that deliver an output that is of the highest quality standards involving high resolution true color output at amazing speed.

Professional printing systems are in general not used like “normal office printers” where you would print to directly from your computer application. In the world of WFPS you rather create plot files in a common format like PDF, PostScript or HPGL/2 (a standard from Hewlett-Packard). These plot files can then be sent to the printer, which takes care of the correct positioning and scaling of the document.

In order to do so, document professionals use a *submitter*. A submitter is an application that interprets plot files and allows the user to compose a job that is ready to be sent to the printer. It uses the printer driver in order to be able to send a stream of data to the printer it can process. This submitter also allows the user to configure the layout of his drawing or document on the media (which is in most cases paper).

The level where this project is situated is this world of submitter applications.

3.1.2 Problem description

In the world of submitters, there is a lot of competition and therefore Océ needs to guard the quality of its submitter applications.

The popularity of the internet and web-based applications also found its way into the world of printing devices and the world of submitters in particular. Most printers nowadays (at least when they are connected to a network) offer a web interface accessible via a web browser. What the user gets differs a lot: from a simple queue interface to fully functional submitter tools. This gave rise to the first discussion point: should the application (being a stand alone application) be made web-based?

A second discussion point came up according to the development environment chosen for the application: Java. In order to make Java desktop applications run, you obligatorily need a Java Runtime Environment (JRE). And in order to get this JRE to work, you need to install it having administrative rights (because the installation of the JRE touches the “core” of Windows operating systems). This particular point — the necessity of having administrative rights to be able to install or update a JRE that is necessary for the application to run — is becoming increasingly problematic.

The third — and last — discussion point that is strongly related to the second is about problems with different JRE versions. A large diversity of JRE versions exist and in the past they were not all necessarily backwards compatible.

The goal was to find a solution to all of these problems, without really knowing what was most important and what was exactly desired.

3.2 Problem translated to specific requirements

As already stated in the previous section and which should be clear after reading it, is that the starting situation was rather vague. A lot of discussions and wishes, but very little of it was concrete. Thus the first task was to find out what exactly was the problem, to identify it — and especially — to correctly formulate it. In order to do so, let’s first recapitulate the three key issues:

- Is a web-based approach necessary and/or desirable?
- An installation without administrative permissions is desired or — even better — having no installation at all;
- No more troubles with Java Runtime Environments.

In fact, these issues can be translated as symptoms of and the idea of a solution for a much larger problem. We will discuss briefly how to identify these issues and to relate them to the problem lying at the base.

3.2.1 Interpretation of the wish for a web application

When you look at the reasons why web-based solutions are chosen, this is usually related to the fact not having to install anything, for the application runs on a server instead of on a client. Furthermore your application will always be up to date and you don’t need to care about user rights. The application will always work, regardless of the environment of the user, as long as he has access to a web browser supporting the technologies used by the web application. Web-based applications are therefore a solution to a problem related to the **deployment** of applications.

3.2.2 Interpretation of the user rights problem

Having a problem with user rights during the installation of an application and thus the related wish of not having to install anything at all is not difficult to relate once again to a **deployment** problem. Especially with the ever increasing demand of more security within and around software, the installation of programs on operating systems is more and more restricted in order to keep malware outside. This has as a direct consequence that if your application needs a high level of access, that installing it becomes increasingly difficult.

3.2.3 Interpretation of conflicting Java Runtime Environments

Conflicting JRE versions are first of all due to the requirement of the application needing a JRE. When developing a Java application, you should know at forehand that in order to have customers use your application, they need to have a JRE installed and sometimes even a specific version. Thus in order to be sure that a customer can use your program; you need to **deploy** a correctly functioning JRE with the application.

3.2.4 Focus on deployment

Resuming the previous three paragraphs leads to the identification of the central problem: deployment. The environment of the concept of deployment knows problems and solutions, like any concept. The key issues presented at the beginning of this project were in fact a subset of the problems and solutions around deployment.

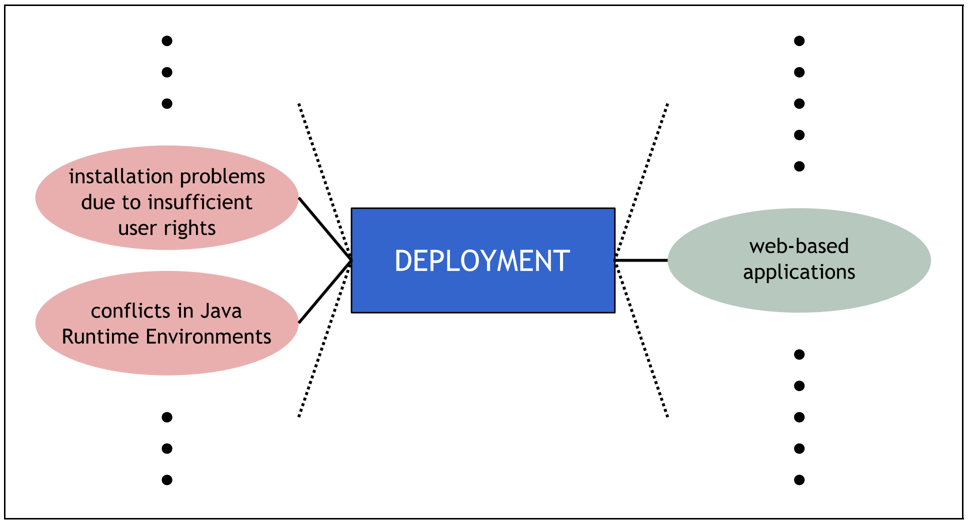

installation problems due to insufficient user rights
conflicts in Java Runtime Environments
DEPLOYMENT
web-based applications

Figure 2: the identification of the central problem: deployment. Identified problems are depicted on the left, an identified potential solution on the right. Of course there are many more potential problems and solutions related to the deployment, depicted by dots.

3.2.5 Specific requirements

After extensive talks with the people involved in the project group, we were able to identify the specific requirements, i.e. to formulate what really matters and needs to be solved without thinking in terms of problems and potential solutions. These are the following:

- Users with a restrictive rights level must be able to install the application;
- The installation (if any) needs to be kept as simple and as straightforward as possible;
- The impact to the existing source code of the program must be kept as minimal as possible;
- Interoperability between different modes of installation is desired.

It is clear – given the environment we are working in – that these requirements require a considerable amount of research and tend to be very complex. On top of that, we have to deal with different workflows, i.e. the way users tend to use a submitter application in their business process. These issues will be addressed on the way.

4. Research and investigation

As a result of the complex environments and many details that play a role as explained in section 3, a considerable amount of research and investigation needed to be done. First of all it was important to find out what options there are in order to facilitate the deployment process. Questions like "What are common technologies used for deployment in Computer Science?" and "How do we keep the impact on the existing source code of the application minimal?" play a central role here. Next to that we will make a discovery trip through the world of submitters and potential architectural layouts for a solution.

4.1 Deployment in Computer Science

Deployment is an important phase within the software engineering process, which also finds its place in the classic waterfall model presented below.

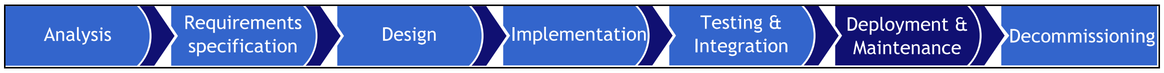


Figure 3: the location of deployment in the waterfall model of software development

The deployment and maintenance phase is the last "constructive" phase of software development, the phase were the product finally meets the customer.

The position of deployment in Computer Science however, is becoming increasingly complicated. We will begin our discussion with presenting six key issues that are related to deployment and how their current state is within the domain of Computer Science.

4.1.1 Operating Systems and environments

Different versions of operating systems and multiple platforms don't make the life of software engineers easy. This is why most developers focus on one platform only, although there is a strong tendency to favor cross-platform solutions. Each operating system has in general its own way of storing program files and how programs are accessible to the user.

Next to the operating system itself, it can also be the case that a special environment is necessary in order to run the application to be deployed, like a framework. Two widely used frameworks are for example the Java Runtime Environment and the MS .NET Framework. Applications written using these frameworks will not be able to run without them.

4.1.2 Resource sharing

The deployment task becomes extra complicated when there are resources being used by the application that might also be used by other applications. Typical questions are then whether or not the shared components are already present and if they are of the correct version; and what to do if they are outdated? Can they be replaced with a newer version or does this have a negative effect on the way existing applications use this resource? Almost any software developer knows the problems that may arise as a result of software libraries that are not of the correct version.

4.1.3 Security

Deploying an application is far from risk-free. In general you don't know anything about the state the machine is in at the moment of deployment. But in general you *do* want to deliver your application exactly the way as intended. Security comes in especially when the application needs to perform actions that need high security levels (like banking applications for example). The security of the application and its resources must be assured and integrity checks on binaries and data files become a real must in this case.

4.1.4 Licensing

When deploying commercial software that needs to be paid for, you don't want that people copy and redistribute your software without permission. It is impossible to stop people from copying installations, but it *is* possible to stop them from executing a deployment process without authorization. The classic way this is done is using a key that is owned by the customer who paid for the product. Of course this method of protection is quite prone to abuse, for keys can be lost or stolen.

4.1.5 Access control

Access control or — more popularly formulated — *user rights* is an increasingly difficult issue when it comes to deployment. Operating systems are being secured tighter and tighter in order to keep malware outside. In a world where viruses and spyware are omnipresent this is not a superfluous luxury. But the negative side of this story is that the deployment of *good* applications is becoming more and more difficult as well. Users in professional organizations almost never have (local) administrative rights, nor any other rights level that suffices for the installation of applications. Sometimes this is intentional, for example in the educative sector where system administrators don't want students to be able to install anything on a computer. But operating systems are now making it tighter already by default. For example on the relatively new operating system MS Windows Vista, you cannot be administrator by default and you will need to give explicit installation permissions whenever you want to install anything.

4.1.6 Updates

Updating software is also becoming very popular and more and more frequent. Sometimes updates are "abused" by software developers in order to deliver their product to the customer before it is actually finished. Feature extensions and bug fixes are then used to provide customers with the final product once finished. With the internet being also much more present among home and office users, updates can be easily acquired. In many cases software developers provide an automated updating mechanism to their users such that they don't have to bother about checking for updates themselves.

The risk of using updates extensively is that your customers don't have the version installed you had in mind, especially if there is no automatic update feature implemented or a missing internet connection.

4.2 The world of submitters

Submitter programs are used mostly by document experts who have to print large amounts of documents in different formats and orientations. A submitter program offers an interface with the printer in such a way that the user is able to apply the correct output settings in order to obtain his document exactly in the way he intended.

Océ has a long history of creating such submitter applications. As technology progresses, Océ also developed submitter applications of higher quality bringing these new technologies to the end user.

4.3 Submitter workflows

Understanding the way submitters should work and what is important for users depends heavily on the way they are actually used. In other terms, we need to investigate the submission process the end-user performs. This end-user is in our case in general a document specialist that needs a large collection of possible settings in order to get his document printed the way he wants as well as an optimized way to get there (that is, not losing a lot of time for making difficult configurations).

This part of the workflow is not specifically important to the scope of this project, the part we are interested in here is the part that comes just before it: installing and launching the application. Therefore we will focus on the installation and usage workflow of submitters.

4.3.1 High level workflow

The high level workflow, where we don't bother about the details, was the first step in order to get a hold on what installation and launching workflow we wanted to support. Between "installation" and "launching" you should also read "updating", which will become visible in the workflow as well. In the diagram below this high level workflow is depicted.

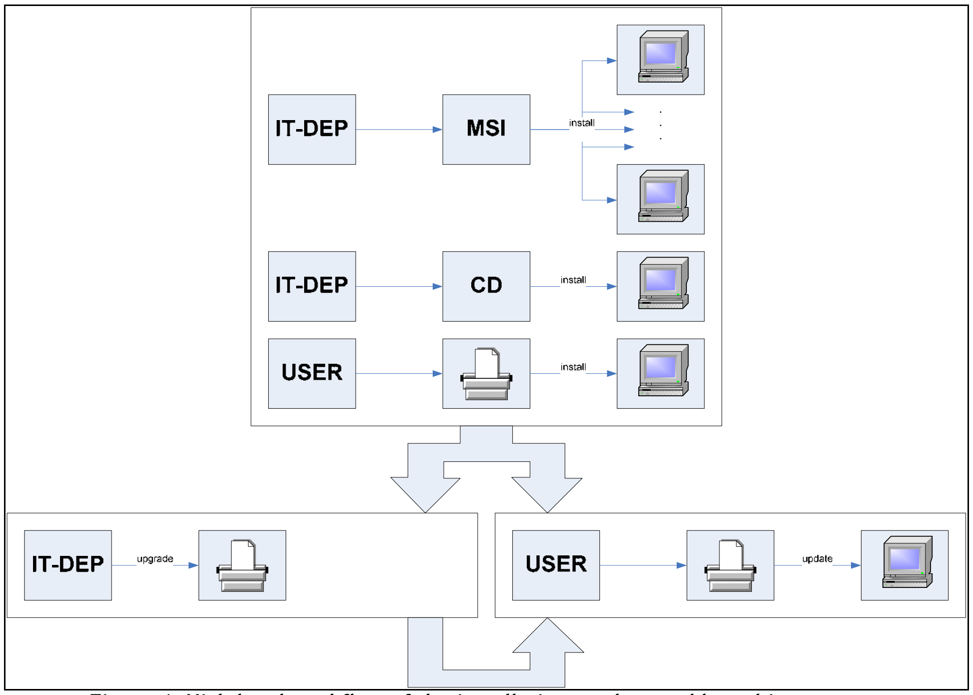


Figure 4: High level workflow of the installation, update and launching process

The basic idea is this: the IT-department or the end-user himself installs the application. In case of an IT specialist, this will typically be done using an MSI (a Windows Installer executable) that can be automatically deployed in a Windows server environment or manually with a CD on each workstation. When left to the user he should acquire the program by downloading it directly from the printer.

Once present on a workstation, the user acquires a possible update automatically via the printer. This will only be the case when an IT specialist put an upgrade of the application on a server device replacing the original installation.

Thus the set-up of the envisaged system for a customer (in case of a large company) is very simple. The IT department of this company only needs to keep the version of the software present on the printer's controller up to date, users over the network using the workstations will acquire updates automatically. In this way the cost of maintenance in time, energy and money is significantly reduced.

4.4 Deployment strategies in computer science

Before narrowing to the setting of this project, we will first discuss general deployment strategies that exist in the domain of computer science and which technologies are laying at the basis of this.

4.4.1 SOFTWARE DEPLOYMENT PROCESS

In [CFSODT] — a technical report of the University of Colorado — the writers identify the software deployment process as a set of the following activities:

- **Release —** the activity that serves as the interface between the deployment and development process;
- **Install** — the initial insertion of a system into a consumer site;
- **Activate** — starting up the executable components of a system;
- **Deactivate** — shutting down any executable components of a system;
- **Update** — special case of installation where replacement, addition or removal of components can take place with respect to a previously installed system, initiated by events at the producer's side;
- **Adaptation** — modification of a previously installed system, initiated by events at the consumer's side;
- **Uninstall** — removal of a system when the system as a whole is no longer required;
- **Retire** — marking of a system as obsolete and support is withdrawn by the producer.

These activities within the software deployment process can be integrated in the following model, also originating from [CFSODT]:

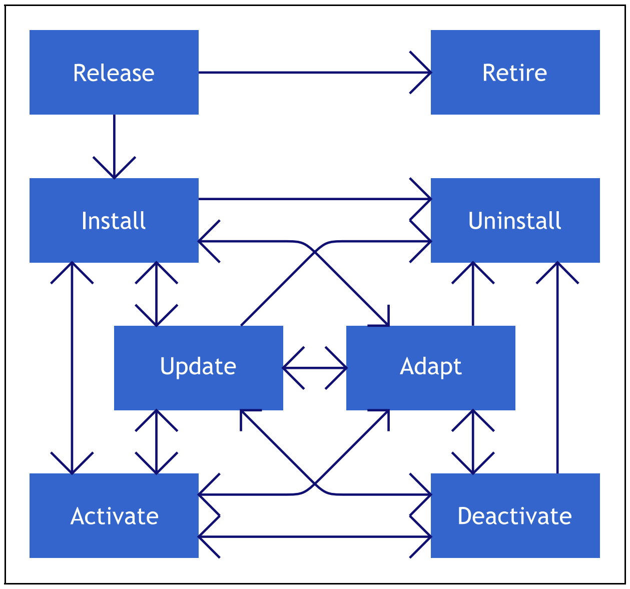


Figure 5: the software deployment process as in [CFSODT]

In the following paragraphs we will discuss existing common approaches to support this deployment process. Each of these approaches applies to supporting one or more activities cited above.

4.4.2 CLASSIC, ATTENDED INSTALL

The classic, attended install is the most common way of deployment. It includes an application that lets the user typically accept a license agreement, choose a destination and select the components to install. The application then copies some files, creates directories and optionally modifies some system files (for example under MS Windows the registry).

The user performing the installation needs to be present, needs to *attend* the installation. Without user interaction the installation cannot complete. For the installation of applications or an operating system on one or a small number of computers, this installation mode is an acceptable and most cases even a suitable procedure. However, when the number of applications and/or the number of computers is increasing, this type of installation can become problematic.

The classic installer supports mainly only the install activity in the development process, although most classic installers currently also include an uninstall feature covering therefore also the uninstall activity.

4.4.3 UNATTENDED INSTALL

The unattended install is a strategy where the user actions in the classic, attended install are omitted, in general using a configuration file that contains the necessary information to make the choices normally the user would make. This method liberates the user from being present during the installation. Concerning the software deployment process, the same items are covered as for the classic installer.

4.4.4 AUTOMATIC REMOTE INSTALL

Automated, usually remote installations are installation methods that are being performed within a network environment where there are many computers (workstations) that must be configured in order to contain the correct operating system and applications. A server is then usually configured that takes care of these computers that once logged on to the network, the software is copied from the server to the workstation and gets installed automatically, bypassing any kind of local user level. This technique is widely integrated in operating systems in a networking configuration like the MS Windows Server editions. Further on in this document this is discussed as one of the possible options for solving the deployment problem of our application. This installation type also covers the same items in the deployment process as the classic installer.

4.4.5 WEB-BASED PROGRAMS/SERVICES

Web-based programs and services are relatively new, but especially as a result of the ongoing technological progress in the field of internet/web based applications, this concept’s popularity is growing drastically.

Web-based programs do simply *not* require any installation at all. The web browser serves as the interface between a (park of) server(s) that contains the executable code and accompanying data. At the client side the result of the user’s operations is only displayed. Execution of the program or service takes entirely place at the server side. This results also in the situation that web-based programs can be characterized with respect to the deployment process with all activities, except release and retire. The other six activities are all part of a web-based system.

This strategy is used further on in this document as a possible solution to our deployment problem.

4.4.6 Image generation and restoration

A rather drastic way of deploying rapidly a preferred system configuration is the image generation and restoration technique. The advantage is that it works very fast compared to any other method and always gives the desired result. The big disadvantage is that it can only be used when using computers with exactly the same hardware configuration. Although this method is rather drastic, it doesn't change the coverage of activities in the deployment process presented above with respect to the classic installer, the install and uninstall activities are the only ones supported.

This method consists out of installing the operating system and applications on one machine to obtain the desired configuration. Once ready, one of the many existing tools in this field can be used to make a low-level copy of the entire hard disk, a so-called *image*. This image can then be written to another hard disk of another computer that has the same hardware configuration of the originating machine. The result is that for the new machine the state of the operating system and applications will be exactly the same as for the originating machine.

This technique is widely used in large companies where there are many computers that must have the same (basic) configuration. Another disadvantage of this method is that you cannot select which applications you want to take over from the originating machine.

4.4.7 Automatic updates

Automatic updates is a rather new concept that searches for updates via a network or internet to download and automatically install them. The "automatic updates" mechanism within MS Windows is widely known and is used for high priority updates on the operating system. Similar systems can be found in widely used software applications and platforms, like for example the Java Runtime Environment. Once an update is available, the user can be alerted that an update should be installed, or the system can follow a more dictatorial approach where the user is forced to perform the update, with or without notice to the user.

The problem with automatic updates however, is that there are situations wherein they have difficulties with performing the tasks they are designed for. When the user is not an administrator, the update process can usually not be executed because the files that must be replaced are only replaceable by an administrator account. In MS Windows for the automatic updates feature this is solved by running the automatic updates process as a system service, bypassing user accounts. For applications that are not part of the operating system this access is not possible.

Within the software deployment process, automatic updates is merely a means that is oriented to the update activity and does not really have anything to do with other activities within the deployment process.

4.4.8 Platform based deployment systems

Platform based installations with automated update and application launching features is a rather new development in the field of software deployment. For the moment, Java Web Start has the lead in this field for Java based applications. Java Web Start exists since 2001, but Microsoft didn't wait a long time before introducing a similar technology into the market for their .NET Framwork, called *ClickOnce* [MSCLON].

Platform based install/update and launch mechanisms are based on covering the install, update, activation and uninstall activity of the deployment process. We could therefore classify this deployment strategy in between the two extremes, the classic installer and the web based application. In the figure below the coverage of platform-based deployment systems with respect to these two extremes is visualized.

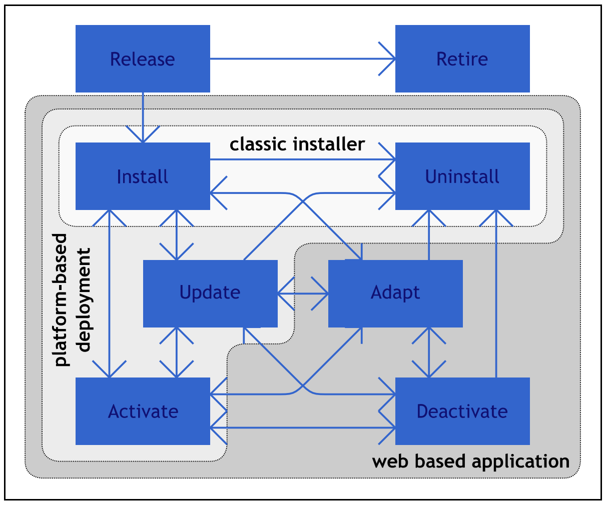


Figure 6: deployment process coverage for a classic installer, a web based application and Java Web Start

4.5 Possible architectural setups

In section 3.2 we saw that the original key issues of the project included the question whether a web-based solution could solve the problem of deployment. Let's keep this particular question in mind as our main trail towards a solution. Around this trail we should keep our eyes open for technology that might help us out.

During the investigation phase of the project a lot of architectural setups were presented and discussed. We will revisit these setups in this section in order to give a clear and complete view to the reader. Later on in the document we will see why certain setups were put aside and others continued to be investigated and even being tried out in the prototyping phase.

Three streams within the architectural setups have been investigated which will be presented in the following subsections:

- Setups using a pure web-based approach;
- Setups using an intermediate platform;
- Setup using a central server.

4.5.1 Architectural setups using a (pure) web-based approach

Before moving to concrete solutions, it is a good idea to start by having a look which options we have in terms of architectural setups of a web-based system where the application runs on a server or printer controller and we have a browser on a client computer as the main means accessing its functionality.

In web design there are not that many options and therefore we can easily cover each of the options in the following sub paragraphs:

A. Direct browser to server communication;

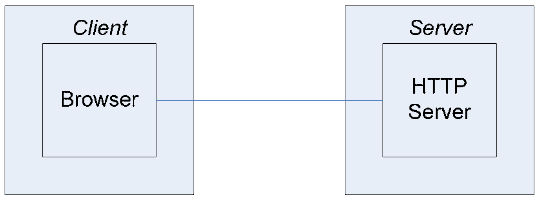
B. Browser to server with an additional functional layer at the client side;

C. Browser to server with an additional function layer at the server side;

D. A combination of the last two.

4.5.1A Direct browser to server communication

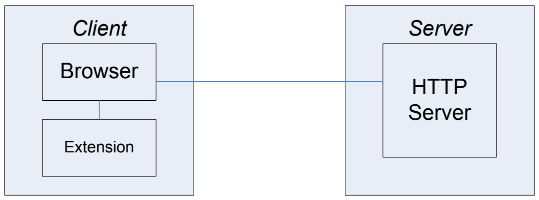
Direct browser to server communication is the simplest form of an architectural setup, which consists in just a server application running on the server side which directly handles all requests done by a client connected to it.



This approach will unfortunately not work for a submitter. The reason for this is that there are complex tasks that need to be handled, like the processing of image data. This requires special libraries that usually do not reside in standard server applications handling HTTP requests.

4.5.1B Browser to server communication with client extensions

This setup is similar to the direct browser to server communication with just having an extra functional layer. This functional layer then fills up the gap of the functionality missing in standard browsers and standard server applications. In most cases these are extensions built into the browser, but one could also continue completely outside of the browser using for example a stand-alone application that is loaded from the browser or a service.

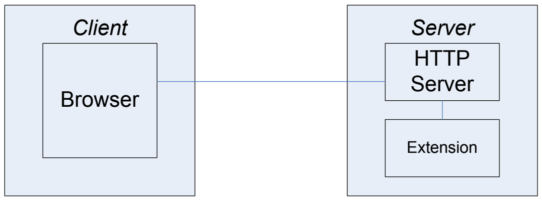


This way of solving the lack of functionality in the standard client-server chain has many advantages. If file analysis is involved (which is the case in our situation) then these files can be analyzed locally without transferring them to the server-side which saves bandwidth and time. A disadvantage is that if the executable code doesn’t reside at the client-side, it has to be downloaded each time it needs to be used (that is, if there is no caching by the browser or that the cache was emptied). Another annoying problem is that in case of an extension to the browser, not all browsers support this function. One can think of an easy example where a user has disabled JavaScript (e.g. for security reasons). Websites that require JavaScript to function correctly won't function anymore.

Other solutions in this setup include the usage of applets or other executable objects that are loaded into the browser. Here not only the ability of the browser to support this kind of interaction is a potential source of problems, but also the supporting platform must be present (like a JRE in the case of java applets or access to Windows system resources in case of an OCX which is on top of that browser dependent).

4.5.1C Browser to server communication with server extensions

Analogue to the setup described in the previous paragraph, one can think of putting the extra functional layer at the server side in stead of the client side. A major advantage is that you can supply the user with a full result of the request without having anything to do client side (and especially, as a result of this, don't have to install anything). This situation is often preferred because it preserves the nice property of the direct browser to server communication setup that runs the use of any additional tools at the client side unnecessary. You immediately solve any potential problems with installations and even cross platform issues. As long as your clients are equipped with a functional web browser that respects basic standards, your application can be used everywhere.



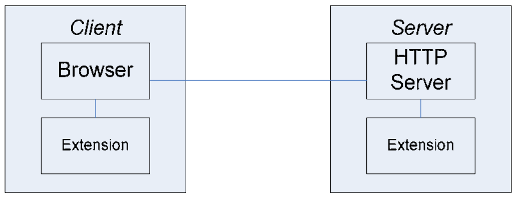
The options developers have in this setup are as extensive as those for the client-side additions. In most cases developers choose for extensions in the form of extra libraries, sometimes even compiled into the server application used in first instance.

For users there are in general two potential problems in this configuration. The first one appears when you have tasks in your application which are quite process power consuming and you have many users that access the application at the same time. The server may not have the hardware capacities you would need in order to serve all the requests in an acceptable timeframe. The second problem is that if file analysis is involved (which is the case for us), you are obliged to transfer the file to the server before the server actually can do something with it. When the files are not too large, this might work well, but if you have files of hundreds of megabytes to transfer, this can result into unacceptable response times. Of course, this also heavily depends on the chosen hardware configuration.

For an analysis in depth I refer to [MIGISS] where a practical central-server based solution is discussed.

4.5.1D Combination of client and server side extensions

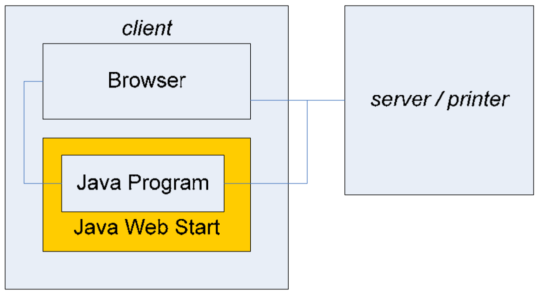
This option doesn't require much explanation. Most ("grown-up") web-based applications use extensions on both sides. The border between which kind of functions should reside on the client side and which should reside on the server side is rather vague and there are no real guidelines that describe this border clearly.



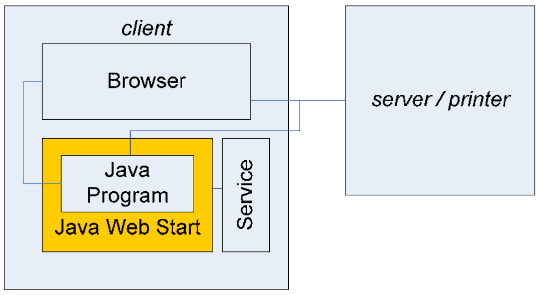
4.5.2 Architectural setups using an intermediate platform

Next to extending a browser in order to add functionality, there also exist platforms that enable programs to run upon. For Java, such a platform exists as well, largely integrated with all the needs related to the web: Java Web Start.

The advantage of such an intermediate platform is that in most cases no large scale adaptations are needed to existing source code. For this project, this was even one of the requirements.



Option 1: complete incorporation into Java Web Start



Option 2: Java Web Start in combination with Windows Service

As in section 4.5.1, the question arises whether one would want to run the entire application on such a platform, or that one would like to use further external functional sources like a Windows Service. Given the fact that Java Web Start caches the application, it won't be necessary to download the entire package again when the user wants to use it once more. Another major advantage is that Java Web Start sorts out automatically whether there is an update available or not and if so, it downloads automatically those parts of the package that are updated and installs them. Given this approach, it looks tempting to keep the application intact, in order to avoid the rights problem when you have to install a service, as well as the update problems that will arise as well.

We will discuss the Java Web Start technology in depth in section 4.6.

4.5.3 SUMMARY OF ARCHITECTURAL SETUPS

We saw in section 4.4.8 — more precisely in figure 8 — where we presented the mapping of the classic installer, platform-based deployment and web-based applications to the key items of the deployment process, that the latter two cover significantly more items than the classic installer. This is the reason why we will primarily focus on the platform-based deployment (Java Web Start, sections 4.6 to 4.9) and a web-based approach (4.10).

Finally, to summarize the discussion above, the following table indicates the main properties and (dis)advantages of the architectural setups discussed.

	Direct browser to server	Browser to server with client extensions	Browser to server with server extensions	Browser to server, extensions both sides	Inter-mediate platform
Feasible for submitter	no	yes	yes	yes	yes
Network load	-	high	high	high	low*
Potential security problems	-	yes	no	yes	yes
Smooth interaction	-	medium	no	medium	yes*
Suited application size	-	small only	small to big	small to medium	small to medium
Adaptations source code	-	medium to high	medium to high	medium to high	low

*) except on first launch

4.6 Java Web Start

Java Web Start (JAWS) is a relatively new platform developed by Sun Microsystems and was introduced in March 2001. Since Java version 1.4, JAWS is installed automatically. In short, JAWS serves as a platform reachable from a browser in order to run Java applications on it.

In the following paragraphs we will discuss the basic principles of JAWS and in the next three sections we will look in detail into the server-side and client-side configuration as well as a minimal sample application to illustrate basic usage of this technology.

4.6.1 JAWS APPLICATIONS LIFECYCLE

In the image below you can see the lifecycle of JAWS applications.

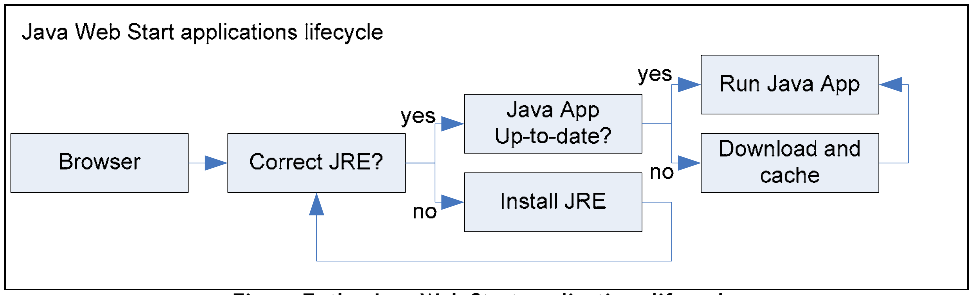


Figure 7: the Java Web Start applications lifecycle

From a browser, there are two options:

1) If there is a JRE installed it detects whether the JRE prescribed by the JAWS application is present on the system. If not, it acquires this JRE automatically.

2) If there is no JRE installed, a link to a JAWS application will have an effect like downloading an unknown file. This situation can be avoided by incorporating a simple script in the webpage the user is viewing. This script can detect whether Java and/or JAWS is installed. If the script detects that this is not the case, the user can be directed to a site where he can download a JRE.

Once the correct JRE is started, JAWS checks whether or not the requested application is already present in the JAWS-cache. If this is the case, JAWS checks if there is a new version available. If not, the application is started from the JAWS-cache. In the other cases JAWS downloads the application (or the part of it that was updated).

The advantage of this lifecycle is that it is fully automated. Once there is a JRE present on a system, the application will always run and it will always be up to date.

4.6.2 NETWORK LOAD

The network load of JAWS is very much optimized. Normally – that is, if the developers do a good job – the entire application is only downloaded once. Afterwards, only the outdated parts are being downloaded and replaced.

Another major advantage of this caching system is that the application is always available, also if the network connection is lost.

Furthermore, JAWS applications use the JNLP (Java Network Launching Protocol) which enables developers to indicate which parts of the program are necessary to be downloaded for basic operation (*eager acquiring*) or those parts that can be downloaded at the moment the user actually tries to access them (*lazy acquiring*).

4.6.3 Different ways to start

JAWS is mostly used to start applications from a web browser. But this is not at all restricted to this start-up mode. JAWS supports the following modes to start an application:

- Via a JNLP-hyperlink in a browser;
- Via the Start-menu (when downloading the application for the first time, a shortcut is created automatically);
- From the Desktop (also created automatically at first download);
- Via the command prompt ("javaws" followed by the application name).

4.6.4 Rights

Looking at the key issues that concern us, we had a problem with rights for installing a program on a computer. This problem is potentially solved by JAWS because it downloads all necessary parts of the program in the Application Data folder of the current user under MS Windows (in Documents and Settings). The current user is always in possession of the necessary rights to put files there and to execute them.

4.6.5 Source code adaptations for Java projects

In order to be able to use JAWS, very little adaptations are required to the existing source code if one is using "standard Java". JAWS was actually developed in such a way that existing Java applications as well as applets could be loaded into a JAWS environment with minimal adaptations. Given the fact that we are focusing here on a Java application, we do not consider the case of Java applets here.

For standard Java applications, the source code should be checked on two issues:

- Resources must be bundled in a JAR file;
- If unrestricted file access is needed, the source code must be signed.

Regarding our application, a short inquiry in the development team showed that resources are already bundled in JAR files. Code signing is a known concept, but is not yet practiced. The effort of signing the code is not considered as a major problem.

Thus from the development point of view, we could be quite optimistic about a JAWS-based solution.

For more details on this signing operation I refer to paragraph 4.9, where signing is discussed in a broader sense, with examples.

4.7 Java Web Start / server side

In order to use JAWS, there are two sides of the tale. We saw in the previous chapter that the client-side adaptations are very minimal. But we also need a server side; that is a device that is capable of providing a user with the packages and the JNLP file that is required in order to run the application from a web browser. This process is thoroughly described in [URL-DEPLOY]. In the following paragraphs I will give a brief summary of this, according to the specific needs of our application.

4.7.1 JNLP Configuration

For deploying the application, one can follow the standard procedure of creating one or multiple JAR files. But it doesn't end here; one has to make an XML document that follows the JNLP specification. This file serves as a configuration file how the application should be used by JAWS in order to deliver it correctly to the user that downloads it. In the following subparagraphs the key sections of this file are discussed.

4.7.1A JNLP Section

This section specifies where the application is located on the server side (that is the folder with the JAR files and the corresponding JNLP file itself). This is the outer section of the XML document.

4.7.1B Information section

Here you can set up a series of descriptive tags that will show up when a user is downloading the application. Tags include a title, vendor, description and a homepage (link).

4.7.1C Offline-allow

This tag (residing within the <information> section) indicates whether or not the application may be run without the presence of an internet or network connection.

4.7.1D Security

This section is optional, but required for applications that need disk access. This section should be set to 'all-permissions' in order to work correctly. This however, will only work if **all** JAR files are signed.

4.7.1E Resources

This section indicates where the JAR file(s) is (are) located that contain the application classes, as well as the JRE that should be used. The JAR tag can be used n times if you are using multiple JAR files.

4.7.1F Main class

If there is a main class specified in the manifest of the JAR file, this section can be skipped, otherwise you will need to specify the main class here using the 'application-desc' tag.

4.7.2 JNLP Installation options

Additionally to section 4.7.1, we have a look at the JNLP options we have that are platform dependent. A full reference for the JNLP elements and tags can be found at:

http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/syntax.html

4.7.3 Signing

The JAR files used must – for security reasons – be signed. In order to do so, it suffices to use the jarsigner tool that comes with the SDK. Without signing, the application won't run within JAWS.

4.7.4 Web server configuration

If the java-based TomCat server version 4.0.x or higher is used, it suffices to put the XML file (stored as a .jnlp file) together with the JAR file(s) on the web server. Accessing the jnlp-file through a normal link (<a href="link_to_file.jnlp">start application</a>) does the job. However, if this is not the case, you need to configure the used web server with the JNLP MIME type. MIME types are file descriptors that can be defined on web servers. Each web server uses its own way of describing these, therefore I point to the web server documentation for further details on this issue. The required MIME type is:

```
application/x-java-jnlp-file
```

4.8 Java Web Start / client side

The client side configuration of JAWS is a rather complicated issue. The fundamental reason is the situation in the way MS Windows handles user profiles. JAWS is developed in the intention of keeping it platform independent and therefore the JRE — which is created for each operating system independently — needs to make certain decisions. For JAWS this comes down to making a choice where to put temporary files (the JAWS cache). Especially when you are in a large company having a network with a central server and workstations, the consequences of where you actually put the JAWS cache can be far-reaching. In this chapter we will look at the standard configuration as set by default with its advantages and disadvantages.

4.8.1 How does the JAWS cache work?

When you download a program that uses JAWS, the necessary files in order to run the application are copied into a caching system. This caching system can be configured for the following items:

- Enable/disable cache (**default**: enabled);
- The location of the cache (**default**: user profile, application data);
- The size of the cache (**default**: 1000 MB);
- Compression (**default**: none).

4.8.2 Potentially problematic settings

Some of these settings can have values that pose problems for running applications like we want for our application.

The first problem arises when a user would choose to disable the cache. When the architecture of your application is such that you put the entire application into JAWS and that the application is rather heavy, then a user needs to download the application each time again when he wants to use it. This causes of course a significant increase in the network utilization, which is in general not appreciated. The same holds for a user that configures a very small cache, where the cache size is less than the size of your application.

Another potential problem is the location of the cache. This issue will be addressed in detail in paragraph 4.8.4 regarding the way MS Windows handles profiles. Here it suffices to notice that if a user chooses for example a network location, that the problem of a significant network load increase shows up again.

4.8.3 CACHE EMPTYING POLICY

When JAWS reaches the limit of the cache (at least, if the cache is not disabled), JAWS will start to delete some applications form the cache in order to make room. JAWS uses the *Least Recently Used* algorithm for this action; that is that the application that you didn't use for the longest time is deleted first.

4.8.4 JAWS CACHE IN MS WINDOWS USER PROFILES

By far the most problematic issue with the JAWS cache is the way MS Windows handles user profiles. First of all there is a different situation when you have stand alone computers (whether or not connected to a network) or a central server with workstations connected to it.

In the first situation there is not much to worry about; it doesn't matter much where the JAWS cache is located. The only risky thing is that an application may appear *n* times on the local hard drive where *n* is the number of users on the system.

If this is a problem, this can be solved by using the system cache of JAWS. In fact, the caching system of JAWS consists out of two parts: a user part and a system part. The system part is only accessible by an administrator and cannot be changed by normal users. If there is an application that will be used by (almost) all users, it is recommended that the administrator puts this application in the system cache in order to avoid the *n*-copies situation.

In the latter situation — the one where we have a central server with workstations connected to it — the situation becomes really complicated and even annoying. This is a result of the concept called *roaming*, which means that user profiles can be stored on the central server and are synchronized each time a user logs on to *any* workstation. In order to make the situation even more complicated, user profiles exist out of roaming and non-roaming parts and these can even be configured by the network administrator.

The ultimate disaster situation can easily be identified: imagine that the JAWS user cache is located in the roaming part of the user profile (which is by default the case under Windows XP) and that the cache size is set to 1000 MB (which is also the case by default) and that we have *n* users on our network which all use JAWS heavily, that is up to the limits. Every morning that our *n* users log in, *n* gigabytes of data have to be transferred…

4.9 Java Web Start / sample

We finalize our discussion on Java Web Start with a sample application. The full source code of this sample can be found in appendix B. This sample application does the following:

- It can create a fresh subdirectory in the current Windows User profile (typically 'c:\Documents and Settings\[username]');
- It can extract an executable from a secondary JAR file and put it into the newly created folder.

When we look at this, we see that we are obliged to get out of the sandbox of JAWS (a protected environment that protects a user against malicious programs coming into his computer). This environment does in a normal situation not allow system critical operations, like disk access. Therefore we need to do the following:

- Generate a JNLP file and set the security level to 'all-permissions'. This will invoke a special notification for the user when downloading the application whether or not he/she is really sure to grant high level permissions to the application;
- All JAR files (in this sample project just two) must be signed;
- In order to be able to sign, a key store and certificate must be created;
- We need to be able to extract files from a JAR file and copy it to the local disk.

In appendix B these issues are addressed for the interested reader.

4.10 Client-server web applications

In our discussion on architectural setups in section 4.4, we saw that client-server web applications are a promising alternative. Now we need to make a choice that if we follow a web-based approach, which approach we should take.

For our submitter application, there are several key aspects we must consider:

- Clients usually have large files that are difficult to transfer to a server "just" for analysis purposes;
- Browsers do not supply sufficient internal functionality to handle the tasks we need;
- For the sake of deployment, we want to avoid any installation of a supplementary tool.

Taking these issues in account, we already see a conflict. There is no solution that meets exactly these demands. Transferring files for analysis is probably no option due to the necessary hardware requirements, thus we are obliged to use at least client-side extensions.

When all given requirements cannot be met into detail, we have no other choice then to see which of the requirements can be relaxed up until a certain point that it enables us to answer to them and that the relaxation of the requirements stays within acceptable bounds. When looking at the key aspects given just above, we can only apply relaxation on the first and last one.

4.10.1 Relaxation on the no file transfer requirement

The “no file transfer for analysis” requirement is a little bit particular. In fact, it is perfectly understandable that for bandwidth issues it is not a desirable solution that users massively upload files to a server (or printer) not in order to print them but just to analyze the file and get a preview.

One could actually think about whether this would actually be the case. Following talks with developers and users in training environments in both Créteil and Venlo pointed out that in most cases users already know what they want to print and the only thing they are interested in is getting their result in the right position. Thus, in other terms, the plot file will be transferred to the server/printer anyway. Therefore one could ask himself the question whether it is a problem to transfer the file a little bit earlier in the printing process.

A possible solution would be to upload files to a local server/printer store, defining the desired configuration with help of previews (that could be generated server-side and send back to the client) and once the correct settings have been chosen, print it directly. Dependent on the storage capacities, it could be possible to load and save jobs with the files in question. Let’s look at the issues we looked into when performing the analysis in [INQSUB] to classify other systems how a server-based solution could meet these issues:

Multiple document selection: this is a potentially problematic issue. With the increase in security rules it becomes virtually impossible to use website embedded objects to access the file system in order to do a multiple file selection, especially under Windows Vista where it is not allowed to access the file system in any way. The sole object remaining to upload a file is the typical file input box with the browse button next to it. This field can be repeated almost infinitely many times, but users will need to select files one by one.

Job load/save: equipping the server/printer with sufficient storage capacity would enable users to store and retrieve their jobs. One could think of personal folders on a local hard disk that could be used to store job and print files. When having many users using large print files, large storage capacities will be necessary.

Web-basedness: this solution is as much web-based as web-based can be.

Previewing/thumbnail: file analysis in general takes some time. One could think of the situation where the server immediately analyzes the file after upload and creates a thumbnail that can later be used to create the preview. Simple raster based image transformations are usually performed in less than a second and can be transferred very quickly through a network to the client computer. When the server is equipped with proper processing power and a good network infrastructure is available, this should not be problematic.

Multiple printers support: this depends entirely from the supported drivers, but the list of supported printers is à priori endless.

Ease of install: given the fact that this system would run from any web browser available this moment, no installation whatsoever would be necessary on the client side. The system works immediately and always, no matter what the situation is on the client computer.

4.10.2 Relaxation on the no installation requirement

Another approach is to see how far we can meet the requirement of not installing anything. For determining this, we need to revisit actually why we didn't want to install anything. The reasons for this were the following:

- Possible rights problems for performing an installation when not being a local administrator or power user;
- Possible difficulties that system administrators might have in keeping all user computers up to date.

When looking at this, we could relax the requirement of "we don't want to install anything" a bit. If we could provide an installation that is guaranteed risk-free for potential rights problems and that we find a way of automating the update process for individual computers in a network, we also solved the problem.

Furthermore, in order to reduce the effort of the developers, we should find a solution that can reuse an absolute maximum of the existing source code.

4.10.3 Determining the preferred solution

Given the restrictions we have, mostly as a result of the image operations and the fact that (over)loading the network with transferring files for analysis purposes between client and server, we have a strong preference for a client-side functionality extension.

Extension of the browser is possible in many ways, but given the fact that our current software is written in java, we should avoid options that require extensive adaptations or even complete rewriting. One could think of putting all functionality in java applets, but given the enormous amount of functionality proposed by our application, this is not really a feasible option. Such a solution would cause an enormous load on the network (due to the size of the applets which would be near to the current application size, which is about 50 MB). Furthermore, Windows Vista poses a huge problem in this approach given the fact that Java Applets that need to perform disk operations (like our application) will no longer work on current Java Runtime Environments [JAVA-VISTA-NOTES], [JAVA-ON-VISTA]. This problem specifically related to Windows Vista will be solved in the future, as Sun announced that their way of handling applets will change in the Java 6 update 10 version [JAWIVI]. This development became known during the project, which ultimately lead to a small feasibility study on how a light version of our application could be incorporated into an applet. The main argument why an applet based solution is not feasible remains, the application is far too heavy to be entirely put into one or more applets. A test run described in the prototypes sections proved this point.

A better solution would already be to split the functionality and host it partially in java applets and partially in a Windows Service. However, the operation to adapt the current source code such that it fits into this framework is quite a heavy one, having large scale consequences. Thus, from a source code point of view, this would not be a favorable solution either.

So we don't favor a server-side extension and client-side extensions aren't really feasible as well. So what we want can't be done? No, there still is another option. We already presented it in section 4.5.2 like using an intermediate platform called Java Web Start (JAWS). It fulfills our requirements (although not being purely web-based) and in the previous sections 4.5 to 4.8 we saw the potential of JAWS. The extension of the sample application discussed in section 4.8 to a functional prototype is a relatively easy step as we will see in section 6.3.

4.11 Microsoft Deployment practices

In this chapter we will discuss what Active Directory is, what functions it has concerning automated deployment of software within a company domain. This issue is of importance in order to know what possibilities are out there that might help Océ customers deploy our software more easily. In this chapter we only dig into the aspects of deployment and the directly related concepts, we do explicitly *not* explain the whole concept of Active Directory, for that would require a study by itself and would be completely off-scope given the project limits.

4.11.1 What is Active Directory?

Active Directory is a virtual directory existing on a company network level that contains objects describing organizational entities.

Active Directory consists on the highest level out of three categories:

- *Resources* (printers, servers, computers and the like);
- *Services* (like e-mail for example)
- *Users*.

Each object within the Active Directory has a unique name and objects can be grouped into organizational units (OU's).

4.11.2 Group Policy Objects

Group Policy Objects (GPO's) are objects that contain rules that can be applied to an object within the AD. Network administrators can use a server edition of MS Windows to configure these GPO's. Within such a GPO, the administrator can define – for example a workstation – the standard status of software on such a machine. This can be done in two ways:

- Predefinition in the AD;
- Publishing via the AD.

Both approaches are now discussed in the following two paragraphs.

4.11.3 Predefinition in the AD

When using a predefinition in the Active Directory, the network administrator configures for a particular (group of) machine(s) what software should be installed by default. When the particular machine is then started, the requested software is automatically installed using administrative rights. For the network administrator it is necessary to configure the GPO using an MSI (Windows installer file). Next to the MSI, an MST can be used, which is a file to make external modifications to the MSI file. Other installation types than MSI are not allowed.

Given the fact that the installation is executed as a system process, there are no possible problems with not having sufficient rights to complete the installation.

4.11.4 Publishing via the AD

Next to predefining the software status on a workstation, the network administrator can also *publish* applications. This means that the application is not installed automatically during start-up like is the case using a predefinition, but shows up in the well-known list “Add and Remove programs”. The logged on user can then choose to install the software.

The installation will then run in a separate process with elevated rights (at least, if the network administrator configured it to be so) which avoids problems with insufficient rights *in most cases*. The problem with this kind of deployment is that the installation can never run as local administrator or system process. This means that installations that require access to core files of the operation system might potentially fail.

The configuration of such a “publishing” can be done with all types of installers (not necessarily an MSI), although the installations must be encapsulated in a so-called ZAP file which contains the parameters and so on for the installation to start and to proceed as desired.

4.12 JSP for web interfaces

As a final step of the project, the developed application must be integrated into the default environment for submitters. We will focus on Tomcat web servers and an interface based on JSP (the Java based server side scripting language). JSP has many complex features that we can impossibly discuss within the scope of this project, but we need to focus on the basics of JSP here.

JSP – or *Java Server Pages* – belongs to the category of server-side scripting languages that enable dynamic website generation. The basic idea of this server-side scripting is that a user can be presented with a customized web page, depending for example on his machine, like the operating system or browser. But also to give access to more advanced functions that require supplementary libraries that are usually not present on client machines.

JSP is the Java dialect in this world of server-side scripting languages. Next to JSP there exist many others which we can impossibly cover in this document, but to give a small overview of the main players in this field, the following table could give some insight in this (classified on popularity):

Abbreviation	**Full name**	**Developed by**
PHP	PHP Hypertext Processor	PHP Group
ASP	Active Server Pages	Microsoft
JSP	Java Server Pages	Sun

For the integration, we need to focus on special extensions that exist within JSP, called tag libraries (popularly abbreviated as “taglib”). A tag library is a library containing predefined tags that can be used, preserving the XML structure of the output. It is a way to separate markup output (which is usually in HTML) and programming code (in this case in Java).

In JSP a tag library descriptor (TLD) exists, which enables the programmer to make the connection between the source code (usually residing in Java classes) and the tag to be used. This TLD file is by itself an XML file that describes the link to the related source code, but also the behavior of the tag with its parameters (if any).

By default, a tag library is included with Tomcat (the Java based web server) which carries the name JSTL (JSP Standard Tag Library). This tag library consists out of a large collection of tags that enable programmers to access most of the programming basic constructs by using tags. Below an example is shown of using this JSTL. The declaration of the tag library is omitted, because Tomcat includes the JSTL by default. When writing an own tag library, you need to include it with a special declaration.

```
<html>
  <head><title>JSTL Demo</title></head>

  <body>

  <% String s = "test"; %>

  <c:if test="${s == 'images'}">
    <p>Test succeeded</p>
  </c:if>

  </body>
</html>
```

Declaration of an arbitrary tag library in a JSP file:

```
<%@ taglib prefix="t" uri="http://www.my_domain/tools" %>
```

JSP in general and tag libraries in particular are covered in depth for the interested reader in appendix C.

5. Environment

Having discussed the research and investigation, we are left with one major subject before we can turn to practice by means of prototyping. This subject is encapsulated by the term "environment" where we will focus on the structure of the application. This structure is a vital issue concerning deployment using the in section 4 identified technique called Java Web Start.

5.1 Application investigation

In our discussion on the World of Submitters, we explained the global functions of such a program. Given the fact that this version of the graduation report concerns the public summary, we will discuss the environment of submitter applications in general.

We will take as our goal in this discussion the creation of a demonstrative (final) prototype that within the educational purpose of this document could be seen as the end product.

5.1.1 Component analysis on size

The first direction in which one has to investigate the composition of an application to deploy is in its size. To perform this analysis we simply take the classic installer and let it install the application on the hard disk. Then we look at the directory structure put there. In this way we were able to create an overview of the package structure of the application.

5.1.2 Component analysis on the functional level

Next to looking at the size, we can also look at the internal structure taking the architecture into account. This architecture can tell us something about the internal structure of the application, rather than the external structure we saw in the previous paragraph where we looked on size. This architectural structure tells us how the application is set up into functional components. These components can help us then to identify logical packages of the application.

5.1.3 Endorsed directory principle for extensions

Java enables developers to 'patch' the JRE installed on a client computer. This is called within the Java world "endorsed directory". This is a subdirectory of the main folders of the JRE where you can put libraries that should override libraries installed by default (in general you put updated libraries in there). The problem with using this principle is that we need administrative rights using the classical approach (classic installer). One could think of the following question: "hasn't Sun put this feature into Java Web Start then?"

The answer is – perhaps surprisingly – no. In order to sustain this statement, we will look at two statements made by Sun:

Tim Quinn (Sun developer) states on his blog the following:

"The whole point of Java Web Start in general, and the GlassFish support of it for launching app clients, is that you want to avoid having to prepare the environment on the client systems ahead of time. You just click and launch, and everything that user needs is downloaded (if it is not already cached on the client system). Requiring you as a developer or administrator to go to every client system to install the 2.1 libraries as an endorsed extension runs completely counter to the intent and basically robs the benefit of the Java Web Start launch feature of any value." [TQB]

Furthermore, Kohsuke Kawaguchi (Sun expert) states during an "Ask the Experts" session the following:

"As you probably already know, Web Start does not have any support for the endorsed directory mechanism. We should be talking to the Web Start team about this, but anything they do will only be available in Java SE 7, so it's not any time soon. There are a lot of pains in JAX-WS 2.1 + Java 1.6, and this is one of them." [ATE]

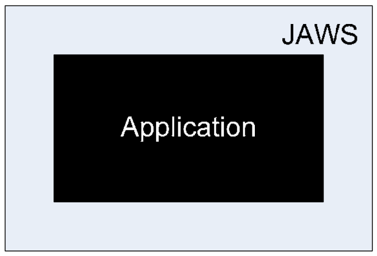
This means that endorsed directory usage within Java Web Start is not possible for the moment. Some supplementary Java components are also available within a Java Web Start package which in most cases should solve the problem. This — however — is still far from a regular situation. When developing for deployment using Java Web Start, this constraint must be taken into account.

5.2 Deployment scenarios / functional axis

Profound research has been conducted in order to find out what feasible deployment scenarios would exist using JAWS. These scenarios are discussed in the following paragraphs; each with their advantages and disadvantages (pros and cons). In this section we will first look on the functional axis, without looking at how to store things (we will look into that issue in the next section). Therefore when you see "DISK" in figures below, the actual location on the disk can be any arbitrary folder.

5.2.1 100% JAWS

The easiest option from a developer point of view is the 100% JAWS solution. This scenario consists out of putting the entire application into Java Web Start.



Pros:

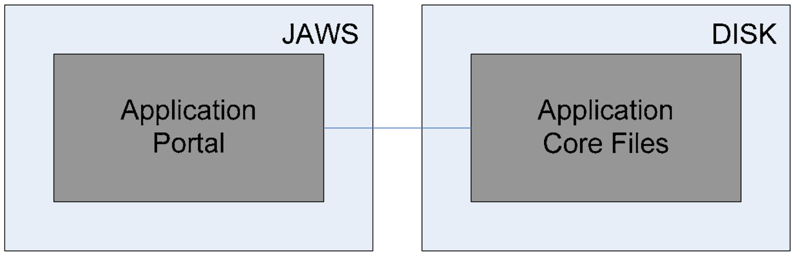
- Little to no adaptations needed in order to use this scenario with the current situation of the application;
- Updates completely handled by JAWS;
- Uninstall is handled by JAWS.

Cons:

- Large amount of space required within the JAWS cache (about 50 MB);
- Application available for current user only (except when using the system cache instead of the user cache).

5.2.2 Extract core part outside of JAWS

This scenario consists out of putting a core part (the part of the application that is *least* likely to change with updates) somewhere on the local disk drive. The parts of the program that are subject to change stay within JAWS. The result is a division of the currently existing source code.



Pros:

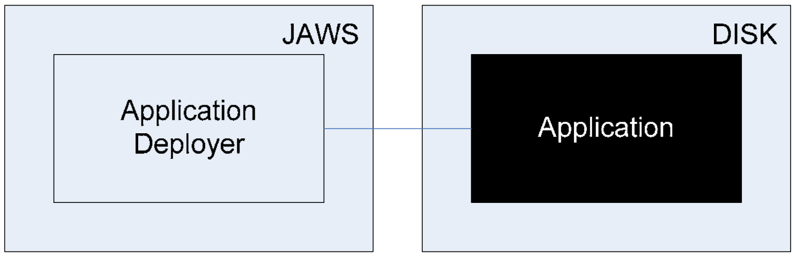
- JAWS cache use is significantly reduced to about 10 MB remaining versus 50 MB of complete residing;
- We can still benefit from the update system of JAWS.

Cons:

- If an update is necessary on the core files side, this is potentially complicated;
- Source code changes (although not dramatic) are necessary;
- Uninstall of the non-JAWS part will need extra programming.

5.2.3 Use JAWS as a deployment engine

The last scenario on the functional axis is to use (or abuse?) JAWS for deployment only. In this case the main application will be installed by a JAWS-based secondary application, on a certain location on the local disk drive.



Pros:

- No adaptations to the source code are needed for we only touch deployment;
- Very little usage of the JAWS cache is required (just the deployment application which could normally be kept under 1 MB).

Cons:

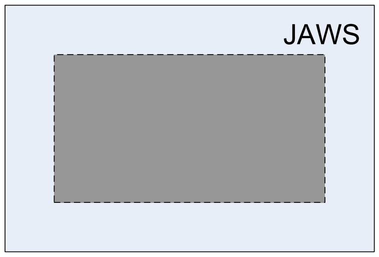
- The update system of JAWS has to be extended with an update system of our own;
- Uninstall of the non-JAWS part will need extra programming.

5.3 Deployment scenarios / storage axis

After the three scenarios on the functional axis where we looked how the functional parts could be organized, we will look now in which configurations we can put them. This is particularly important for the scenarios discussed in sections 5.2.2 and 5.2.3 where we have a non-JAWS part. An important issue to take into consideration is that the organization of the JAWS cache is different under MS Windows Vista than under MS Windows XP. In XP, the cache is residing in the *roaming* part of the user profile, where in Vista this is *not* the case.

5.3.1 COMPLETE JAWS CACHE RESIDENCE

This is the easiest configuration; I refer to 5.2.1 for details on this.



Pros:

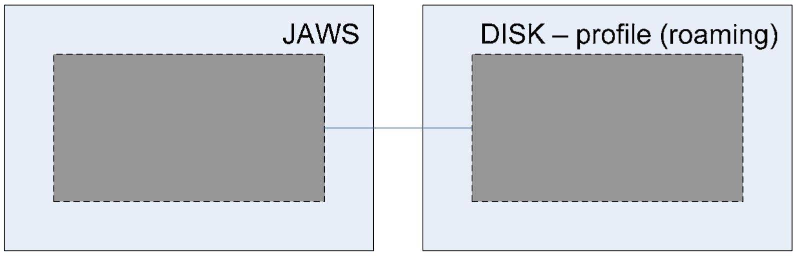
- Everything is handled by JAWS and thus – by default settings of JAWS – available everywhere because of the roaming user profile location in Windows XP. For Windows Vista the cache is outside of the roaming part.

Cons:

- Network load might increase drastically: about 50 MB of files need to be transferred each time;
- Installation is necessary for *each* user on the machine (so we could get x duplicates for x users on the same machine).

5.3.2 STORAGE IN ROAMING CURRENT USER PROFILE

This is the first storage option for the setup of section 5.2.2 and 5.2.3. Here we focus on the option of storing the non-JAWS part inside the roaming current user profile. This means that if the user has a network account and he logs in on another (physical) computer, all files will be copied to that computer. For JAWS this is already the case (for Windows XP only), because the JAWS cache is stored in the roaming current user profile by default.



Pros:

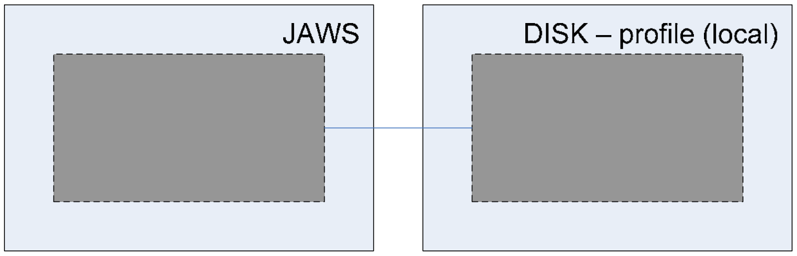
- Continued availability everywhere the user wants to use the application when using Windows XP (does not count for Vista).

Cons:

- Network load might increase drastically: about 50 MB of files need to be transferred each time;
- Installation is necessary for *each* user on the machine (so we could get x duplicates for x users on the same machine).

5.3.3 STORAGE IN LOCAL CURRENT USER PROFILE

This option is similar to the roaming one with difference that in this case the files are not put in the roaming part, disabling portability towards other computers within a network. JAWS cache however is in the roaming part (for XP, non-roaming for Vista), which might give the user the impression that the application is available everywhere, where this is actually not the case.



Pros:

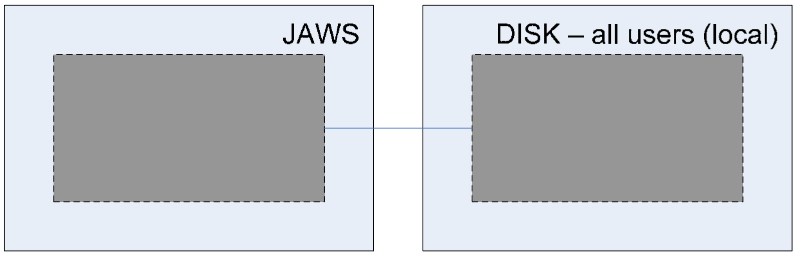
- Less network load because the (core) files are left on the local machine.

Cons:

- Some extra handling needs to be done when running into the situation where the JAWS cache is copied, but the (core) files are not accessible;
- Installation is necessary for *each* user on the machine (so we could get x duplicates for x users on the same machine).

5.3.4 Storage in (local) All Users profile

This option is a result of an attempt to find a way to solve the problem that the application must be downloaded and installed for every single user that uses the machine. Therefore this option does not store the (core) files in the current user profile, but in the All Users profile. As a result, the (core) files don't need to be copied each time for every user.



Pros:

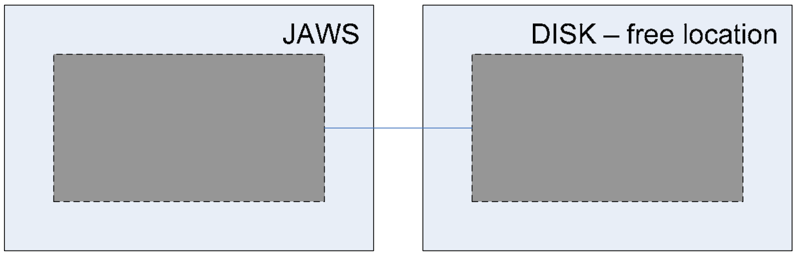
- Less network load because the (core) files are accessible for all users;
- Less disk storage needed because the (core) files are stored only once for all users.

Cons:

- The solution is not portable, where the JAWS cache still is (for XP only).

5.3.5 Storage on a free to choose disk location

This solution tries to get completely out of the user profiles and lets the user choose a directory somewhere on the disk to install the (core) files.



Pros:

- When properly used, it could be used for all users;
- Less network load as a result of the first reason given.

Cons:

- The solution is not portable, where the JAWS cache still is (for XP only).

5.3.6 STORAGE WITHIN THE JAWS SYSTEM CACHE

Analogous to each of the different options given in sections 5.3.1 to 5.3.5, one may choose to put the application not in the user cache of JAWS, but in the system cache. This system cache is available to all users on the particular machine for running the application, but normal users may not modify anything.

Pros:

- Application available for every user;
- No (unnecessary) copies;
- Normal users cannot "mess up" the application.

Cons:

- Users cannot perform updates;
- All users are obliged to use the same version of the product;
- The installation (and also updates) needs to be performed by the administrator.

5.3.7 SUMMARY

Below you find a table with the most important properties of each of the options on both axes that were discussed.

A = JAWS only	(5.2.1)
B = JAWS + external core	(5.2.2)
C = JAWS for installer + full program externally	(5.2.3)

1 = JAWS only	(5.3.1)
2 = JAWS + roaming current user profile	(5.3.2)
3 = JAWS + local current user profile	(5.3.3)
4 = JAWS + all users profile (local)	(5.3.4)
5 = JAWS + free disk location	(5.3.5)

Property	A+1	B+2	B+3	B+4	B+5	C+2	C+3	C+4	C+5
No code adaptations	x	-	-	-	-	x	x	x	x
Updates automatically	x	-	-	-	-	-	-	-	-
Uninstalls automatically	x	-	-	-	-	-	-	-	-
Low JAWS cache usage	-	x	x	x	x	x	x	x	x
Available for all users	-	-	-	-	-	-	-	x	x
Full portability (roaming)	x	x	-	-	-	x	-	-	-
Single local install (no dup)	-	-	-	x	x*	-	-	x	x*

* = depends on the installation procedure

Remark: roaming property based on findings within Windows XP. For Windows Vista portability is only achievable when we choose for the option C+5 where the “free disk location” is a roaming user profile.

5.4 Installation mode interoperability

With a change in the installation mode, the question arises whether or not the new way the application will be deployed is compatible with the existing installation modes. This classic installer exists in the form of a .exe file. In the following two paragraphs we will discuss briefly the classic installation procedure and the new installation procedure using JAWS.

5.4.1 Classic installation

The classic installation of an application follows the following (classic) deployment strategy:

- Copy program files to the %Program Files% directory;
- Optionally register file extensions in the Windows Registry;
- Create shortcuts on desktop and start menu.

This deployment strategy requires local administrative rights and can be performed in several ways:

- Manual installation using the installer acquired via CD or via the internet;
- Automated installation by the IT department using Group Policy Objects;
- Command line installation;
- All three above can be executed for one user or multi-user.

5.4.2 JAWS Installation

The JAWS installation follows a fundamentally different approach than its classical counterpart. This approach can be summarized as follows:

- Program files are stored in a local cache;
- File extensions are registered in a virtual way (at least not permanently);
- Shortcuts on desktop and start menu are created automatically.

This deployment strategy always works, regardless the rights level of the logged-on user. The program is stored entirely in a local cache of JAWS that resides in the current user profile.

5.4.3 UPDATING

Updating the software application is very different in both approaches. When following the classic approach, in most cases the existing installation is simply overwritten by the new one. This is a rather drastic approach which is not necessary most of the time. Sometimes, upgrades do only replace specific files, with leaving the majority of them untouched.

When using JAWS, this idea changes completely. JAWS uses a cache which is used to store the program files. These program files are stored in a per file way, which makes the task of updating easy and efficient. Furthermore, this system permits to easily include extensions necessary for the program to run.

6. Experimental prototyping

Discovering possibilities, trying out usability, investigating how technologies work in practice… these are the main issues that form the experimental prototyping phase. In total four major prototype versions were made along two prototyping axes which will be described in section 6.2. The experimental prototyping phase was probably the most moving phase of the entire project that showed whether or not certain approaches were feasible.

6.1 Purpose and introduction

Creating one or more prototypes is in general a well-known included part of the software development process. Its purpose is in general to determine whether possible approaches to a solution are feasible or not. For this project, the experimental prototyping was also aimed at discovering possible ways of organizing files in the JAWS cache or somewhere on the local disk. As in the previous section possible architectural setups were discussed, they could now be put into practice in order to determine which one works best and to proceed with towards a final solution.

But the prototyping phase had also another very important purpose which was to get used to this relatively new technology called Java Web Start. When new to this technology it is a rather complicated task in setting it all up successfully. The small sample application described in section 4.8 showed already that creating JAWS applications that need full permissions is not a straightforward business. When extending this on a scale of a middle-large applications, the task only gets more complicated. The experience gained through this prototyping phase was therefore vital for creating a final solution successfully.

There were four prototypes built that could be placed on two axes with respect to incorporation into JAWS. These prototypes are named P1, P2, P3 and P4. There is a whole spectrum that could be used between the extremes of each axis, described and visualized in the following sections.

In order to understand why these approaches were taken and how the decomposition was made in terms of program pieces, we must analyze the decomposition into components.

With the construction of the prototypes we try to identify the most convenient solution. In general this comes down to where we actually want to place our application on the prototype axis as defined in section 2.1, but we will anyway sum up the key objectives here:

- What is the most convenient **level** of putting (groups of) components on the local disk;
- If we put files on the local disk, then **where** to put them;
- What **versioning** techniques can be used for files put on the local disk;
- How to keep the **installation easy**.

For each of the prototypes discussed in section 6.3 to 6.6, we will discuss each of these aspects in order to get to the most convenient prototype that could actually be used for the final recommendation.

6.2 Prototype axes

In order to classify the prototypes, two axes have been identified during the prototyping: an axis describing how much of the application will reside in the JAWS cache and a second axis describing the granularity of the components put on the local disk in order to facilitate optimal updating and versioning.

6.2.1 The JAWS cache axis

The first axis identified is determining how much of the application will reside within the JAWS cache. This is visualized below:

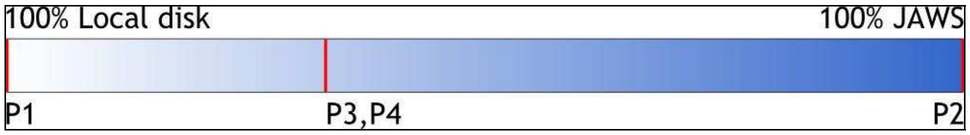


Figure 8: first prototype axis

On the left we have a situation where we put everything on a specific local disk location and thus using JAWS uniquely for deployment/install. This approach was followed in the first prototype (P1). On the far right side we have a situation where we use the caching mechanism of JAWS completely by putting the entire application into the JAWS cache. This approach was followed in the second prototype (P2). Finally, in the third and fourth prototype (P3 and P4) we followed an approach to put only Java-based components into the JAWS cache and putting everything else on a specific disk location.

6.2.2 The granularity axis

The second axis identified is the granularity axis, determining the granularity of the packages deployed on the local disk. The smaller the packages (high granularity), the more efficient updates and versioning can be performed. This axis is visualized below:

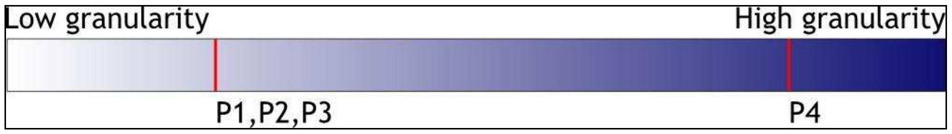


Figure 9: second prototype axis

On the left we have a situation where we have a low granularity that translates into a situation with large packages. The inconvenience of this property is that when updating, large amounts of data need to be replaced, even when within such a package only minor changes were made (as was the case for the first three prototypes). High granularity was achieved in the fourth prototype (P4) that resulted into a much more optimal way of updating and versioning.

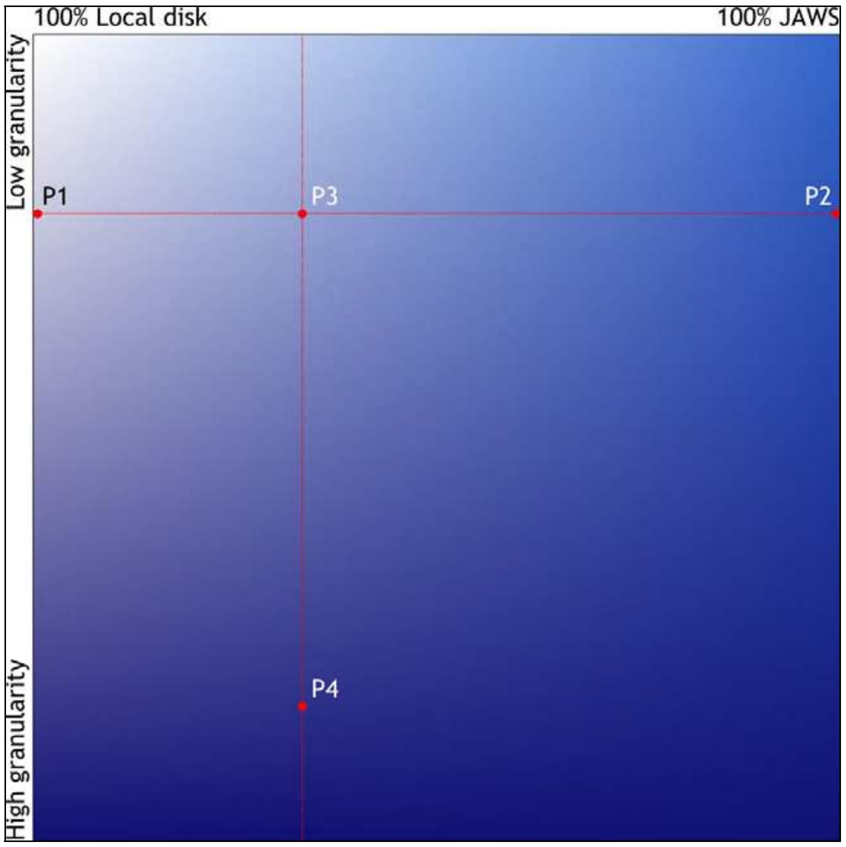


Figure 10: both prototyping axes combined

In order to give a full view of the prototyping axes, the figure above showing both at the same time with the location of the four prototypes is presented.

In the following sections we will discuss the four prototypes with their specific issues and what the results were of these.

6.3 Prototype P1 (100% local disk)

In the first prototype we tried to create a sort of installer that runs in JAWS and that copies files from an included jar-file into a particular directory on the local disk.

The first question that arises in this technique is of course: "Where to put the files?" An important issue that we need to keep in mind is the key objective of the whole operation of getting a web-based solution for an application which is that the installation should be performable by each user type, also for users with restricted rights. This already rules out certain disk locations like the typical Program Files and Windows directory. But next to these locations, administrators could have given the user no write access to any other folder on the hard disk (this situation is – I admit – very rare, but not impossible). Fortunately there are some locations that are *always* accessible. The 'All Users' profile and the current user profile are always available for reading and writing data. One might choose to put everything into the current user profile in order to keep the files for the current user, but for optimization reasons we chose for this prototype to put everything into the ‘All Users’ profile in order to avoid multiple copies of the application on the local hard disk.

6.3.1 File organization and copying

The primary point of interest is how we organize the files in jar-files for the installation and how we get them out for copying them into the destination folder.

In the prototype we have chosen to first create a jar-file that contains all the files that must be deployed, while keeping the directory structure intact. Then, for JAWS, we put this jar-file together with the deployment application class-files into a new jar-file which is signed.

In order to be able to extract the files from the initial jar-file, we define the following statement:

```
InputStream fis = getClass().getResourceAsStream("myjar.jar");
```

Using this InputStream object, we copy its content towards a temporary file using the following code:

```
temp = File.createTempFile("temp",".tmp");
temp.deleteOnExit();
FileOutputStream fos = new FileOutputStream(temp);
```

With a standard binary copy towards the temporary file, we can now use this file for normal jar-operations as follows:

```
jarPubSel = new JarFile(temp);

Enumeration enmEntries = jarPubSel.entries();
while (enmEntries.hasMoreElements())
{
        strEntry = (String)(enmEntries.nextElement().toString());
        copyFromJar(strEntry, strDestination);
}
```

In this code we use a self defined procedure called `copyFromJar` that takes (input location, output location) as arguments.

This copying procedure is analogue to the copying procedure of the jar file itself. An important detail is the creation of directories where the files need to be put into. This can be done using the following procedure:

```
intIndex = strDestination.lastIndexOf("/");
strDir = strDestination.substring(0, intIndex);
boolean success = (new File(strDir)).mkdirs();
```

Where strDir contains the value of the sub directories that should be created. The mkdirs method then tries to make the entire directory structure given and returns *false* when this was not possible for some reason (like not having the necessary rights or that the directory/directories already exist).

6.3.2 Running the application and removal

Apart from copying files, we must also be able to run the application, as well as update and remove it. In this prototype updating is equal to complete removal and re-copying, therefore we do not discuss this issue separately here. Application launching and removal needs some discussion however.

Running the application is a rather simple operation and can be done by using the Runtime object. The following code runs an arbitrary exe file:

```
runtime.exec(strProfilePath + "\\my_exe_file.exe");
```

Where strProfilePath denotes the location of the location of the folder where the application was copied to in the All Users profile.

Application removal is a little bit more difficult and in this prototype we haven't provided a complete removal. Files are removed, but the directory structure (though empty) remains. It should not be difficult however to create this missing extension.

For removing the application, we first need to delete the files in the directories created. This is a rather straightforward operation that can be performed using the following code:

```
private void deleteDirectory(File path)
{
        if (path.exists())
        {
                File[] files = path.listFiles();
                for(int i=0; i<files.length; i++)
                {
                        if(files[i].isDirectory())
                        {
                                deleteDirectory(files[i]);
                        }
                        else
                        {
                                files[i].delete();
                        }
                }
        }
        System.out.println("Deleting files...");
}
```

6.3.3 How far are the objectives met?

For each prototype we will discuss in the last subsection how far the objectives we defined in section 6.1 are met.

[1] Is the location on the prototype axis convenient?

Not really. Using JAWS just for installing purposes is a waste of the supported functionality of JAWS. The result of putting everything on the local disk like classic installations leaves us with a complete versioning control that must be carried out for all components.

[2] Where to put the files on the local disk?

We chose to put them in the All Users profile in order to avoid multiple copies of the same program. Putting everything into the All Users profile however, leaves us with the disadvantage that the same program with the same settings will be presented to each user. I.e. users cannot make personal configurations, nor choose their own set of different versions of the product.

[3] What versioning techniques can be used?

In this setup the versioning needs to be done entirely on our side. This creates a heavy overhead of administration which is not desirable. Especially for the part that is most likely to change this is potentially problematic. Versioning is not supported by the prototype, because it is far too complex to put into a prototype.

[4] Is the installation easy?

Yes, however the installation is not automated in this prototype. But the user is presented to a simple user interface that leaves no doubt and that performs the installation as good as automatically once chosen for one of the four basic operations (install, update, remove and run).

Furthermore, to refer to the key issue, this prototype can be successfully executed by any type of user under both Windows XP and Windows Vista.

6.4 Prototype P2 (100% JAWS)

In the second prototype we turn towards the other end of the prototype axis by putting the entire application into the JAWS cache. This approach was significantly more difficult than the 100% local disk approach, a result when starting with an application that in the original situation was installed onto the local disk. An approach where we do not touch this structure is per definition the easiest one.

An interesting point of this approach is that it is not necessary to write a new program that performs the actual deployment. This approach where we put everything into the JAWS cache has the great advantage that JAWS is doing all the work for us. In fact, this approach lets us profit maximally of the technology JAWS offers. The only thing we have to do is to create the signed jar file that JAWS is going to use, as well as the JNLP file that defines how the application must be initialized and … — this is the most irritating part — we need to perform some source code adaptations in order to make our application work.

6.4.1 JAR ORGANISATION

For this prototype we chose a different approach of ‘jarring’ then in the first prototype. We didn’t put everything into one big jar file. A reason for this is that the application already uses jar files that could be used directly. The only thing that was necessary to do was to sign them.

This might appear simple and straightforward, but it turned out that there are some tricky issues involved. If you use third party jar files, you could find yourself in the situation where a jar file is already signed. If you try to (over)sign this jar file with your own signature, there is no message that tells you that this is not possible. Actually, the jarsigner application of Java simply signs your jar file and doesn’t complain about the existing signature. However, a problem arises at the moment you want to use the jar file by JAWS. Then JAWS tells you that not all jar files were signed by the same certificate. Thus, in fact, resigning does **not** override the previous signature. The signature must first be removed (which is easy to do: just unzip the jar file, delete the META data and the manifest and reconstruct the jar).

6.4.2 JNLP Configuration

The JNLP configuration gets more complicated when you want to have a specific deployment. JNLP is specially developed for this goal so there is no problem with that. In fact you could compare configuring a JNLP file with some form of scripting in order to guide the application to the direction you want.

In order to have a workable JNLP configuration where we can actually use native code (that is, for example DLL's), we need to use a special specification. First of all you need to specify for which operating system the libraries are intended and then using the `nativelib` tag you must specify for JAWS that it concerns native code. For this prototype this looks like:

```
   <resources os="Windows">
        <nativelib href="core.jar"/>
   </resources>
```

The core.jar file contains win32 DLL files. But doing this does not suffice. The problem is that having the DLL's visible for JAWS, does not include that they are also visible for the application. So if you would try to run the application in this configuration you would get errors like that the program cannot find the DLL files. In order to solve this problem and make the DLL files available for your program, you need to specify these DLL files in the code of your program using the loadLibrary function of Java. Furthermore you need to specify these libraries **in order**, which means that you need to do a dependency analysis of your DLL files used and you must give the loadLibrary commands from the lowest to the highest level (thus start with the DLL that has no dependencies and then moving upwards towards those using the DLL). This issue has as a consequence that you need to make changes in the **existing source code** which could be seen as a disadvantage, however it should be quite obvious by now that no changes at all is an almost impossible exercise.

For the application source code we needed to include a new class. This class has as function to prepare it in such a way that it launches correctly.

6.4.3 Changing system property values about Java environment

Sometimes it might be necessary to change the values of the environment variables in JAWS. JAWS does initialize for example the user.home variable to be the desktop folder of the current user profile. This is not always what you might want. If you need to change such a setting, this can be done using the following directive:

```
<property name="key" value="overwritten"/>
```

This directive should be placed in the <resources> scope.

6.4.4 How far are the objectives met?

[1] Is the location on the prototype axis convenient?

No. The reason for this is that using the JAWS cache for the entire application puts a heavy load on to this cache (about 50 MB). This means that if users have disabled the cache (which is supported by JAWS), the 50 MB need to be downloaded each time. Furthermore, for Windows XP when working in an environment where you have a central server with workstations, the JAWS cache is located in a roaming user profile which means that it will be copied back and forth to the server each time a user logs on or off. On the other hand, for the developer this is a very nice solution because there are very little adaptations necessary on the source code side of the story and (almost) no additional tools have to be written for putting files in the right place on the local disk. The program is further guaranteed to work, because it doesn't have to be written somewhere, since JAWS is doing all the stuff for us (except lack of disk space of course).

[2] Where to put the files on the local disk?

In principle, when no further data files are required, this is no issue. However, if your application has data files that are necessary to run and that must reside on a disk location in order to be found, you will need to write a procedure that copies these files to the desired location at startup of your program. These data files themselves could then be saved in a separate JAR file.

[3] What versioning techniques can be used?

In this prototype no versioning is required because JAWS is doing it all for us.

[4] Is the installation easy?

The installation is a straightforward JAWS initialization of a program and thus very easily executable by a user having little to no knowledge of installing applications.

Furthermore, to refer to the key issue, this prototype can be successfully executed by any type of user under both Windows XP and Windows Vista.

6.5 Prototype P3 (balanced local disk/JAWS)

The third prototype is not much about coding yet another prototype, but more an investigation in finding the right balance between putting parts in the JAWS cache and putting files in the current user profile. The question is mainly: "What to put where?" For the third prototype we have decided that we stick to the JAWS cache for java. That means that only Java components will be deployed by using JAWS. Additional, non-java components (like win32 DLL's) and supporting data files will all be put into the current user profile. But it doesn't end just with putting files in the user profile, because we want to give the user more options than just running the newest version of our application. We want to offer a versioning system that enables each user to use his own preferred version of the application.

6.5.1 Controlling JNLP

Next to versioning issues we also want to optimize the deployment which needs usage of an extra feature of JAWS, the JNLP API. This API enables the programmer to control the way JNLP (and thus JAWS) performs its common tasks, like for example the way resources are cached [JNLP-API].

We will need to use this API in order to remove items from the JAWS cache once we have transferred them to the directory of the current user in his profile (otherwise we would store the data twice which is not desirable of course). The way of doing this is as follows:

First we need two imports:

```
import javax.jnlp.*;
import java.net.*;
```

The javax.jnlp package is not standard included into java, you need to specify it in the class path defining the location of the jnlp.jar file which at its turn *is* included in the JDK versions 1.5 and higher.

Next, we need to define the download service object and fetch the available service into it from the Service Manager. This procedure is depicted below:

```
DownloadService ds;

try
{
        ds = (DownloadService)ServiceManager.lookup(
                                    "javax.jnlp.DownloadService");
}
catch (UnavailableServiceException e)
{
        ds = null;
}
```

Finally, we need to identify the resource *exactly on the same address as mentioned in the JNLP file*. After that it can be removed using the remove resource routine from the Download Service object:

```
try
{
        URL url = new URL("http://localhost:8080/oce3/pokerst.jar");

        if (ds != null)
        {
                ds.removeResource(url, null);
        }
}
catch (Exception e)
{
        // error handling code
}
```

An important thing to note is that if the given URL does not exist, the Download Service object does *not* raise an exception about it. A developer could therefore be tricked to think that the application works correctly. But if the resource is not identified correctly, it will simply stay within the JAWS cache. In order to check whether the resource has been correctly deleted, it suffices to consult the cache visualization of Java and check the size of the application. Even if the window is opened, the size is updated automatically when the program removes it.

6.5.2 ARE THE OBJECTIVES MET?

[1] Is the location on the prototype axis convenient?

Yes. By putting a large part of the program resources outside of the cache, the cache usage is significantly reduced. We also do profit from the versioning that is done by JAWS itself, leaving the part that is most likely to change over to JAWS. For the parts outside of the JAWS cache we need to do some versioning by ourselves. The current idea is to create a new directory for each new version and put the files in the particular subfolder. However, estimations are that changes in this part are much less likely than changes in the Java-based part.

[2] Where to put the files on the local disk?

For accessibility reasons the data and core files are put into the current user profile.

[3] What versioning techniques can be used?

The versioning chosen is done per subdirectory. If the required version does not exist, it is copied to the correct location from JAWS, otherwise it is just used.

[4] Is the installation easy?

The installation copies the necessary files and performs the checks automatically. For the user there is no difference in the installation session and the normal application run session, except that in the first situation the start up takes a little bit longer due to downloading and copying.

6.6 Prototype P4 (balanced, with high granularity)

The fourth prototype is heavily based on the third prototype and is part of the chosen path towards a definitive solution. In the third prototype we determined the global set up of packages. The fourth prototype continues in this direction and splits up resources in smaller parts.

Next to more granulation we also included the notion of shared and private resources. Shared resources are stored in the All Users profile, private resources in the current user profile. The whole idea is to save disk space and reduce download time for most of the resources only need to be downloaded once. This approach assures rapid start up times for the application as well as efficient storage. With this effect also comes that updates can be processed much more efficiently. Instead of replacing complete packages of many megabytes in size, we are able to replace much more precisely a component without re-downloading 90% of the data we already had.

6.6.1 Deployment configuration

In order to be able to configure the subdivision in an easy way, an external configuration file was created. This is especially important for future use for the creation of updates, but also during the prototyping phase in order to find a suitable level of granulation.

Two things immediately need some further explanation: why do we use an encapsulated jar file and why do we use global variables. This is explained below.

In order to deploy files easily towards a target directory, the easiest way is to have a jar file you can simply unzip to that particular target. Unfortunately the jar used by JAWS is already unzipped by JAWS itself and cannot be accessed through a JarFile object. Thus we are forced to retrieve an encapsulated jar file using the standard getResource routine from JAWS.

The use of global variables is another story. System variables for Java can be changed at runtime and also by means of the *property* declaration in a JNLP file, but the annoying thing is that the JRE does **not** take them into account. So you can change whatever you want (and even by forcing an output see that the change actually took place), but when it comes to finding resources it turns out that the changes are not taken into account.

So in order that JAWS finds the resources we stored on the local disk exactly there where we want JAWS to find them, we are obliged to use explicit, absolute paths. These are – of course – constructed in a relative way by taking the paths to the *All Users* profile or the current *User Profile*. For each of the items these variables are set using the System.setProperty(…) function and these are in the application source code at the locations where necessary retrieved using the System.getProperty(…) function.

6.6.2 UNINSTALL

The fourth prototype differs also from the third one on the point of uninstall. The 3rd prototype did not have any uninstall functionality; the 4th prototype has a fully functional uninstaller. The uninstaller removes *all* resources of the application in the *All Users* profile as well as in the current *User Profile*.

Furthermore, all resources (in fact all jar files that are in the JNLP) are removed from the JAWS cache in order to clean up the JAWS cache as well. This operation is almost identical to the manual uninstall in the Java Control Panel. The only difference is that the reference to the application remains present in the list, but the application doesn't occupy any disk space anymore.

6.7 Applet prototype

Originally, the applet option was dropped because it would be unfeasible due to long loading times and the fact that a large application should not reside in one or more applets for practical reasons. At the end of the project, the remaining time was used to perform an investigation into this matter to really see what the effect on the loading time would be when we would try to build an applet containing (the core part) of the application anyhow.

Internally, this became our prototype P5, where we adapted more or less at will parts of the application such that it would run within an applet in a mode where a user can only directly select its documents for printing and being able to send it to a printing device. This last item — being able to actually print — was left out, because it appeared too complicated in the time available. The application was therefore made to run in the demonstration mode.

The prototyping performed with this "light" version proved our initial point that it would be too heavy for an applet-based approach, as loading times exploded (especially under Windows Vista, where the applet loading mechanism has been updated in order to make it work again in the JRE beta 6 update 10 version). Also the "light" version still needed a JAR file of about 47 MB. As a result of the necessity for full access to system resources, this JAR file must be signed (same procedure as for JAWS). The result is that at launch, all files inside this archive file must be checked in order to determine whether their digital signatures are correct. This takes a lot of time and seriously reduces the feasibility of this solution.

6.8 Prototype comparison

In order to be able to choose the prototype that serves us best, we need to identify a collection of criteria in order to measure the quality of each of the prototypes.

This section presents these criteria and applies them to the prototypes discussed in the previous sections in order to come to a concluding advice on which way to go.

6.8.1 Comparison criteria

The criteria that will be used to measure the quality of the prototypes are defined as follows:

- CC1: Ease of installation (measured by the minimal number of actions required by the user to complete the installation);
- CC2: Hard disk usage when having *n* users and *m* versions (measured by the number of megabytes occupied) with:
  - s = full program size (= sp + su);
  - sp = size of program core files;
  - su = size of program user files;
- CC3: Installation duration (measured in seconds counting from the click on the installation link to installation completion);
- CC4: Application start up duration (measured in seconds);
- CC5: Impact on existing code (measured by the amount of changes necessary in the existing code in order to have the application running);
- CC6: Ease of adaptation (measured by an estimation of the time required to create a new version or update).

6.8.2 Comparison table

In the table below the prototypes are compared using the definitions of the comparison criteria:

	100% local disk (P1)	**100% JAWS (P2)**	**Balanced (P3)**	**Granulation (P4)**	**(applet)**
CC1	2 actions	2 actions	2 actions	2 actions	1 action
CC2	m*n*s	n*s (no per user versioning possible)	m*sp + n*su	≤ m*sp + n*su (due to granulation)	n*(s)
CC3	50 seconds	35 seconds	41 seconds	41 seconds	55 sec
CC4	9 seconds	9 seconds	9 seconds	9 seconds	
CC5	no impact	medium (2 classes)	medium (2 classes)	medium (2 classes)	high
CC6	half an hour	several minutes	depending on the update from half an hour to several hours	depending on the update from half an hour to several hours	probably hours

Note for applet: this one is included only for the record, because it is not considered as a feasible solution.

6.8.3 Conclusion

The criteria defined above need a discussion about priority in order to make a well-founded choice for which one we should use for development. We see that the outcome for certain criteria remain stable, like CC1 and CC4. Therefore we can ignore these for the moment (although these are important as well, but show us in fact that their outcome doesn't change when we change the underlying architecture).

The highest priority was given to a most optimal use of resources, most notably the disk usage. This disk usage is defined by two criteria, CC2 (everything on the local disk except the JAWS cache) and CC7 (the JAWS cache itself). Especially CC7 is important, because for Windows XP users we saw that the JAWS cache resides in the roaming part of the user profile (and thus will be copied back and forth between server and workstation when logging on). As a result of this, we consider prototype P2 not suitable (for it has the highest JAWS cache use). Prototype P1 looks inviting with a very low value for CC7, but has another major disadvantage: a high value for CC2. When we compare the expression in CC2 for P1, P3 and P4, we see that P3 and P4 perform much better, with a relatively small increase in the value of CC7. The impact on the values for the remaining criteria CC3, CC5 and CC6 are also considered to be small and acceptable. Based on this, we declare P1 not suitable as well.

What remains are the prototypes P3 and P4. These can be compared easily, for they only differ on criterion CC2. Determining the value for this criterion is a difficult task for P4 and has therefore been abbreviated to giving an upper bound. Thanks to the introduced granularity, this upper bound will only be achieved during the first runtime. Each of the following runtimes (when updates are available) will result in a much lower value than this upper bound (unless all packages are affected, which will be a rare occasion).

P3 does not use this granularity and will therefore loose this advantage of a significant reduction of disk space usage resulting of future updates. Therefore we choose P4 over P3.

6.9 Prototype environmental testing

Having chosen P4 as our favorite prototype to continue with, we now need to determine its robustness by testing it in different environments in order to determine whether it is up to the task we envisaged it for.

JAWS provides interesting automated updating mechanisms that liberate programmers from doing a lot of administration by themselves in this particular field. Furthermore, Sun claims that JAWS updates itself with the Java Runtime Environment (JRE) in case of need. Particularly this claim together with the ability to run the program is the main subject of the test session that led to an extensive test report [JATERE] of which this section is a synthesis.

6.9.1 Environments used for testing

The following operating systems were used for testing:

- MS Windows XP Professional SP2 NL
- MS Windows Vista Ultimate NL

In both cases preconfigured images were used.

For accessing the prepared web page containing the link to the JNLP file for starting the JAWS application the following browsers were used:

- MS Internet Explorer 7
- Mozilla Firefox 2

For each of these combinations the following user account levels in Windows were used:

- Standard user (SU)
- Power user (PU)
- Local administrator (Admin)

Last but not least, the following JRE's were used to start with:

- JRE 1.3.0 initial release with separate JAWS (1.0.1_02)
- JRE 1.4.0 initial release (JAWS incorporated)
- JRE 1.5.0 initial release (JAWS incorporated)
- JRE 1.6.0 initial release (JAWS incorporated)

Taking all possibilities together, this situation gives rise to a total number of 48 test cases (4 x 3 x 2 x 2).

6.9.2 Test description

The following test scenario was executed:

- Install the required JRE [1.3 | 1.4 | 1.5 | 1.6] as Admin
- Log on as the requested user [SU | PU | Admin]
- Launch the requested browser [MSIE | MFF]
- Launch the application

Attention was paid to the following aspects:

- In case of another JRE than version 1.6, how the update process of the JRE was executed;
- How the application launched (errors, strange behavior etc).

6.9.3 Environmental issues

As described In [JAWEST], the JNLP syntax for Java 6 prescribes a change of the <j2se> tag to <java>. This had to be changed back into <j2se>, for otherwise the JNLP file is rejected by older JRE versions than 1.6.

Therefore the JNLP configuration tag for the JRE to be used became as follows:

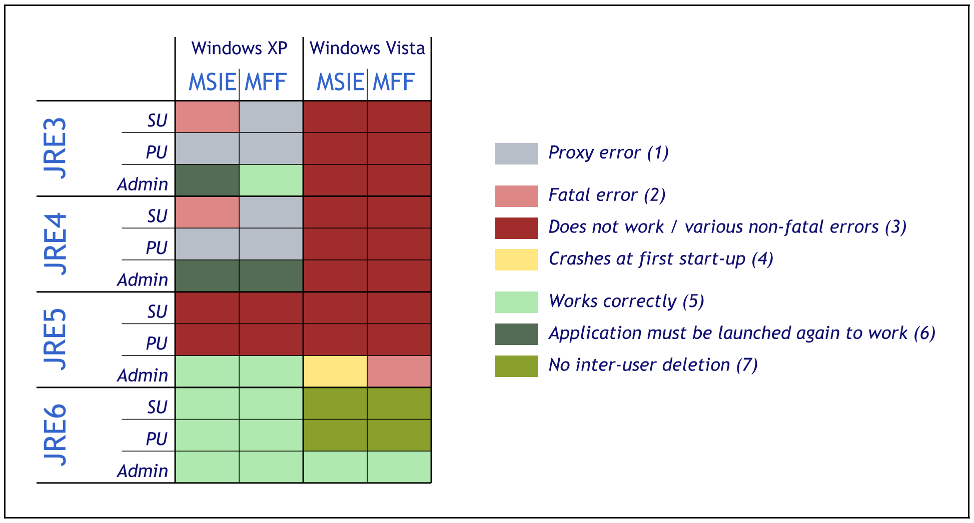
```
<j2se href="http://java.sun.com/products/autodl/j2se" version="1.6.*" />
```

Furthermore there was an issue on JAR files. JAWS works with these files and they need to be signed when we want to have unlimited access to the file system which is the case for our application. The problem with this signing is that the signatures – once the JAR file downloaded – need to be checked by JAWS. The early versions of JAWS seem to use the manifest of the JAR file for this purpose in a rather stupid way. They just scan all the lines of the manifest and take every entry as a file record. But newer versions of the JAR utility that comes with Java also put comments and directives in this manifest. Of course these "files" cannot be found in the JAR.

A simple, but rather brutal way to solve this, is to edit the existing JAR's and *delete* the manifest and all signature files (in the directory META-INF) entirely and resign the resulting JAR file.

Finally, given the fact that a proxy server is used in the environment where the tests were conducted, we had to configure this. It turned out that this was not always possible. In the test results errors caused by this particular configuration are marked separately.

6.9.4 TEST RESULTS



		Windows XP		Windows Vista	
		MSIE	MFF	MSIE	MFF
JRE3	SU	🟪	⬜	🟥	🟥
	PU	⬜	⬜	🟥	🟥
	Admin	⬛	🟩	🟥	🟥
JRE4	SU	🟪	⬜	🟥	🟥
	PU	⬜	⬜	🟥	🟥
	Admin	⬛	⬛	🟥	🟥
JRE5	SU	🟥	🟥	🟥	🟥
	PU	🟥	🟥	🟥	🟥
	Admin	🟩	🟩	🟨	🟪
JRE6	SU	🟩	🟩	🟫	🟫
	PU	🟩	🟩	🟫	🟫
	Admin	🟩	🟩	🟩	🟩

⬜	Proxy error (1)
🟪	Fatal error (2)
🟥	Does not work / various non-fatal errors (3)
🟨	Crashes at first start-up (4)
🟩	Works correctly (5)
⬛	Application must be launched again to work (6)
🟫	No inter-user deletion (7)

Figure 11: results of environmental testing

The given numbers correspond with the numbering given in the legend right of the result matrix.

1. Proxy error

For some scenario's it was not possible to configure the proxy server in such a way that JAWS executed correctly.

2. Fatal error

These were very serious situations where during the update of JRE something went seriously wrong that not only had as a consequence that the installation of the update crashed, but also that it left the Java system behind in an unusable state. The default JRE pointed to the failed installation and as a consequence Java applications could no longer run.

3. Does not work/various non-fatal errors

The following two errors occurred:

- Insufficient rights to proceed with JRE update (administrative rights needed);
- JRE update program crashes because Windows blocks access to JAR content to check signatures (this only occurred on Vista).

4. Crashes at first start-up

A special situation where the update program crashes at first which provokes a “compatibility mode” pop-up of Vista. After accepting this suggestion and launching the update again, the installation succeeds.

5. Works correctly

No problems or strange behavior detected.

6. Application must be launched again to work

After updating the JRE, the application cannot run directly and thus the user must click again on the link to launch the program (which launches correctly).

7. No inter-user deletion

Under Vista with JRE6 (where there is à priori no need for update) strange things seem to happen with file access rights. When the application is installed by user A, user B can no longer run the integral deletion utility for the shared part in the All Users profile.

6.10 Prototype comparison testing

In section 6.8.1 comparison criteria (CCs) were described in order to compare the different prototypes. But these CCs can also be applied to one prototype in different environments. This is exactly what has been done in order to get insight into the effect of different environments.

We took the prototype P4 and ran it in different environments and measured the differences. As can be seen in the table below these measurements show drastic differences between Windows XP and Windows Vista, especially for execution times.

6.10.1 Recapitulation of the comparison criteria for testing

Let’s recapitulate what CCs were identified:

- CC1: Ease of installation (measured by the minimal number of actions required by the user to complete the installation);
- CC2: Hard disk usage when having *n* users and *m* versions (measured by the number of megabytes occupied) with sp for size of the program files and su for the size of the user files;

- CC3: Installation duration (measured in seconds counting from the click on the installation link to installation completion);
- CC4: Application start up duration (measured in seconds);
- *CC5: Impact on existing code (measured by the amount of changes necessary in the existing code in order to have the application running);*
- *CC6: Ease of adaptation (measured by an estimation of the time required to create a new version or update);*
- **CC7: Duration of deletion procedure (measured in seconds).**

Note that CC5 and CC6 (in italic font) are not applicable in these tests and that CC7 (in bold font) has been added to the list for these tests because of its relevance here.

6.10.2 TESTING ENVIRONMENT

The tests were carried out using Microsoft Virtual PC 2007 with the following configuration:

- 400 MB of RAM
- Pentium-4 2.8 Ghz processor
- Prototype P4
- JRE 1.6.0_05
- Using the administrator account
- MS Windows XP SP2 NL Professional (5.1)
- MS Windows Vista NL Ultimate (6.0)

6.10.3 TEST RESULTS

	Windows XP		**Windows Vista**	
	MSIE	**MFF**	**MSIE**	**MFF**
CC1 (# actions)	2	3	3	3
CC2 (MB)	sp × m + su × (m + n)			
CC3 (seconds)	72	65	147	165
CC4 (seconds)	33	34	119	107
CC8 (seconds)	10	12	24	25

Note: CC1 is obtained by counting the number of dialogs that need a user action.

7. Development / Application

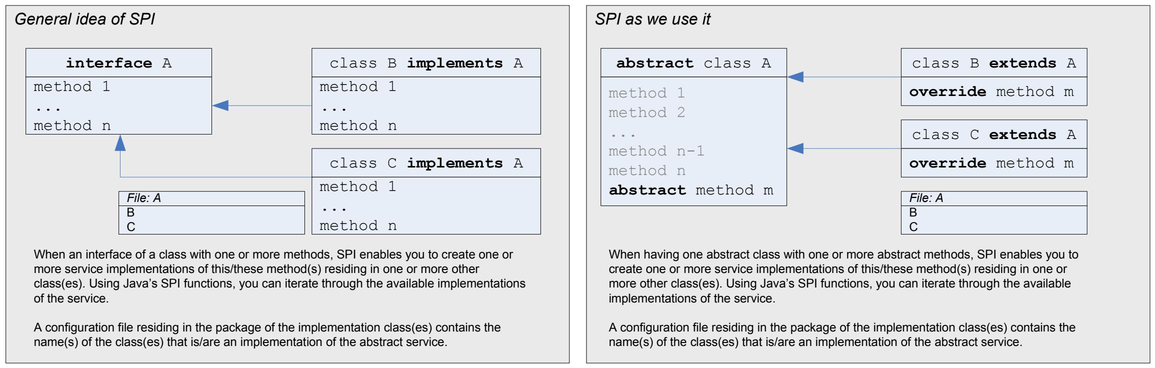
Because of confidentiality concerns, the chapters treating the development phase only cover general technical information.

We will therefore look here into a technical detail concerning a framework for native libraries. Suppose that you have a Java application that uses win32 DLL's (native libraries). In a setting of Java Web Start we saw that explicit loading is necessary. Java contains some interesting technical tools, of which we will discuss an important once here called SPI (Service Provider Interface).

7.1 Handling native libraries

Native libraries can be of a major problem when using Java Web Start. If these libraries at themselves do not use any further resources, the problem can still be solved as we saw in the 100% JAWS prototype. JAWS does support native libraries and makes them load, but when you happen to have native libraries that use resources at their turn, then it becomes potentially complicated.

A way to tackle this problem is discussed in this section that is devoted to development issues related to the application we want to deploy. The idea is to create a framework for libraries, called the "Library Provider Framework" (LPF). This framework is based on the relatively new principle called SPI (Service Provider Interface) that has been put into Java SE version 6. SPI is a technology that enables interfaces or abstract classes to be implemented by several implementations among which can be chosen at runtime. This feature is essential within the Library Provider framework, because it enables us to connect native libraries to an application using the same, generic interface.



When an interface of a class with one or more methods, SPI enables you to create one or more service implementations of this/these method(s) residing in one or more other class(es). Using Java's SPI functions, you can iterate through the available implementations of the service.

A configuration file residing in the package of the implementation class(es) contains the name(s) of the class(es) that is/are an implementation of the abstract service.

When having one abstract class with one or more abstract methods, SPI enables you to create one or more service implementations of this/these method(s) residing in one or more other class(es). Using Java's SPI functions, you can iterate through the available implementations of the service.

A configuration file residing in the package of the implementation class(es) contains the name(s) of the class(es) that is/are an implementation of the abstract service.

Figure 12: the SPI mechanism explained

Thanks to the chosen architecture, LPF forms also a solid basis for the future. New native libraries can be added with stunning ease, without touching anything of the existing code to make it work. Changes related to the application for library loading have therefore a much wider effect than "just" making it possible to deploy and run the application within the JAWS environment.

8. Development / Portal

The portal is the denotation of a small application that takes care of the interface between JAWS and the actual application to deploy. Such a small application might also be integrated into the application itself as a part of the start-up process. The task to be fulfilled by this portal is to take care that all files are in place that might be required for launching the (core) application. Here once again, as a result of imposed confidentiality, only general technical items are discussed.

8.1 Requirements

A portal that we envisage here must provide us with the means to perform actions that put files in a certain location on the user's local hard disk as well as to download necessary resources to the JAWS cache.

Furthermore the portal must provide a certain means of versioning. This versioning is the central requirement of the portal, but it can be decomposed. A portal will typically have requirements like:

- Downloading resources to the JAWS cache;
- Removing resources from the JAWS cache;
- Copying resources from the JAWS cache to an arbitrary disk location;
- Detection of presence resources on an arbitrary disk location;
- Creation of directory structures on arbitrary disk locations for storing resource versions;
- Perform integrity checks on the present resources.

8.2 Functional design

An LPF like structure can be reused for the installer or portal. Not for recovering an instance of a native library of course (for this is not applicable here), but for recovering the resources of a library. These resources are important for the installer to know about, such that it can check whether the necessary files are present at the locations where they are expected to be.

In this section the development of the installer is discussed in detail.

8.2.1 Installer workflow

The installer does not only play a role as installer, but merely as a portal application that launches, updates and installs the application when needed. The idea is that a user launches the portal application through Java Web Start. The portal application will then search within the LPF for the native libraries provided together with their resources. When these files are not present, they are downloaded at the background into the JAWS cache, copied to their correct location and finally deleted from the JAWS cache. As a last step, the portal launches the application itself. The figure below shows how the execution cycle of such a portal application could take shape.

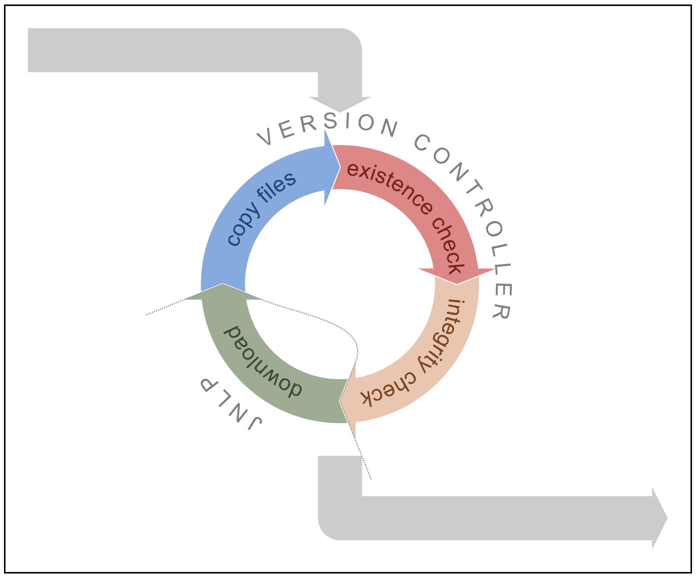


Figure 13: version controller and JNLP execution cycle

8.2.2 INTEGRITY CHECKING

As integrity checking is also an important issue, we will discuss this as well. Integrity checking can be implemented by means of creating a ZIP archive of all files in the base directory of a resource in question. This ZIP file is then temporarily saved in order to open it again, convert its contents to a byte array and calculate the MD5 hash of it. The MD5 hashing algorithm provides a 16 byte fingerprint of a given array of bytes (usually represented as a 32 characters long hexadecimal string). Even the smallest change in the composition of the files in the ZIP file will lead to a dramatic change in the outcome of the MD5 hash. In the figure below the integrity checking process is visualized. It suffices to store the MD5 hash of the file collection as it should be in a configuration file to compare it with the outcome of the algorithm on the user's disk.

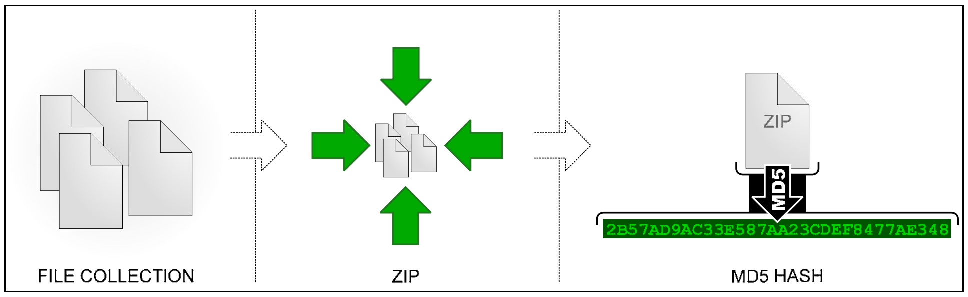


Figure 14: three step process of integrity checking

9. Conclusion and future work

Having addressed all items that made up this project, we can now pass on to the conclusion. In this section we will discuss the final result and test it once more to the initial requirements that were presented at the beginning of the project. In eight months time a lot was achieved in both research and development of a practical application. The deployment of applications, a phase that is often taken for granted as a necessary phase at the end of the software development process, is finally getting more attention and is becoming an interesting field at its own within computer science.

9.1 General application deployment

Application deployment in general is becoming more and more important. A significant increase in the importance of security obliges computer scientists to also address the problems that they create in the deployment phase of applications.

Deploying with elevated rights or even administrative rights must be avoided if possible. This has a direct influence on the architecture that is chosen in phases earlier in the software development process. We could therefore say that there is a tendency of a shift of importance of deployment to earlier phases.

With the existence of more and more control within professional organizations concerning the way computers are used — and what programs that are or can be installed — also the techniques used for deployment need serious attention. When delivering a commercial product, you should not only take the single user into account, but also network administrators who sometimes have to deploy applications over hundreds of computers at the same time.

Keeping applications up to date is an integral issue in application deployment. During the development of applications this must be taken into account. Bug fixes and the addition of features in new versions must be easily accessible by the user — and by preference should be deployed automatically. Especially internet and the more and more developing web services offer a beautiful way of supporting this important process.

Last — but certainly not least — in order to emphasize the importance of a good deployment, we see also that customers are more and more aware of potential problems with deployment. The decision to buy your product rather than the one of your competitor nowadays also depends on this issue. We may therefore conclude that negligence in the field of deployment may be harmful to a product's success.

In the light of these observations, we saw that a platform-based deployment system like Java Web Start for Java applications looks most promising. The fact that this technology was followed by Microsoft for their .NET Framework in 2006 is also a strong indicator for the potential of these systems. As we saw in the beginning of this document in section 4.4.8 where we described the phases of the deployment process, these systems tend to support deployment activities in a broader sense than almost any other deployment strategy. Whenever a pure web-based approach is impossible or impractical, these systems offer a valid alternative. Although Microsoft's ClickOnce technology only works for .NET applications, it might be interesting to give a short comparison between the two technologies with respect to key abilities that are surprisingly similar (based on [MSCLON]):

Microsoft ClickOnce	**Sun's Java Web Start**
.NET Framework applications	Java (JRE) based applications
Web launch only from MSIE [SPCOFF]	Web launch from any browser (with Java)
Caches applications in user profile	Caches applications in user profile
Uses security sandbox	Uses security sandbox
Needs pre-installed .NET Framework	Needs pre-installed JRE
Applications are admin-rights free	Applications are admin-rights free
Uses two xml-based manifest files	Uses one xml-based JNLP file

9.2 Deployment of our application

Our application is no exception to the situation described above that applies for deployment in general. Océ printers are in general used in large companies where many users have access to such a printing device. Installing the application and keeping it up to date is an essential issue. Next to the installation and update processes, printing devices should be accessible in an intuitive way. Within this project we tried to offer a solution to each of these three fields. We created a fully functional solution that is printer-centric and installs and updates itself automatically with an absolute minimum of user interaction. Network administrators only need to care about maintenance on the printing devices, significantly reducing the number of objects that require maintenance in their working environment.

9.2.1 Test to initial requirements

In chapter 3 — more precisely in section 3.2.5 — we formulated the initial four requirements after identifying what it really was all about. We saw that our application had to be able to be installed by users with a restrictive rights level. In the solution created, this is indeed the case. Users of an arbitrary rights level are able to install, update and use the application. The only marginal note that needs to be made on this point is that the JRE, the Java platform, still needs administrative rights to install. This is an unsolvable issue, but fortunately becoming less and less problematic. Most users have already a JRE installed. And in professional organizations the deployment of a JRE will beyond any doubt be much less problematic than a third party program. Having a (recent) JRE becomes more and more important, for many applications — especially in the internet domain — won't run without having a Java platform.

We also saw in the requirements that the installation should be kept simple. We followed this point rigorously by reducing the number of user actions to an absolute minimum and to provide the user with clear information in a pleasant user interface. The installation process itself only "costs" one click. The installation process is also not really presented as such, but merely as a preparation phase for the application.

The third requirement was that the impact on the source code of the application had to be kept as minimal as possible. No far-reaching changes had to be made. Only very small parts of specific code had to be touched in order to create sufficient generic access to resources such that we were able to configure JAWS in the correct way. Finally, the occasion was also used to create a generic framework for library loading. In principle, this adaptation was not necessary, but in the context of this project it was sufficiently relevant and will definitely ease future maintenance and extensions of the source code.

9.3 Gain for Océ

The gain for Océ using the Java Web Start based scenario as worked out during this project is one consisting out of many aspects. The following main issues could be identified as a significant gain with respect to the original situation before the project:

- Much larger coverage of the activities related to the deployment process of applications (main goal);
- Better integration into the preferred usage workflow by customers by offering a printer-centralized solution;
- Better support for future maintenance and extension of functionality with respect to native libraries thanks to the Library Provider Framework;
- Huge amount of documented research that can be used for many applications within the company. During this project around 20 official documents were written summing up to a total of over 450 pages of technical documentation of high value.

9.4 Future development

As for future development, there are more than sufficient issues that are still open that could be subject to future projects. This depends primarily of the scope in which one wishes to place this project, for as we saw already at the beginning, there are many issues involved that are not always clearly related to one central subject. Deployment plays the central role, but next to that the launching process and the usage workflow by customers is also important. We also discovered the need for more generic setups like the Library Provider Framework for native libraries.

For future development on short term, we could envisage for example the following items (explicitly without attempting to give a complete list):

- Extension for sharing resources within network shares;
- Interoperability of multiple installation modes;
- Mirror-based acquirement of resources (JAWS future);
- If desirable, further inquiry into applet-based approach, but this will require at first a major change in the way JRE’s function under Vista.

Especially the point of complete interoperability of multiple installation modes could be a particularly interesting item. Within this graduation project it was foreseen to tackle this item if there was sufficient time remaining. Finally, it was decided to use this time to investigate rather another method of deployment that was seen by Océ to be more interesting at this moment: the ability to create some kind of “light” version of the application to run inside an applet. The time remaining to perform this investigation only enabled us to prototype this idea, but proved our initial hypothesis that an applet — even when the environment is made suitable — is not feasible.

10. Evaluation

As a final section of this graduation report and thus to conclude this project, we will look back to the eight months spent on this project. We will discuss how things went and what things were learned. The graduation project marks the final phase of the master studies in Computer Science at the Eindhoven University of Technology, the last phase before becoming an engineer. An ultimate project to put the knowledge acquired in years into practice — and until a certain degree — to the test.

10.1 Project quality

Determining the project quality depends on many factors. And many factors that played an important role there were indeed. The graduation project offered a large amount of research that needed to be done in the field of deployment in computer science that is far from mature. In a certain sense one could characterize this project therefore as a pioneering project.

There was also a large component of practice present, which took shape in experimental prototyping and the actual development phase. I learned a lot about practical programming in Java that I knew only skin-deep before. During my studies I had to use Java several times for small programming assignments and the memories I had of Java were not always positive. I would now tend to say that Java is a programming language that is not easily accessible for beginners, but when you got through the beginners part, Java offers a huge range of very interesting and useful language constructs and tools. This project truly changed my point of view towards this language and programming environment.

The most important aspect in determining the project quality is however in my point of view the issue that within a commercial setting a solution needed to be found for a problem that was in the beginning rather vague. It is what makes this project suitable as graduation project for future engineers. Extracting what really matters and formulating and identifying explicit goals is in many cases the most critical and also most difficult phase of a project.

And last — but far from least — there was the international character of the project. Cultural differences sometimes make the situation difficult, but it is heavily reduced when compared to the benefit that can be obtained from it. France is known in Europe to have a strong identity which can be seen clearly in working environments. Performing this graduation project in an international context has been an enormous enrichment.

10.2 Project execution

The way projects are executed in France differs from the Netherlands. In the Netherlands we like in general everything to be clear with many sub goals that enable you to determine whether the original planning is still feasible and needs adjustment or not. In France — following my experience — this is different. The final goal is in general clear, but it is up to you to fill in the rest of the time. There is a lot of freedom for creativity. Excellence in the final result is highly appreciated and important.

When being used to the typical Dutch culture, this is a switch that must be made. In the beginning this required some adaptations. French colleagues are in general also different from what one finds in the Netherlands. The famous hand-shaking ritual every morning upon arrival may appear strange to most Dutch people, but it serves as a nice barrier-breaking means to get to know each other better.

Concerning the work performed, no problems were encountered. I had the opportunity to perform the necessary research in a relatively relaxed tempo with essential help from Mr. Appercel whenever needed.

The experience that I obtained in projects like OGO and more specifically the Software Engineering Project and an external internship last year enabled me to execute the important key elements of the graduation project. It proves, in my opinion, the importance of these items in the curriculum.

10.3 Special thanks

Many people were involved in making this project a reality and also in assisting to make this project a success. It would be impossible to mention them all, so I would first of all like to thank in general the personnel of Océ R&D in Créteil, who were always there to assist in case of need. Not only for direct project related subjects, but also for many, many issues alongside as a result of the international character of this project. The Netherlands and France are separated from one another by only a few hundred kilometers and one (relatively small) country (Belgium). Many destinations between Paris and the Netherlands are much closer to one another than many destinations within France itself. However, the differences between the two countries are omnipresent. Of course there are no enormous differences on the cultural and administrative level, but a huge collection of small differences finally sum up to quite a heavy load.

Next to these general thanks I would anyhow like to thank a number of people who were of particular help during this project:

- Ad Aerts — my academic coordinator of the Eindhoven University of Technology with whom I had regular contact and who was always there for support and advice when needed;
- Stéphane Appercel — my project coordinator at Océ R&D in Créteil, a software architect from whom I learned a lot on software development within the Java-based world;
- Patrick Battistini — project coach at Océ R&D Créteil for many organizational issues;
- Stéphane Binétruy — developer at Océ R&D Créteil who is a living gold mine on programming issues;
- Vincent Merk — professional French teacher at the CTT of the Eindhoven University of Technology who enabled me to drastically increase my language capabilities in French and was of very valuable support regarding many administrative issues.

A. References

For consultation convenience, the references have been categorized in internal documentation (Océ R&D internal documentation) and external documentation (documentation from outside of Océ R&D consulted for this document).

A.1 Internal documentation

[INQSUB]	An inquiry into submitters C.J.M.F.C. Raemaekers - Océ R&D
[MSDEPP]	Microsoft deployment practices C.J.M.F.C. Raemaekers - Océ R&D
[MIGISS]	Migration issues C.J.M.F.C. Raemaekers - Océ R&D
[JAUPDN]	Java 6 Update N C.J.M.F.C. Raemaekers - Océ R&D
[JAWEST]	Java Web Start C.J.M.F.C. Raemaekers - Océ R&D
[INTJSP]	Introduction into JSP C.J.M.F.C. Raemaekers - Océ R&D
[PUSCAN]	Application component analysis C.J.M.F.C. Raemaekers - Océ R&D
[JADESC]	Java Web Start deployment scenarios C.J.M.F.C. Raemaekers - Océ R&D
[PROTOG]	Prototyping C.J.M.F.C. Raemaekers - Océ R&D
[JATERE]	Java Web Start test report C.J.M.F.C. Raemaekers - Océ R&D
[DEVREP]	Development report C.J.M.F.C. Raemaekers - Océ R&D
[INCOIS]	Installation compatibility issues C.J.M.F.C. Raemaekers - Océ R&D
[FUMMIN]	Follow-up on multi-mode installations C.J.M.F.C. Raemaekers - Océ R&D
[PSCOCH]	Application compatibility changes C.J.M.F.C. Raemaekers - Océ R&D
[CPJACO]	Application changes for Java Web Start compliance C.J.M.F.C. Raemaekers - Océ R&D
[PUACSO]	Application accessibility solutions C.J.M.F.C. Raemaekers - Océ R&D
[PSNALI]	Application native library integration C.J.M.F.C. Raemaekers - Océ R&D
[PROCOM]	Product comparison C.J.M.F.C. Raemaekers - Océ R&D

A.2 External documentation

[ATE]	JAX-WS 2.1 Ask the Experts session http://java.sun.com/developer/community/askxprt/2007/jl0226.html
[IBM-TL]	JSP best practices: Intro to taglibs http://www.ibm.com/developerworks/java/library/j-jsp07233.html
[J6JAXWS]	Introducing JAX-WS 2.0 With the Java SE 6 Platform, Part 1 http://java.sun.com/developer/technicalArticles/J2SE/jax_ws_2/
[J6U4JAXWS]	JAX-WS 2.1 and JAXB 2.1 is available in JDK 6 Update 4 release http://weblogs.java.net/blog/ramapulavarthi/archive/2008/01/jaxws_21_and_ja.html
[JAI]	Java Advanced Imaging API Web Start Releases https://jai-webstart.dev.java.net/releases.html
[JAI-JAWS]	jai-webstart https://jai-webstart.dev.java.net/
[JAVA-ON-VISTA]	Chet Haase Java Weblog: Vista and IE7 sandboxes, the Unfixable http://weblogs.java.net/blog/chet/archive/2006/10/java_on_vista_y.html
[JAVA-TL]	Tag Libraries Tutorial http://java.sun.com/products/jsp/tutorial/TagLibrariesTOC.html
[JAVA-VISTA-NOTES]	Java 6 Release Notes section Java Vista Notes http://java.sun.com/javase/6/webnotes/#windowsvista
[JNLP-API]	JNLP API Reference http://java.sun.com/products/javawebstart/docs/javadoc/index.html
[JSTL-REF]	JSTL Reference http://java.sun.com/products/jsp/jstl/1.1/docs/tlddocs/index.html
[TL-TUT]	<taglib:tutorial/> http://www.orionserver.com/docs/tutorials/taglibs/
[TQB]	JAX-WS 2.1 now part of Java 1.6.0_04 http://blogs.sun.com/quinn/entry/jax_ws_2_1_now
[URL-DEPLOY]	Deploying Software with JNLP and Java Web Start http://java.sun.com/developer/technicalArticles/Programming/jnlp/
[CFSODT]	A Characterization Framework for Software Deployment Technologies Carzaniga, Fugetta, Hall, Heimbigner, van der Hoek, Wolf Technical Report University of Colorado CU-CS-857-98 (1998)
[MSCLON]	Introduction to ClickOnce Deployment http://msdn.microsoft.com/en-us/vbasic/ms789088.aspx
[SPCOFF]	MSDN Blogs: ClickOnce and FireFox http://blogs.msdn.com/saurabh/archive/2006/03/02/541988.aspx
[JAWIVI]	Release notes for the next-generation Java Plug-In Technology https://jdk6.dev.java.net/plugin2/
[CORBRO]	Océ Corporate Brochure 2007 – “Beyond the Ordinary” Océ PR http://www.oce.com/en/about/Profile/download.htm

B. Source code JAWS sample

This appendix part covers a minimal sample application using JAWS as indicated in section 4.8.

B.1 Extract a file from a JAR and copy it to the local disk

An important detail is how to get a resource (in our case an executable) out of a secondary JAR file and copy it to the destination folder. This operation is not that difficult, it is more a game of knowing which type of stream one needs to use and to write it back into a file. The code snippet just below shows how to do so:

```
try
{
  InputStream stream =
             getClass().getResourceAsStream("isAdmin.exe");
  FileOutputStream output = new FileOutputStream(usrprofile
                            + "\\.TestApp\\isAdmin.exe");
  b = new byte[1000];
  int v = 0;
  while((v = stream.read(b)) > -1)
  {
    output.write(b,0,v);
  }
  stream.close();
  output.flush();
  output.close();
}
catch(Exception ex) {System.out.println(ex.getMessage());}
```

Using the getResourceAdStream method we can extract a file from the JAR file. Note that the JAR file itself doesn't need to be specified, Java sorts out by itself where to look. Once the stream is loaded, we write it into the destination file in chunks of 1000 bytes each (there is no particular reason why this is 1000 here, according to the file size of the file to be copied, you might wish to use a different data chunk size). This entire procedure needs also to be put in a try-catch construct.

B.2 Configuring the JNLP file

The JNLP file needs first of all to contain a correct pointer to the codebase; that is the location where to look for the JAR files and the JNLP file itself. This is done using the jnlp tag in the following way:

```
<jnlp spec="1.0+" codebase="http://test.oce.com/testapp"
href="TestApp.jnlp">
```

Furthermore, we need to set the `<all-permissions/>` tag in the security section, otherwise we will not be able to do anything at all with disk access and file copying. And last - but not least - we need to specify which JAR files are contained by the application in the resources section:

```
<jar href="TestApp.jar"/>
<jar href="data.jar"/>
```

```
<java version="1.6+" href="http://java.sun.com/products/autodl/j2se"/>
```

Note that we declare two JAR files here (TestApp.jar for the application itself and data.jar for the executable the program should extract). Furthermore the desired version of the JRE is mentioned (here: 1.6 or higher).

B.3 JAR Signing and certificate creation

JAR file signing is very similar to most signing operations in Computer Science. We need to create a so-called key store that creates a certificate with a public and private key. The private key must remain at our side in order to sign the JAR's, the public key is available for clients to check our signatures.

The idea of this key system is based on the fact that there are two separate keys: one for encoding and one for decoding. The one for encoding is in this case the private key (which should remain secret and which is almost impossible to obtain by cracking) and the one for decoding is the public key here. This public key can be transferred to anyone who wishes to access the encoded information.

Java gives us the tools necessary to perform all these operations. Here is a step-by-step overview of this procedure.

1. Key generation

```
keytool -genkey -alias signLegal -keystore ocetest
```

This creates a key store (public and private keys) with the name 'ocetest'. One can choose a more convenient name here if desired. The keytool will ask you several straightforward questions like the name of the person generating the keys, the company, its location, etc. The program also lets you choose two passwords that should be chosen with attention.

2. JAR signing

Now we have the key store, we can actually sign the JAR files. This is done in a straightforward way like this:

```
jarsigner -keystore ocetest -signedjar sdata.jar data.jar signLegal
```

Note that we use the same key store ('ocetest') like we did in step 1. It is a good idea to use a different name for the signed store than the original one if you need to keep the original for other purposes. During signing you need to provide the passwords you chose during the key generation procedure.

All JAR files must be signed in order to have the application working correctly.

3. Certificate creation for public key retrieval

If a public key certificate is desired for consultation of clients, this can be created using the following command:

```
keytool -export -keystore ocetest -alias signLegal -file ocetest.cer
```

This creates a file ocetest.cer that you could put on your web server to be consulted by clients in order to retrieve the public key. For JAWS however, this is not obligatory.

B.4 Program code:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;

public class TestApp
{
  JFrame fraMain;
  JButton cmdClose;
  JButton cmdCreateDir;
  JButton cmdCopyFile;
  JLabel lblInfo;
  JPanel pnlMain;

  public static void main(String[] args)
  {
    System.out.println("Program init succesfully executed...");
    new TestApp();
  }

  TestApp()
  {
    fraMain = new JFrame("Test Application");
    fraMain.setSize(600,400);
    pnlMain = new JPanel();
    cmdClose = new JButton("Quit");
    cmdCreateDir = new JButton("Create directory
                               %USERPROFILE%\\.TestApp");
    cmdCopyFile = new JButton("Copy file isAdmin.exe to
                              %USERPROFILE%\\.TestApp");
    lblInfo = new JLabel("THIS CAPTION HAS BEEN CHANGED -- NEW VERSION");
    pnlMain.add(lblInfo);
    pnlMain.add(cmdClose);
    pnlMain.add(cmdCreateDir);
    pnlMain.add(cmdCopyFile);

    cmdClose.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        System.out.println("...program succesfully terminated.");
        System.exit(0);
      }
    });

    cmdCreateDir.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        String temp = "";
        Runtime runtime = Runtime.getRuntime();
        try
        {
          Process process = runtime.exec("cmd.exe /c set USERPROFILE");
          BufferedReader in = new BufferedReader(new
                             InputStreamReader(process.getInputStream()));
          temp = in.readLine();
          temp = temp.substring(12);
        }
        catch(Exception ex) { System.out.println("error" +
                               ex.getMessage());}

        boolean success = (new File(temp + "\\.TestApp")).mkdir();
```

```
       if (!success) System.out.println("!! --> error making directory");
        else System.out.println("...directory \""+temp+"\\.TestApp\"
                          created...");
      }
    });

    cmdCopyFile.addActionListener(new ActionListener()
    {
      public void actionPerformed(ActionEvent e)
      {
        byte b[];
        String temp = "";

        Runtime runtime = Runtime.getRuntime();
        try
        {
          Process process = runtime.exec("cmd.exe /c set USERPROFILE");
          BufferedReader in = new BufferedReader(new
                             InputStreamReader(process.getInputStream()));
          temp = in.readLine();
          temp = temp.substring(12);
        }
        catch(Exception ex) { System.out.println("error" +
                                  ex.getMessage());}

        try
        {
          InputStream stream = getClass().getResourceAsStream
                                            ("isAdmin.exe");
          FileOutputStream output = new FileOutputStream(temp +
                                   "\\.TestApp\\isAdmin.exe");
          b = new byte[1000];
          int v = 0;
          while((v = stream.read(b)) > -1)
          {
            output.write(b,0,v);
          }
          stream.close();
          output.flush();
          output.close();
        }
        catch(Exception ex) { System.out.println("error" +
                                     ex.getMessage());}

        System.out.println("...file isAdmin.exe copied to " + temp +
                          "\\.TestApp directory...");
      }
    });

    fraMain.getContentPane().add(pnlMain);
    fraMain.setVisible(true);
  }

}
```

B.5 JNLP Code:

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://test.oce.com/testapp"
href="TestApp.jnlp">
  <information>
     <title>Test Application</title>
      <vendor>Oce</vendor>
      <description>Test Application for JAWS</description>
      <homepage href="http://www.oce.com"/>
      <description kind="short">This application tries to create a subdir
       in the user profile directory and put an executable in it.
       </description>
      <offline-allowed/>
   </information>
   <security>
       <all-permissions/>
   </security>
   <resources>
       <jar href="TestApp.jar"/>
       <jar href="data.jar"/>
       <java version="1.6+"
                      href="http://java.sun.com/products/autodl/j2se"/>
   </resources>
   <application-desc main-class="TestApp"/>
</jnlp>
```

C. JSP and Tag Libraries

This appendix describes JSP and Tag Libraries in depth as discussed in section 4.11.

C.1 Scripting tags

The general idea of all server-side scripting languages is that programming code can be interleaved with the standard HTML output. In order to make sure that the interpreter can distinguish between HTML code and scripting code, the scripting code must be encapsulated between special tags. The following table denotes the different encapsulation tags for the three main players:

	Opening tag	**Closing tag**	**Usage**
PHP	<?php	?>	general
	<?	?>	general
	<?=	?>	direct output
ASP	<%	%>	general
	<%=	%>	direct output
JSP	<%	%>	normal code
	<%!	%>	declarative code
	<%=	%>	direct output
	<%@	%>	include
	<%--	--%>	comment

PHP and ASP only make a difference between general tags and direct output tags. What this means is illustrated by the following examples:

```
<?php echo "Hello World" ?>
<?= "Hello World" ?>
```

```
<% Response.Write("Hello World") %>
<%= "Hello World" %>
```

The first two lines are in PHP, the last two lines in ASP. All four lines give the same result. In JSP the situation is somewhat more complicated. JSP makes a difference between normal code and declarative code. With declarative code we mean the declaration of variables and objects with their methods. The normal code is the actual usage of this code. These two kinds of usage must be separated; JSP does not allow them to be mixed like in PHP or ASP. The same holds for including pieces of code residing in other files. PHP and ASP have "include" constructs for this that can be used in the general tags. In JSP these declarations must be made using the special include tag <%@ … %>.

C.2 Output to HTML

The output to HTML is always done using a dedicated function or object. This principle holds for all three languages discussed here:

	Output to HTML
PHP	echo “my text”
ASP	Response.Write(“my text”)
JSP	out.println(“my text”)

C.3 Basic constructs

JSP likes to distinguish between different code types. Declarations must be separated from normal code and includes. Consider the following example:

```
 1 <%!
 2      // JSP Sample
 3
 4      String makeHeader(String strTitle)
 5      {
 6              return "<html>\n\t<head>\n\t\t<title>" + strTitle +
 7                      "</title>\n\t</head>\n\t<body>";
 8      }
 9
10      String makeFooter()
11      {
12              return "\n\t</body>\n</html>";
13      }
14 %>
15
16 <%
17      out.println(makeHeader("JSP Test"));
18
19      out.println("\t\t<p>JSP output <br>\n\t\t" +
20              "The current date is: " + new java.util.Date() + "</p>");
21 %>
22
23 <%= makeFooter() %>
```

If you want to use functions in JSP, you need to declare them in a declarative scope using the <%! … %> encapsulation. You see this in the example on lines 1 to 14.

In contrast with standard Java, you can define functions here that do not belong to a class (although you can also create classes of course). Using the **return** keyword you can then pass the return value.

In the non-declarative part (lines 16 to 21) we actually use the functions we defined in the first part. Notice that this part is encapsulated in “standard” tags <% … %>. There are three calls made using the out.println(…) command. This actually prints the result into the web page. Make sure to use the **out** variable and not to follow the standard Java way for the command line which is **System.out.println**. This last one will not raise errors, because it prints the arguments passed into the server log.

The usage of standard Java objects is supported by JSP as can be seen on line 20 where we make a call to the java.util.Date object to retrieve the current date.

In order to finalize our example, on line 23 an example is given of the direct output encapsulation. Notice that the out.println does not appear here.

C.4 JSP Tag libraries and JSTL

Next to the similar principles that can be found in PHP and ASP, JSP has a special feature that is used extensively: tag libraries.

In this paragraph we will briefly look into this technology in order to get a basic overview of it, without the intention of presenting a fully covered discussion here. The tag libraries description is heavily inspired by the document [IBM-TL].

Tag libraries are the result of the wish of many web designers to be able to define their own HTML (or XML) tags that enable them to add functionality to web pages. The idea of Sun was to keep web sites in their original form; that is in XML notation. When you look into most ASP, PHP or JSP files, you will discover large amounts of HTML, mixed with the corresponding scripting language.

In order to conform to this idea, Sun came up with a mechanism where developers are able to create a so-called TLD, an abbreviation for Tag Library Descriptor. This is an XML file where you must define your own tags and where you can specify the behavior of that particular tag by linking to a Java class file. Below you see an example of such a file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE taglib
    PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2/EN"
           "http://java.sun.com/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>
    <tlib-version>1.0</tlib-version>
    <jsp-version>1.2</jsp-version>
    <short-name>tools</short-name>
    <uri>http://www.my_domain/tools</uri>

    <tag>
      <name>my_tag</name>
      <tag-class>my_class_implementing_the_tag</tag-class>
      <body-content>empty</body-content>
    </tag>
</taglib>
```

This file begins with an XML descriptor tag, followed by a DOCTYPE declaration. After that, the actual file begins. After some initializing tags, the actual tag description starts with the <tag> tag. There you can specify the name of the tag, as well as the class implementing the tag. This tag does not require a body.

Then, if we want to use this TLD in our JSP file, we only need to include the following declaration:

```
<%@ taglib prefix="t" uri="http://www.my_domain/tools" %>
```

Note the prefix that we define here “t”. Now we are ready for using the tag called “my_tag” in the rest of the JSP file. This is done as follows:

```
<t:my_tag />
```

Of course this tag can be used everywhere in the HTML/JSP file where we want. The only thing that needs to be done, is writing a Java class file that implements the behavior of this tag.

A huge advantage of this approach is that developers can first create all classes with the according TLD file. Then web page creators can use the new tags in their HTML as they are used to do with normal HTML/XML tags. The resulting file will eventually look much more like an HTML document and remains in fact a valid XML file, with no more interleaved Java code.

C.5 Tomcat configuration for using a TLD

The best way of using TLD files is to go to the Tomcat installation directory and browse to webapps/ROOT/WEB-INF. If not already present, you should create two subdirectories:

- tld: for the storage of TLD files;
- classes: for the storage of the accompanying java class files.

Using a tag library declaration as stated in section 4.11.4 in a JSP file suffices from that point on to actually use the tag library.

C.6 Creation of Java classes for a TLD

For each tag you must write a separate class file. You should begin this file with a package declaration. The class file must then be put in the same relative location in the “classes” subfolder. For example, if you declare a package com.oce.jaws, then you should put the .class-file into the webapps/ROOT/WEB-INF/com/oce/jaws folder.

After the package declaration you need two imports for using Java for JSP:

```
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;
```

These imports enable you to access functions specific for JSP. The rest of the class file follows your declaration of the tag in the TLD file. So first we declare our class as an extension to the TagSupport class we just imported:

```
public class Footer extends TagSupport
```

Following this declaration, you declare the class with at least the following two methods:

```
public int doStartTag() throws JspException
{
        try
        {
                pageContext.getOut().print("Hello World");
        }
        catch (java.io.IOException e)
        {
                throw new JspTagException("MyTag: "+e.getMessage());
        }
        return SKIP_BODY;
}
```

```
public int doEndTag()
{
        return EVAL_PAGE;
}
```

The doStartTag() procedure is executed when the opening tag is encountered, the doEndTag() procedure is executed when the closing tag is encountered. The example given here prints “Hello World” in place of the tag and does no analysis of the body (the space between the begin and end tag).

When you want to pass attributes with your tag, you need to declare them both in the TLD file and also in the class file. In the Java class file this is done by concatenating “get” and “set” in front on the attribute name. The attribute name then starts with a capital letter (regardless how you spell the attribute in your JSP files). Suppose we have the attribute “image”, then this would become as follows:

```
public void setImage(String str)  {   strImage = str;              }
public String getImage()          {   return strImage;             }
```

The variable strImage should be declared at class level such that you can use its value in the doStartTag() procedure. The attribute methods are guaranteed to be executed before the doStartTag() procedure, so you don’t need to worry about the initialization of your variables.

C.7 JSTL

JSTL is the JSP Standard Tag Library that comes with the Tomcat web server. JSTL supports the most common functions you would need when you start using a tag library approach. In the [JSTL-REF] you can find the complete description of all available tags that are categorized into 5 categories:

- Core library;
- Formatting library;
- SQL library;
- XML library;
- Functions library.

Below you find an example of the core-function “if”:

```
<html>
  <head><title>JSTL Demo</title></head>

  <body>

  <% String s = "test"; %>

  <c:if test="${s == 'images'}">
    <p>Test succeeded</p>
  </c:if>

  </body>
</html>
```

Notice that the tag library declaration is missing. This is due to the fact that JSTL is by default enabled on Tomcat web servers. Extensive information about the JSTL and tag libraries can be found in [TL-TUT] and [JAVA-TL] which have been consulted as well for this section. Also [INTJSP] can be consulted for further details.

D. Definitions and abbreviations

This appendix holds the definitions and abbreviations that are used throughout the document.

D.1 Definitions

Administrator	Highest rights level of a user on an operating system
Browser	Program that enables a user to visualize web pages and to navigate over the inter- or intranet
Bug	Popular term for an “error” or “fault” in software
Eclipse	Java IDE
Java	Sun’s Java™ programming language
JavaScript	Scripting language based on Java used on web pages
Malware	Malicious software, software harming your computer and/or your files, usually in a concealed way
Media	The material to print on (usually paper)
Netbeans	Java IDE
Océ	Océ Print Logic Technologies in Créteil (France)
Plot file	A file containing print data (for example a PostScript file)
Submitter	Application that interprets documents in order that they can be sent to a printing device
Sun	Company that developed Java and delivers the JRE
Windows	Microsoft Windows operating system

D.2 List of abbreviations

AD	Active Directory
Admin	Administrator
AJAX	Asynchronous Javascript And XML
API	Application Programming Interface
App	Application
ASP	Active Server Pages (server side scripting language from MS)
AUI	Administration User Interface (administration interface on a printing device)
CC	Comparison criterion
CD	Compact Disc
CMD	Command line (command prompt)
GPO	Group Policy Object
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hyper Text Transfer Protocol (default web protocol)
IDE	Integrated Development Environment
IIS	Internet Information Services (Microsoft’s web server)
IT-DEP(T)	Information Technology Department (the department of a company in charge of IT related activities)
JAI	Java Advanced Imaging (extension library for imagery)

JAR	Java ARchive (collection of files, in fact a zip file)
JAWS	Java Web Start
JAX-WS	Java Extension for Web Services
JNLP	Java Network Launching Protocol (in XML format)
JRE	Java Runtime Environment
JSP	Java Server Pages (server side scripting language)
JSTL	JSP Standard Tag Library
LPF	Library Provider Framework
MFF	Mozilla FireFox (web browser)
MIME	Multipurpose Internet Mail Extensions (file type definition descriptor)
MS	Microsoft
MSI	Microsoft Installer archive (used by Windows Installer)
MSIE	Microsoft Internet Explorer (web browser)
NL	Nederlands (Dutch)
NS	Netscape Navigator (web browser)
OCX	OLE Component eXtension
OLE	Object Linking and Embedding (Microsoft's Component Object Model)
OS	Operating System
OU	Organizational Unit (group of objects within the AD)
P1	Prototype version 1
P2	Prototype version 2
P3	Prototype version 3
P4	Prototype version 4
PC	Personal Computer
PDF	Portable Document Format (Adobe document format)
PHP	PHP Hypertext Processor (server side scripting language)
PU	Power user (user rights level)
R&D	Océ's Research and Development department
SA	Stand alone (application fully running on a client)
SDK	Software Development Kit
SP2	Service Pack 2
SPI	Service Provider Interface
SQL	Standard Query Language (database query language)
TLD	Tag Library Descriptor
TU/e	Eindhoven University of Technology
UI	User Interface
URL	Unified Resource Locator (location identifier)
Vista	Microsoft Windows Vista
WFPS	Wide Format Printing System
Win	Windows
WPD	Windows Printer Driver
WYSIWYG	What you see is what you get
WYSIWYP	What you see is what you print
XML	eXtensible Markup Language (data file format standard)
XP	Microsoft Windows XP